RTI Connext Java API

Generated by Doxygen 1.9.3

Chapter 1

RTI Connext

Core Libraries and Utilities

Real-Time Innovations, Inc.

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

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

1.1 Available Documentation.

The documentation for this release is provided in two forms: the API Reference HTML documentation and PDF documents. If you are new to RTI Connext, the **Documentation Roadmap** (p. 154) will provide direction on how to learn about this product.

1.1.1 The documents for the Core Libraries and Utilities are:

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

1.1.2 The API Reference HTML documentation contains:

- **RTI Connext DDS API Reference** (p. 157) - The RTI Connext API reference.
- **Additional RTI Connext Communication Patterns** (p. 160) - RTI Connext APIs for additional communication patterns
- **Programming How-To's** (p. 161) - Describes and shows the common tasks done using the API.

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

1.2 Feedback and Support for this Release.

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

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

Module Documentation

6.1 RTI Connext Exceptions

RTI Connext return-code exceptions.

RTI Connext return-code exceptions.

The request-reply API reports internal DDS errors as exceptions. Each RTI Connext return code maps to a exception.

Note that RETCODE_TIMEOUT and RETCODE_NO_DATA are not thrown in the request-reply API, because they are not considered errors. For example in case of no data or timeouts operations may return empty containers (e.g. **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(int)** (p. 1573)) or false (e.g. **com.rti.connext.requestreply.Requester<TReq,TRep>.waitForReplies(int,Duration_t)** (p. 1580)).

See also

Return Codes (p. 260)

Error handling example (p. 153)

6.2 Clock Selection

APIs related to clock selection.

APIs related to clock selection.

RTI Connext uses clocks to measure time and generate timestamps.

The middleware uses two clocks, an internal clock and an external clock. The internal clock is used to measure time and handles all timing in the middleware. The external clock is used solely to generate timestamps, such as the source timestamp and the reception timestamp, in addition to providing the time given by **com.rti.dds.domain.DomainParticipant.get_current_time** (p. 732).

6.2.1 Available Clocks

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

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

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

6.2.2 Clock Selection Strategy

By default, both the internal and external clocks use the real-time clock. If you want your application to be robust to changes in the system time, you may use the monotonic clock as the internal clock, and leave the system clock as the external clock. Note, however, that this may slightly diminish performance in that both the send and receive paths may need to obtain times from both clocks. Since the monotonic clock is not available on all architectures, you may want to specify "monotonic,realtime" for the internal_clock (see the table below). By doing so, the middleware will attempt to use the monotonic clock if available, and will fall back to the realtime clock if the monotonic clock is not available.

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

6.2.3 Configuring Clock Selection

To configure the clock selection, use the **PROPERTY** (p. 248) QoS policy associated with the **com.rti.dds.domain.↩ DomainParticipant** (p. 670).

See also

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1438)

The following table lists the supported clock selection properties.

Table 6.1 ***Clock Selection Properties***

Property	Description
dds.clock.external_clock	Comma-delimited list of clocks to use for the external clock, in the order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"
dds.clock.internal_clock	Comma-delimited list of clocks to use for the internal clock, in the order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"

6.3 Domain Module

Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.

Modules

- **DomainParticipantFactory**

  com.rti.dds.domain.DomainParticipantFactory *(p. 761) entity and associated elements*
- **DomainParticipants**

  com.rti.dds.domain.DomainParticipant *(p. 670) entity and associated elements*
- **Built-in Topics**

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

6.3.1 Detailed Description

Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.

6.4 DomainParticipantFactory

com.rti.dds.domain.DomainParticipantFactory (p. 761) entity and associated elements

Classes

- class **DomainParticipantFactory**

  *<<**singleton**>> (p. 156) <<**interface**>> (p. 156) Allows creation and destruction of **com.rti.dds.domain.Domain↩ Participant** (p. 670) objects.*
- class **DomainParticipantFactoryQos**

  *QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).*

Variables

- static final **DomainParticipantQos PARTICIPANT_QOS_DEFAULT**

  *Special value for creating a **DomainParticipant** (p. 670) with default QoS.*
- static **DomainParticipantFactory TheParticipantFactory** = create_singletonI()

  *Can be used as an alias for the singleton factory returned by the operation **com.rti.dds.domain.DomainParticipant↩ Factory.get_instance** (p. 764).*
- static **DomainParticipantConfigParams_t PARTICIPANT_CONFIG_PARAMS_DEFAULT**

  *Special value for creating a **com.rti.dds.domain.DomainParticipant** (p. 670) from configuration using default parameters.*

6.4.1 Detailed Description

com.rti.dds.domain.DomainParticipantFactory (p. 761) entity and associated elements

6.4.2 Variable Documentation

6.4.2.1 PARTICIPANT_QOS_DEFAULT

```
final DomainParticipantQos PARTICIPANT_QOS_DEFAULT [static]
```

Initial value:

```
=
    new DomainParticipantQos()
```

Special value for creating a **DomainParticipant** (p. 670) with default QoS.

When used in **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 765), this special value is used to indicate that the **com.rti.dds.domain.DomainParticipant** (p. 670) should be created with the default **com.rti.dds.domain.DomainParticipant** (p. 670) QoS by means of the operation **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos()** (p. 767) and using the resulting QoS to create the **com.rti.dds.domain.DomainParticipant** (p. 670).

When used in **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768) operation had never been called.

When used in **com.rti.dds.domain.DomainParticipant.set_qos** (p. 714), this special value is used to indicate that the QoS of the **com.rti.dds.domain.DomainParticipant** (p. 670) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) that the **com.rti.dds.domain.DomainParticipant** (p. 670) belongs to.

RTI Connext treats this special value as a constant.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values from the **DomainParticipant** (p. 670) factory; for this purpose, use **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos** (p. 767).

See also

NDDS_DISCOVERY_PEERS (p. 222)

com.rti.dds.domain.DomainParticipantFactory.create_participant() (p. 765)

com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos() (p. 768)

com.rti.dds.domain.DomainParticipant.set_qos() (p. 714)

6.4.2.2 TheParticipantFactory

```
DomainParticipantFactory TheParticipantFactory = create_singletonI()  [static]
```

Can be used as an alias for the singleton factory returned by the operation **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764).

See also

com.rti.dds.domain.DomainParticipantFactory.get_instance (p. 764)

Referenced by **DomainParticipantFactory.finalize_instance()**, and **DomainParticipantFactory.get_instance()**.

6.4.2.3 PARTICIPANT_CONFIG_PARAMS_DEFAULT

```
DomainParticipantConfigParams_t PARTICIPANT_CONFIG_PARAMS_DEFAULT  [static]
```

Initial value:

```
=
            new DomainParticipantConfigParams_t()
```

Special value for creating a **com.rti.dds.domain.DomainParticipant** (p. 670) from configuration using default parameters.

This value can be used only in **com.rti.dds.domain.DomainParticipantFactory.create_participant_from_config_w_params** (p. 785) and indicates that the **com.rti.dds.domain.DomainParticipant** (p. 670) must be created applying the information defined in the participant configuration. That is, the domain ID, participant entity name, and QoS profiles for all the entities will be retrieved from the configuration.

RTI Connext treats this special value as a constant.

See also

com.rti.dds.infrastructure.DomainParticipantConfigParams_t (p. 757)

com.rti.dds.domain.DomainParticipantFactory.create_participant_from_config_w_params (p. 785)

6.5 DomainParticipants

com.rti.dds.domain.DomainParticipant (p. 670) entity and associated elements

Classes

- interface **DomainParticipant**

  *<<**interface**>> (p. 156) Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 669) objects.*
- class **DomainParticipantAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- interface **DomainParticipantListener**

  *<<**interface**>> (p. 156) Listener for participant status.*
- class **DomainParticipantProtocolStatus**

  *<<**extension**>> (p. 155) The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.*
- class **DomainParticipantQos**

  *QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 670) entity.*

Variables

- static final **TopicQos** **TOPIC_QOS_DEFAULT** = new **TopicQos**()

  *Special value for creating a **com.rti.dds.topic.Topic** (p. 1807) with default QoS.*
- static final **TopicQos** **TOPIC_QOS_PRINT_ALL** = new **TopicQos**()

  *Special value which can be supplied to **com.rti.dds.topic.TopicQos.toString(TopicQos baseQos, QosPrintFormat format)** (p. 1826) indicating that all of the QoS should be printed.*
- static final **PublisherQos** **PUBLISHER_QOS_DEFAULT** = new **PublisherQos**()

  *Special value for creating a **com.rti.dds.publication.Publisher** (p. 1466) with default QoS.*
- static final **PublisherQos** **PUBLISHER_QOS_PRINT_ALL** = new **PublisherQos**()

  *Special value which can be supplied to **com.rti.dds.publication.PublisherQos.toString(PublisherQos baseQos, QosPrintFormat format)** (p. 1491) indicating that all of the QoS should be printed.*
- static final **SubscriberQos** **SUBSCRIBER_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.subscription.SubscriberQos.toString(SubscriberQos baseQos, QosPrintFormat format)** (p. 1758) indicating that all of the QoS should be printed.*
- static final **SubscriberQos** **SUBSCRIBER_QOS_DEFAULT**

  *Special value for creating a **com.rti.dds.subscription.Subscriber** (p. 1730) with default QoS.*
- static final **DomainParticipantQos** **DOMAINPARTICIPANT_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.domain.DomainParticipantQos.toString(DomainParticipantQos baseQos, QosPrintFormat format)** (p. 798) indicating that all of the QoS should be printed.*
- static final **DomainParticipantFactoryQos** **DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.domain.DomainParticipantFactoryQos.toString(DomainParticipantFactoryQos baseQos, QosPrintFormat format)** (p. 789) indicating that all of the QoS should be printed.*
- static final **FlowControllerProperty_t** **FLOW_CONTROLLER_PROPERTY_DEFAULT**

  *<<**extension**>> (p. 155) Special value for creating a **com.rti.dds.publication.FlowController** (p. 1055) with default property.*
- static final String **SQLFILTER_NAME**

  *<<**extension**>> (p. 155) The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*
- static final String **STRINGMATCHFILTER_NAME**

  *<<**extension**>> (p. 155) The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

6.5.1 Detailed Description

com.rti.dds.domain.DomainParticipant (p. 670) entity and associated elements

6.5.2 Variable Documentation

6.5.2.1 TOPIC_QOS_DEFAULT

```
final TopicQos TOPIC_QOS_DEFAULT = new TopicQos() [static]
```

Special value for creating a **com.rti.dds.topic.Topic** (p. 1807) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706), this special value is used to indicate that the **com.rti.dds.topic.Topic** (p. 1807) should be created with the default **com.rti.dds.topic.Topic** (p. 1807) QoS by means of the operation get_default_topic_qos and using the resulting QoS to create the **com.rti.dds.topic.Topic** (p. 1807).

When used in **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.domain.↩ DomainParticipant.set_default_topic_qos** (p. 679) operation had never been called.

When used in **com.rti.dds.topic.Topic.set_qos** (p. 1809), this special value is used to indicate that the QoS of the **com.rti.dds.topic.Topic** (p. 1807) should be changed to match the current default QoS set in the **com.rti.dds.↩ domain.DomainParticipant** (p. 670) that the **com.rti.dds.topic.Topic** (p. 1807) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Topic; for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 679).

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

com.rti.dds.domain.DomainParticipant.set_default_topic_qos (p. 679)

com.rti.dds.topic.Topic.set_qos (p. 1809)

6.5.2.2 TOPIC_QOS_PRINT_ALL

```
final TopicQos TOPIC_QOS_PRINT_ALL = new TopicQos() [static]
```

Special value which can be supplied to **com.rti.dds.topic.TopicQos.toString(TopicQos baseQos, QosPrintFormat format)** (p. 1826) indicating that all of the QoS should be printed.

The **com.rti.dds.topic.TopicQos.toString(TopicQos baseQos, QosPrintFormat format)** (p. 1826) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_PRINT_ALL** (p. 45) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.↩ domain.DomainParticipant.TOPIC_QOS_PRINT_ALL** (p. 45) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.topic.TopicQos.toString(TopicQos base↩ Qos, QosPrintFormat format)** (p. 1826) API.

Referenced by **TopicQos.toString()**.

6.5.2.3 PUBLISHER_QOS_DEFAULT

```
final PublisherQos PUBLISHER_QOS_DEFAULT = new PublisherQos() [static]
```

Special value for creating a **com.rti.dds.publication.Publisher** (p. 1466) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693), this special value is used to indicate that the **com.rti.dds.publication.Publisher** (p. 1466) should be created with the default **com.rti.dds.publication.↩ Publisher** (p. 1466) QoS by means of the operation get_default_publisher_qos and using the resulting QoS to create the **com.rti.dds.publication.Publisher** (p. 1466).

When used in **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 682), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.domain.↩ DomainParticipant.set_default_publisher_qos** (p. 682) operation had never been called.

When used in **com.rti.dds.publication.Publisher.set_qos** (p. 1476), this special value is used to indicate that the QoS of the **com.rti.dds.publication.Publisher** (p. 1466) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipant** (p. 670) that the **com.rti.dds.publication.Publisher** (p. 1466) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Publisher; for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_publisher_qos** (p. 681).

See also

com.rti.dds.domain.DomainParticipant.create_publisher (p. 693)

com.rti.dds.domain.DomainParticipant.set_default_publisher_qos (p. 682)

com.rti.dds.publication.Publisher.set_qos (p. 1476)

6.5.2.4 PUBLISHER_QOS_PRINT_ALL

```
final PublisherQos PUBLISHER_QOS_PRINT_ALL = new PublisherQos() [static]
```

Special value which can be supplied to **com.rti.dds.publication.PublisherQos.toString(PublisherQos baseQos, QosPrintFormat format)** (p. 1491) indicating that all of the QoS should be printed.

The **com.rti.dds.publication.PublisherQos.toString(PublisherQos baseQos, QosPrintFormat format)** (p. 1491) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_PRINT_↩ ALL** (p. 46) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.↩ domain.DomainParticipant.PUBLISHER_QOS_PRINT_ALL** (p. 46) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.publication.PublisherQos.toString(↩ PublisherQos baseQos, QosPrintFormat format)** (p. 1491) API.

Referenced by **PublisherQos.toString()**.

6.5.2.5 SUBSCRIBER_QOS_PRINT_ALL

```
final SubscriberQos SUBSCRIBER_QOS_PRINT_ALL [static]
```

Initial value:

```
=
    new SubscriberQos()
```

Special value which can be supplied to **com.rti.dds.subscription.SubscriberQos.toString(SubscriberQos base↩ Qos, QosPrintFormat format)** (p. 1758) indicating that all of the QoS should be printed.

The **com.rti.dds.subscription.SubscriberQos.toString(SubscriberQos baseQos, QosPrintFormat format)** (p. 1758) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_↩ QOS_PRINT_ALL** (p. 47) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.↩ domain.DomainParticipant.SUBSCRIBER_QOS_PRINT_ALL** (p. 47) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.subscription.SubscriberQos.toString(↩ SubscriberQos baseQos, QosPrintFormat format)** (p. 1758) API.

Referenced by **SubscriberQos.toString()**.

6.5.2.6 SUBSCRIBER_QOS_DEFAULT

```
final SubscriberQos SUBSCRIBER_QOS_DEFAULT [static]
```

Initial value:

```
=
    new SubscriberQos()
```

Special value for creating a **com.rti.dds.subscription.Subscriber** (p. 1730) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697), this special value is used to indicate that the **com.rti.dds.subscription.Subscriber** (p. 1730) should be created with the default **com.rti.dds.subscription.Subscriber** (p. 1730) QoS by means of the operation get_default_subscriber_qos and using the resulting QoS to create the **com.rti.dds.subscription.Subscriber** (p. 1730).

When used in **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687) operation had never been called.

When used in **com.rti.dds.subscription.Subscriber.set_qos** (p. 1744), this special value is used to indicate that the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1730) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipant** (p. 670) that the **com.rti.dds.subscription.Subscriber** (p. 1730) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a Subscriber; for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 689).

See also

com.rti.dds.domain.DomainParticipant.create_subscriber (p. 697)

com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 689)

com.rti.dds.subscription.Subscriber.set_qos (p. 1744)

6.5.2.7 DOMAINPARTICIPANT_QOS_PRINT_ALL

```
final DomainParticipantQos DOMAINPARTICIPANT_QOS_PRINT_ALL [static]
```

Initial value:

```
=
    new DomainParticipantQos()
```

Special value which can be supplied to **com.rti.dds.domain.DomainParticipantQos.toString(DomainParticipantQos baseQos, QosPrintFormat format)** (p. 798) indicating that all of the QoS should be printed.

The **com.rti.dds.domain.DomainParticipantQos.toString(DomainParticipantQos baseQos, QosPrintFormat format)** (p. 798) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.domain.DomainParticipant.↩ DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 48) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.↩ domain.DomainParticipant.DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 48) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.domain.DomainParticipantQos.toString(↩ DomainParticipantQos baseQos, QosPrintFormat format)** (p. 798) API.

Referenced by **DomainParticipantQos.toString()**.

6.5.2.8 DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL

```
final DomainParticipantFactoryQos DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL [static]
```

Initial value:

```
=
    new DomainParticipantFactoryQos()
```

Special value which can be supplied to **com.rti.dds.domain.DomainParticipantFactoryQos.toString(Domain↩ ParticipantFactoryQos baseQos, QosPrintFormat format)** (p. 789) indicating that all of the QoS should be printed.

The **com.rti.dds.domain.DomainParticipantFactoryQos.toString(DomainParticipantFactoryQos baseQos, QosPrintFormat format)** (p. 789) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.domain.↩ DomainParticipant.DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 49) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.↩ dds.domain.DomainParticipant.DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 49) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.domain.DomainParticipantFactoryQos.↩ toString(DomainParticipantFactoryQos baseQos, QosPrintFormat format)** (p. 789) API.

Referenced by **DomainParticipantFactoryQos.toString()**.

6.5.2.9 FLOW_CONTROLLER_PROPERTY_DEFAULT

```
final FlowControllerProperty_t FLOW_CONTROLLER_PROPERTY_DEFAULT [static]
```

Initial value:

```
=
    new FlowControllerProperty_t()
```

<<***extension***>> *(*p. *155)* Special value for creating a **com.rti.dds.publication.FlowController** (p. 1055) with default property.

When used in **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 691) and **com.rti.dds.↩ publication.FlowController.set_property** (p. 1057), this special value represents the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678), or else, if the call was never made, the default values listed in **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059).

When used in **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678), this special value indicates that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.↩ domain.DomainParticipant.set_default_flowcontroller_property** (p. 678) operation had never been called.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default properties for a FlowController; for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_flowcontroller_↩ property** (p. 677).

See also

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 691)

com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property (p. 678)

com.rti.dds.publication.FlowController.set_property (p. 1057)

6.5.2.10 SQLFILTER_NAME

```
final String SQLFILTER_NAME [static]
```

<<***extension***>> *(*p. *155)* The name of the built-in SQL filter that can be used with ContentFilteredTopics and Multi↩ Channel DataWriters.

See also

Queries and Filters Syntax (p. 104)

6.5.2.11 STRINGMATCHFILTER_NAME

```
final String STRINGMATCHFILTER_NAME  [static]
```

<<***extension***>> *(p. 155)* The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.

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

See also

Queries and Filters Syntax (p. 104)

6.6 Built-in Topics

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

Modules

- **Participant Built-in Topics**

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

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

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

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

  Builtin topic for accessing requests from different services within RTI Connext.
- **Common types and functions**

  Types and functions related to the built-in topics.

6.6.1 Detailed Description

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

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

A set of built-in topics and corresponding **com.rti.dds.subscription.DataReader** (p. 450) objects are introduced to be used by the application to access this discovery information.

The information can be accessed as if it were normal application data. This allows the application to know when there are any changes in those values by means of the **com.rti.dds.infrastructure.Listener** (p. 1236) or the **com.rti.dds.infrastructure.Condition** (p. 429) mechanisms.

The built-in data-readers all belong to a built-in **com.rti.dds.subscription.Subscriber** (p. 1730), which can be retrieved by using the method **com.rti.dds.domain.DomainParticipant.get_builtin_subscriber** (p. 720). The built-in **com.rti.dds.subscription.DataReader** (p. 450) objects can be retrieved by using the operation **com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1740), with the topic name as a parameter.

Built-in entities have default listener settings as well. The built-in **com.rti.dds.subscription.Subscriber** (p. 1730) and all of its built-in topics have 'nil' listeners with all statuses appearing in their listener masks (acting as a NO-OP listener that does not reset communication status). The built-in DataReaders have null listeners with no statuses in their masks.

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

The built-in **com.rti.dds.subscription.DataReader** (p. 450) will not provide data pertaining to entities created from the same **com.rti.dds.domain.DomainParticipant** (p. 670) under the assumption that such entities are already known to the application that created them.

The discovery information provided by each built-in reader won't be received until it is first looked up. And if the DomainParticipant was created in a disabled state (see **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 1035)), the built-in subscriber and readers will need to be enabled explicitly (by the **com.rti.dds.infrastructure.Entity.enable** (p. 1032) operation).

Refer to **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349), builtin.TopicBuiltinTopicData, **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) and **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) for a description of all the built-in topics and their contents.

The QoS of the built-in **com.rti.dds.subscription.Subscriber** (p. 1730) and **com.rti.dds.subscription.DataReader** (p. 450) objects is given by the following table:

Table 6.2 ***QoS of built-in com.rti.dds.subscription.Subscriber (*** p. ***1730) and com.rti.dds.subscription.DataReader (*** p. ***450)***

QoS	Value
com.rti.dds.infrastructure.UserDataQosPolicy (p. 1959)	0-length sequence
com.rti.dds.infrastructure.TopicDataQosPolicy (p. 1819)	0-length sequence
com.rti.dds.infrastructure.GroupDataQosPolicy (p. 1127)	0-length sequence
com.rti.dds.infrastructure.DurabilityQosPolicy (p. 830)	com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_LOCAL_DURABILITY_QOS
com.rti.dds.infrastructure.DurabilityServiceQosPolicy (p. 837)	Does not apply as **com.rti.dds.infrastructure.DurabilityQosPolicyKind** (p. 835) is com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_LOCAL_DURABILITY_QOS

QoS	Value
com.rti.dds.infrastructure.PresentationQosPolicy (p. 1379)	access_scope = com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.↩ PresentationQosPolicyAccessScopeKind.TOPIC_↩ PRESENTATION_QOS coherent_access = com.rti.dds.infrastructure.false ordered_access = com.rti.dds.infrastructure.false
com.rti.dds.infrastructure.DeadlineQosPolicy (p. 632)	Period = infinite
com.rti.dds.infrastructure.LatencyBudgetQosPolicy (p. 1231)	duration = 0
com.rti.dds.infrastructure.OwnershipQosPolicy (p. 1342)	**com.rti.dds.infrastructure.OwnershipQosPolicy↩ Kind.SHARED_OWNERSHIP_QOS** (p. 1347)
com.rti.dds.infrastructure.OwnershipStrengthQos↩ Policy (p. 1348)	value = 0
com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1243)	kind = com.rti.dds.infrastructure.LivelinessQos↩ PolicyKind.LivelinessQosPolicyKind.AUTOMATIC_↩ LIVELINESS_QOS lease_duration = 0
com.rti.dds.infrastructure.TimeBasedFilterQos↩ Policy (p. 1804)	minimum_separation = 0
com.rti.dds.infrastructure.PartitionQosPolicy (p. 1365)	0-length sequence
com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)	kind = **com.rti.dds.infrastructure.ReliabilityQos↩ PolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) max_blocking_time = 100 milliseconds
com.rti.dds.infrastructure.DestinationOrderQos↩ Policy (p. 635)	com.rti.dds.infrastructure.DestinationOrderQos↩ PolicyKind.DestinationOrderQosPolicyKind.BY_↩ RECEPTION_TIMESTAMP_DESTINATIONORDER_↩ QOS
com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)	kind = com.rti.dds.infrastructure.HistoryQosPolicy↩ Kind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_↩ QOS depth = 1
com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)	max_samples = **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) max_instances = **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) max_samples_per_instance = **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH↩ _UNLIMITED** (p. 259)
com.rti.dds.infrastructure.ReaderDataLifecycle↩ QosPolicy (p. 1520)	autopurge_nowriter_samples_delay = infinite autopurge_disposed_samples_delay = infinite
com.rti.dds.infrastructure.EntityFactoryQosPolicy (p. 1035)	autoenable_created_entities = com.rti.dds.↩ infrastructure.true

6.7 Topic Module

Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.

Modules

- **Topics**

  com.rti.dds.topic.Topic *(p. 1807) entity and associated elements*
- **FlatData Topic-Types**

  *<<**extension**>> (p. 155) FlatData Language Binding for IDL topic-types*
- **Zero Copy Transfer Over Shared Memory**

  *<<**extension**>> (p. 155) Zero Copy transfer over shared memory*
- **User Data Type Support**

  Defines generic classes and macros to support user data types.
- **Type Code Support**

  *<<**extension**>> (p. 155) A **com.rti.dds.typecode.TypeCode** (p. 1873) is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 65) capability or to inspect the type information you receive from remote readers and writers.*
- **Built-in Types**

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

  *Types used as part of **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349), **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452), **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) to describe Trust Plugins configuration.*
- **Dynamic Data**

  *<<**extension**>> (p. 155) The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.*

6.7.1 Detailed Description

Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.

6.8 Topics

com.rti.dds.topic.Topic (p. 1807) entity and associated elements

Classes

- interface **ContentFilter**

  *<<**interface**>> (p. 156) Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436)*
- interface **ContentFilteredTopic**

  *<<**interface**>> (p. 156) Specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows for content-based subscriptions.*
- class **ExpressionProperty**

  *Provides additional information about the filter expression passed to **com.rti.dds.topic.WriterContentFilter.writer_compile** (p. 2003) .*
- class **FilterSampleInfo**

  Provides meta information associated with the sample.
- class **InconsistentTopicStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS
- interface **MultiTopic**

  ***[Not supported (optional)]** <<**interface**>> (p. 156) A specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows subscriptions that combine/filter/rearrange data coming from several topics.*
- class **PrintFormatKind**

  Format kinds available when converting data samples to string representations.
- class **PrintFormatProperty**

  A collection of attributes used to configure how data samples will be formatted when converted to a string.
- interface **Topic**

  *<<**interface**>> (p. 156) The most basic description of the data to be published and subscribed.*
- class **TopicAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- interface **TopicDescription**

  ***com.rti.dds.topic.Topic** (p. 1807) entity and associated elements*
- interface **TopicListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.topic.Topic** (p. 1807) entities.*
- class **TopicQos**

  *QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1807) entity.*
- interface **WriterContentFilter**

  *<<**interface**>> (p. 156) Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).*

6.8.1 Detailed Description

com.rti.dds.topic.Topic (p. 1807) entity and associated elements

6.9 FlatData Topic-Types

*<<**extension**>> (p. 155)* FlatData Language Binding for IDL topic-types

*<<**extension**>> (p. 155)* FlatData Language Binding for IDL topic-types

Warning

The FlatData language binding is available in the Traditional C++ API and in the Modern C++ API. It is not available in this API.

FlatData is a language binding for IDL types in which the in-memory representation of a sample matches the wire representation. Therefore, the cost of serialization/deserialization is zero.

Note

For a complete description of the FlatData language binding and its benefits, and a tutorial, see the "Sending Large Data" chapter in the **RTI Connext User's Manual**.

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

6.10 Zero Copy Transfer Over Shared Memory

*<<**extension**>> (p. 155)* Zero Copy transfer over shared memory

*<<**extension**>> (p. 155)* Zero Copy transfer over shared memory

Note

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

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

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

6.11 User Data Type Support

Defines generic classes and macros to support user data types.

Classes

- class **Foo**

  A representative user-defined data type.
- class **FooTypeSupport**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type specific interface.*
- class **InstanceHandle_t**

  Type definition for an instance handle.
- class **InstanceHandleSeq**

  *Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.↩ InstanceHandle_t** (p. 1152) > .*
- interface **TypeSupport**

  *<<**interface**>> (p. 156) An abstract marker interface that has to be specialized for each concrete user data type that will be used by the application.*

6.11.1 Detailed Description

Defines generic classes and macros to support user data types.

DDS specifies strongly typed interfaces to read and write user data. For each data class defined by the application, there is a number of specialised classes that are required to facilitate the type-safe interaction of the application with RTI Connext.

RTI Connext provides an automatic means to generate all these type-specific classes with the rtiddsgen utility. The complete set of automatic classes created for a hypothetical user data type named `Foo` are shown below.

an application data type named `Foo`"

The macros defined here declare the strongly typed APIs needed to support an arbitrary user defined data of type `Foo`.

See also

the `Code Generator User's Manual`

6.12 Type Code Support

<<***extension***>> *(*p. *155)* A ***com.rti.dds.typecode.TypeCode*** *(*p. *1873)* is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 65) capability or to inspect the type information you receive from remote readers and writers.

Classes

- class **EnumMember**

  A description of a member of an enumeration.
- class **ExtensibilityKind**

  Type to indicate the extensibility of a type.
- class **PRIVATE_MEMBER**

  Constant used to indicate that a value type member is private.
- class **PUBLIC_MEMBER**

  Constant used to indicate that a value type member is public.
- class **StructMember**

  A description of a member of a struct.
- class **TCKind**

  Enumeration type for ***com.rti.dds.typecode.TypeCode*** *(*p. *1873) kinds.*
- class **TypeCode**

  The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.
- class **TypeCodeFactory**

  A singleton factory for creating, copying, and deleting data type definitions dynamically.
- class **UnionMember**

A description of a member of a union.

- class **ValueMember**

  A description of a member of a value type.
- class **VM_ABSTRACT**

  Constant used to indicate that a value type has the `abstract` *modifier.*
- class **VM_CUSTOM**

  Constant used to indicate that a value type has the `custom` *modifier.*
- class **VM_NONE**

  Constant used to indicate that a value type has no modifiers.
- class **VM_TRUNCATABLE**

  Constant used to indicate that a value type has the `truncatable` *modifier.*

Variables

- static final **ExtensibilityKind FINAL_EXTENSIBILITY**

  Specifies that a type has FINAL extensibility.
- static final **ExtensibilityKind EXTENSIBLE_EXTENSIBILITY**

  Specifies that a type has EXTENSIBLE extensibility.
- static final **ExtensibilityKind MUTABLE_EXTENSIBILITY**

  Specifies that a type has MUTABLE extensibility.

6.12.1 Detailed Description

<<***extension***>> *(*p. *155)* A ***com.rti.dds.typecode.TypeCode*** *(*p. *1873)* is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. You will need to understand this API in order to use the **Dynamic Data** (p. 65) capability or to inspect the type information you receive from remote readers and writers.

Type codes are values that are used to describe arbitrarily complex types at runtime. Type code values are manipulated via the **com.rti.dds.typecode.TypeCode** (p. 1873) class, which has an analogue in CORBA.

A **com.rti.dds.typecode.TypeCode** (p. 1873) value consists of a type code *kind* (represented by the **com.rti.dds.typecode.TCKind** (p. 1786) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **com.rti.dds.typecode.TypeCode** (p. 1873), and in the case of complex types (structures, arrays, and so on), these contained type codes contain their own members.

There are a number of uses for type codes. The type code mechanism can be used to unambiguously match type representations. The com.rti.dds.typecode.TypeCode.TypeCode.equals method is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

6.12.2 Accessing a Local com.rti.dds.typecode.TypeCode

When generating types with rtiddsgen, type codes are always enabled. For these types, a **com.rti.dds.typecode.TypeCode** (p. 1873) may be accessed via the **FooTypeCode.VALUE** member. This API also includes support for dynamic creation of **com.rti.dds.typecode.TypeCode** (p. 1873) values, typically for use with the **Dynamic Data** (p. 65) API. You can create a **com.rti.dds.typecode.TypeCode** (p. 1873) using the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) class. You will construct the **com.rti.dds.typecode.TypeCode** (p. 1873) recursively, from the outside in: start with the type codes for primitive types, then compose them into complex types like arrays, structures, and so on. You will find the following methods helpful:

- **com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc** (p. 1935), which provides the **com.rti.dds.typecode.TypeCode** (p. 1873) instances corresponding to the primitive types (e.g. **com.rti.dds.typecode.TCKind.TK_LONG** (p. 1788), **com.rti.dds.typecode.TCKind.TK_SHORT** (p. 1787), and so on).
- **com.rti.dds.typecode.TypeCodeFactory.create_string_tc** (p. 1931) and **com.rti.dds.typecode.TypeCodeFactory.create_wstring_tc** (p. 1932) create a **com.rti.dds.typecode.TypeCode** (p. 1873) representing a text string with a certain *bound* (i.e. maximum length).
- **com.rti.dds.typecode.TypeCodeFactory.create_array_tc** (p. 1933) and **com.rti.dds.typecode.TypeCodeFactory.create_sequence_tc** (p. 1932) create a **com.rti.dds.typecode.TypeCode** (p. 1873) for a collection based on the **com.rti.dds.typecode.TypeCode** (p. 1873) for its elements.
- **com.rti.dds.typecode.TypeCodeFactory.create_struct_tc** (p. 1923) and **com.rti.dds.typecode.TypeCodeFactory.create_value_tc** (p. 1925) create a **com.rti.dds.typecode.TypeCode** (p. 1873) for a structured type.

6.12.3 Accessing a Remote com.rti.dds.typecode.TypeCode

In addition to being used locally, RTI Connext can transmit **com.rti.dds.typecode.TypeCode** (p. 1873) on the network between participants. This information can be used to access information about types used remotely at runtime, for example to be able to publish or subscribe to topics of arbitrarily types (see **Dynamic Data** (p. 65)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote **com.rti.dds.typecode.TypeCode** (p. 1873) information is shared during discovery over the publication and subscription built-in topics and can be accessed using the built-in readers for these topics; see **Built-in Topics** (p. 51). Discovered **com.rti.dds.typecode.TypeCode** (p. 1873) values are not cached by RTI Connext upon receipt and are therefore not available from the built-in topic data returned by **com.rti.dds.publication.DataWriter.get_matched_subscription_data** (p. 568) or **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466).

The space available locally to deserialize a discovered remote **com.rti.dds.typecode.TypeCode** (p. 1873) is specified by the **com.rti.dds.domain.DomainParticipant** (p. 670)'s **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_code_max_serialized_length** (p. 817) QoS parameter. To support especially complex type codes, it may be necessary for you to increase the value of this parameter.

See also

com.rti.dds.typecode.TypeCode (p. 1873)

Dynamic Data (p. 65)

the `Code Generator User's Manual`

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

6.12.4 Variable Documentation

6.12.4.1 FINAL_EXTENSIBILITY

final **ExtensibilityKind** FINAL_EXTENSIBILITY [static]

Specifies that a type has FINAL extensibility.

A type may be final, indicating that the range of its possible values is strictly defined. In particular, it is not possible to add elements to members of collection or aggregated types while maintaining type assignability.

The following types are always final:

- **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790)
- All primitive types

6.12.4.2 EXTENSIBLE_EXTENSIBILITY

final **ExtensibilityKind** EXTENSIBLE_EXTENSIBILITY [static]

Specifies that a type has EXTENSIBLE extensibility.

A type may be appendable (formerly called extensible), indicating that two types, where one contains all of the elements/members of the other plus additional elements/members appended to the end, may remain assignable.

Referenced by **TypeCodeFactory.create_enum_tc()**, **TypeCodeFactory.create_struct_tc()**, **TypeCodeFactory.create_union_tc()**, and **TypeCodeFactory.create_value_tc()**.

6.12.4.3 MUTABLE_EXTENSIBILITY

final **ExtensibilityKind** MUTABLE_EXTENSIBILITY [static]

Specifies that a type has MUTABLE extensibility.

A type may be mutable, indicating that two types may differ from one another in the additional, removal, and/or transposition of elements/members while remaining assignable.

The following types are always mutable:

- **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790)
- **com.rti.dds.typecode.TCKind.TK_STRING** (p. 1790)
- **com.rti.dds.typecode.TCKind.TK_WSTRING** (p. 1792)

6.13 Built-in Types

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

Modules

- **String Built-in Type**

  Built-in type consisting of a single character string.
- **KeyedString Built-in Type**

  Built-in type consisting of a string payload and a second string that is the key.
- **Octets Built-in Type**

  Built-in type consisting of a variable-length array of opaque bytes.
- **KeyedOctets Built-in Type**

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

6.13.1 Detailed Description

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

The middleware provides four built-in types:

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

The String and com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString types are appropriate for simple text-based applications. The com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes and com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes types are appropriate for applications that perform their own custom data serialization, such as legacy applications still in the process of migrating to RTI Connext. In most cases, string-based or structured data is preferable to opaque data, because the latter cannot be easily visualized in tools or used with content-based filters (see **com.rti.dds.topic.ContentFilteredTopic** (p. 436)).

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

- At compile time, by generating code from an IDL or XML file using the rtiddsgen utility
- At runtime, by using the **Dynamic Data** (p. 65) API

6.13.2 Managing Memory for Builtin Types

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

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

You can configure the maximum size of the builtin types on a per-DataWriter and per-DataReader basis using the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) in DataWriters, DataReaders or Participants.

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

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

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the **com.rti.dds.↩ type.builtin.StringDataWriter** (p. 1717) or received the **com.rti.↩ dds.type.builtin.StringDataReader** (p. 1714) (includes the NULL-terminated character). Default: dds.builtin_type.string.max_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_key_size	Maximum size of the keys used by the com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.KeyedStringDataWriter or com.rti.↩ dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_string.alloc_size	Maximum size of the strings published by the com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.KeyedStringDataWriter or received by the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString↩ DataReader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_size if defined. Otherwise, 1024.
dds.builtin_type.octets.alloc_size	Maximum size of the octet sequences published the com.rti.↩ dds.type.builtin.com.rti.dds.type.builtin.BytesDataWriter or received by the com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesData↩ Reader. Default: dds.builtin_type.octets.max_size if defined. Otherwise, 2048.
dds.builtin_type.keyed_octets.alloc_key_size	Maximum size of the key published by the com.rti.dds.type.builtin.↩ com.rti.dds.type.builtin.KeyedBytesDataWriter or received by the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesData↩ Reader (includes the NULL-terminated character). Default: dds.builtin_type.keyed_octets.max_key_size if defined. Otherwise, 1024.
dds.builtin_type.keyed_octets.alloc_size	Maximum size of the octets sequences published by a com.rti.↩ dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter or received by a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ BytesDataReader. Default: dds.builtin_type.keyed_octets.max_size if defined. Otherwise, 2048.

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

6.13.3 Typecodes for Builtin Types

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

```
module DDS {

    struct String {

        string value;

    };


    struct KeyedString {

        string key;

        string value;

    };


    struct Octets {

        sequence<octet> value;

    };


    struct KeyedOctets {

        string key;

        sequence<octet> value;

    };

};
```

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

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

Property	Description
dds.builtin_type.string.max_size	Maximum size of the strings published by the StringDataWriters and received by the StringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_key_size	Maximum size of the keys used by the KeyedStringDataWriters and KeyedStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_size	Maximum size of the strings published by the KeyedStringData↩ Writers and received by the KeyedStringDataReaders belonging to a DomainParticipant using the builtin type (includes the NULL-terminated character). Default: 1024

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

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

6.14 Built-in Topic's Trust Types

Types used as part of **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349), **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452), **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) to describe Trust Plugins configuration.

Classes

- class **EndpointTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered endpoint.
- class **EndpointTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered endpoint.
- class **EndpointTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered endpoint.
- class **ParticipantTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustKeyEstablishmentAlgorithmInfo**

  Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered DomainParticipant.
- class **ParticipantTrustSignatureAlgorithmInfo**

  Trust Plugins signature algorithm information associated with the discovered DomainParticipant.
- class **TrustAlgorithmRequirements**

  Type to describe Trust Plugins algorithm requirements for an entity.

6.14.1 Detailed Description

Types used as part of **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349), **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452), **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) to describe Trust Plugins configuration.

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

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

6.15 Dynamic Data

<<***extension***>> *(p. 155)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

Classes

- class **DynamicData**

  A sample of any complex data type, which can be inspected and manipulated reflectively.
- class **DynamicDataInfo**

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*
- class **DynamicDataMemberInfo**

  A descriptor for a single member (i.e. field) of dynamically defined data type.
- class **DynamicDataProperty_t**

  *A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataReader**

  *Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).*
- class **DynamicDataSeq**

  *An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 847) elements.*
- class **DynamicDataTypeProperty_t**

  *A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataTypeSerializationProperty_t**

  Properties that govern how data of a certain type will be serialized on the network.
- class **DynamicDataTypeSupport**

  *A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataWriter**

  *Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).*

Functions

- **DynamicDataInfo** ()

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*
- **DynamicDataInfo** (int **member_count**, int **stored_size**, boolean is_optimized_storage)

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*
- **DynamicDataMemberInfo** ()

  A descriptor for a single member (i.e. field) of dynamically defined data type.
- **DynamicDataMemberInfo** (int **member_id**, String **member_name**, boolean **member_exists**, **TCKind member_kind**, int representation_count, int **element_count**, **TCKind element_kind**)

  A descriptor for a single member (i.e. field) of dynamically defined data type.

Variables

- static final **DynamicDataProperty_t PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 955).*
- static final **DynamicDataTypeProperty_t TYPE_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 990).*

6.15.1 Detailed Description

<<***extension***>> *(p. 155)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

This API allows you to define new data types, modify existing data types, and interact reflectively with samples. To use it, you will take the following steps:

1. Obtain a com.rti.dds.typecode.TypeCode (p. 1873) (see Type Code Support (p. 57)) that defines the type definition you want to use.
A **com.rti.dds.typecode.TypeCode** (p. 1873) includes a type's *kind* (**com.rti.dds.typecode.TCKind** (p. 1786)), *name*, and *members* (that is, fields). You can create your own **com.rti.dds.typecode.TypeCode** (p. 1873) using the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) class – see, for example, the **com.rti.dds.typecode.TypeCodeFactory.create_struct_tc** (p. 1923) method. Alternatively, you can use a remote **com.rti.dds.typecode.TypeCode** (p. 1873) that you discovered on the network (see **Built-in Topics** (p. 51)) or one generated by rtiddsgen (see the `Code Generator User's Manual`).

2. Wrap the com.rti.dds.typecode.TypeCode (p. 1873) in a com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 995) object.
See the constructor com.rti.dds.dynamicdata.DynamicDataTypeSupport.DynamicDataTypeSupport.DynamicDataTypeSupport. This object lets you connect the type definition to a **com.rti.dds.domain.DomainParticipant** (p. 670) and manage data samples (of type **com.rti.dds.dynamicdata.DynamicData** (p. 847)).

3. Register the com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 995) with one or more domain participants.
See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 997). This action associates the data type with a logical name that you can use to create topics. (Starting with this step, working with a dynamically defined data type is almost exactly the same as working with a generated one.)

4. Create a com.rti.dds.topic.Topic (p. 1807) from the com.rti.dds.domain.DomainParticipant (p. 670).
Use the name under which you registered your data type – see **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706). This **com.rti.dds.topic.Topic** (p. 1807) is what you will use to produce and consume data.

5. Create a com.rti.dds.dynamicdata.DynamicDataWriter (p. 1003) and/or com.rti.dds.dynamicdata.DynamicDataReader (p. 959).
These objects will produce and/or consume data (of type **com.rti.dds.dynamicdata.DynamicData** (p. 847)) on the **com.rti.dds.topic.Topic** (p. 1807). You can create these objects directly from the **com.rti.dds.domain.DomainParticipant** (p. 670) – see **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 700) and **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 703) – or by first creating intermediate **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730) objects – see **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693) and **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697).

6. Write and/or read the data of interest.

7. Tear down the objects described above.
You should delete them in the reverse order in which you created them. Note that unregistering your data type with the **com.rti.dds.domain.DomainParticipant** (p. 670) is optional; all types are automatically unregistered when the **com.rti.dds.domain.DomainParticipant** (p. 670) itself is deleted.

6.15.2 Function Documentation

6.15.2.1 DynamicDataInfo() [1/2]

```
DynamicDataInfo ( )
```

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.get_info (p. 865)

6.15.2.2 DynamicDataInfo() [2/2]

```
DynamicDataInfo (
            int member_count,
            int stored_size,
            boolean is_optimized_storage )
```

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.get_info (p. 865)

References **DynamicDataInfo.member_count**, and **DynamicDataInfo.stored_size**.

6.15.2.3 DynamicDataMemberInfo() [1/2]

```
DynamicDataMemberInfo ( )
```

A descriptor for a single member (i.e. field) of dynamically defined data type.

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 871)

References **TCKind.TK_NULL**.

6.15.2.4 DynamicDataMemberInfo() [2/2]

```
DynamicDataMemberInfo (
            int member_id,
            String member_name,
            boolean member_exists,
             TCKind member_kind,
            int representation_count,
            int element_count,
             TCKind element_kind )
```

A descriptor for a single member (i.e. field) of dynamically defined data type.

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 871)

References **DynamicDataMemberInfo.element_count**, **DynamicDataMemberInfo.element_kind**, **DynamicData↩ MemberInfo.member_exists**, **DynamicDataMemberInfo.member_id**, **DynamicDataMemberInfo.member_kind**, and **DynamicDataMemberInfo.member_name**.

6.15.3 Variable Documentation

6.15.3.1 PROPERTY_DEFAULT

```
final DynamicDataProperty_t PROPERTY_DEFAULT [static]
```

Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 955).

Pass this object instead of your own **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 955) object to use the default property values:

```
DynamicData sample = new DynamicData(

    myTypeCode,

    DynamicData.DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also

com.rti.dds.dynamicdata.DynamicDataProperty_t (p. 955)

Referenced by **BytesTypeSupport.data_to_string()**, **KeyedBytesTypeSupport.data_to_string()**, **KeyedString↩ TypeSupport.data_to_string()**, and **StringTypeSupport.data_to_string()**.

6.15.3.2 TYPE_PROPERTY_DEFAULT

```
final DynamicDataTypeProperty_t TYPE_PROPERTY_DEFAULT [static]
```

Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 990).

Pass this object instead of your own **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 990) object to use the default property values:

```
DynamicDataTypeSupport support = new DynamicDataTypeSupport(

    myTypeCode,

    DynamicDataTypeSupport.DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT);
```

See also

com.rti.dds.dynamicdata.DynamicDataTypeProperty_t (p. 990)

6.16 Publication Module

Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**

  com.rti.dds.publication.Publisher *(p. 1466) entity and associated elements*
- **Data Writers**

  com.rti.dds.publication.DataWriter *(p. 553) entity and associated elements*
- **Flow Controllers**

  *<<**extension**>> (p. 155)* ***com.rti.dds.publication.FlowController*** *(p. 1055) and associated elements*
- **Multi-channel DataWriters**

  APIs related to Multi-channel DataWriters.

6.16.1 Detailed Description

Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.

"DCPS Publication package"

6.17 Publishers

com.rti.dds.publication.Publisher (p. 1466) entity and associated elements

Classes

- interface **Publisher**

  *<<**interface**>> (p. 156) A publisher is the object responsible for the actual dissemination of publications.*
- class **PublisherAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- interface **PublisherListener**

  *<<**interface**>> (p. 156)* ***com.rti.dds.infrastructure.Listener*** *(p. 1236) for* ***com.rti.dds.publication.Publisher*** *(p. 1466) status.*
- class **PublisherQos**

  QoS policies supported by a ***com.rti.dds.publication.Publisher*** *(p. 1466) entity.*
- class **PublisherSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.publication.Publisher*** *(p. 1466) > .*

Variables

- static final **DataWriterQos** **DATAWRITER_QOS_DEFAULT**

  Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 553) with default QoS.*
- static final **DataWriterQos** **DATAWRITER_QOS_USE_TOPIC_QOS** = new **DataWriterQos**()

  Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 553) with a combination of the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 612) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1824).*
- static final **DataWriterQos** **DATAWRITER_QOS_PRINT_ALL** = new **DataWriterQos**()

  Special value which can be supplied to ***com.rti.dds.publication.DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)*** *(p. 615) indicating that all of the QoS should be printed.*

6.17.1 Detailed Description

com.rti.dds.publication.Publisher (p. 1466) entity and associated elements

6.17.2 Variable Documentation

6.17.2.1 DATAWRITER_QOS_DEFAULT

```
final  DataWriterQos DATAWRITER_QOS_DEFAULT  [static]
```

Initial value:

```
=
    new DataWriterQos()
```

Special value for creating **com.rti.dds.publication.DataWriter** (p. 553) with default QoS.

When used in **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471), this special value is used to indicate that the **com.rti.dds.publication.DataWriter** (p. 553) should be created with the default **com.rti.dds.publication.DataWriter** (p. 553) QoS by means of the operation get_default_datawriter_qos and using the resulting QoS to create the **com.rti.dds.publication.DataWriter** (p. 553).

When used in **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469) operation had never been called.

When used in **com.rti.dds.publication.DataWriter.set_qos** (p. 556), this special value is used to indicate that the QoS of the **com.rti.dds.publication.DataWriter** (p. 553) should be changed to match the current defualt QoS set in the **com.rti.dds.publication.Publisher** (p. 1466) that the **com.rti.dds.publication.DataWriter** (p. 553) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a **DataWriter** (p. 553); for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 684).

See also

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.publication.Publisher.set_default_datawriter_qos (p. 1469)

com.rti.dds.publication.DataWriter.set_qos (p. 556)

6.17.2.2 DATAWRITER_QOS_USE_TOPIC_QOS

```
final DataWriterQos DATAWRITER_QOS_USE_TOPIC_QOS = new DataWriterQos() [static]
```

Special value for creating **com.rti.dds.publication.DataWriter** (p. 553) with a combination of the default **com.rti.dds.publication.DataWriterQos** (p. 612) and the **com.rti.dds.topic.TopicQos** (p. 1824).

The use of this value is equivalent to the application obtaining the default **com.rti.dds.publication.DataWriterQos** (p. 612) and the **com.rti.dds.topic.TopicQos** (p. 1824) (by means of the operation **com.rti.dds.topic.Topic.get_qos** (p. 1810)) and then combining these two QoS using the operation **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1484) whereby any policy that is set on the **com.rti.dds.topic.TopicQos** (p. 1824) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **com.rti.dds.publication.DataWriter** (p. 553).

This value should only be used in **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471).

See also

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1469)

com.rti.dds.topic.Topic.get_qos (p. 1810)

com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1484)

6.17.2.3 DATAWRITER_QOS_PRINT_ALL

```
final DataWriterQos DATAWRITER_QOS_PRINT_ALL = new DataWriterQos() [static]
```

Special value which can be supplied to **com.rti.dds.publication.DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)** (p. 615) indicating that all of the QoS should be printed.

The **com.rti.dds.publication.DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)** (p. 615) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.publication.Publisher.DATAWRITER_QOS_PRINT_ALL** (p. 72) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.publication.Publisher.DATAWRITER_QOS_PRINT_ALL** (p. 72) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.publication.DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)** (p. 615) API.

Referenced by **DataWriterQos.toString()**.

6.18 Data Writers

com.rti.dds.publication.DataWriter (p. 553) entity and associated elements

Classes

- class **FooDataWriter**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type specific data writer.*
- class **AcknowledgmentInfo**

  Information about an application-level acknowledged sample.
- interface **DataWriter**

  *<<**interface**>> (p. 156) Allows an application to set the value of the data to be published under a given **com.rti.dds.topic.Topic** (p. 1807).*
- class **DataWriterAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods or functions.)*
- class **DataWriterCacheStatus**

  *<<**extension**>> (p. 155) The status of the **DataWriter** (p. 553)'s cache. Provides information on cache related metrics such as the number of samples and instances in the **DataWriter** (p. 553) queue.*
- interface **DataWriterListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for writer status.*
- class **DataWriterProtocolStatus**

  *<<**extension**>> (p. 155) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.*
- class **DataWriterQos**

  *QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 553) entity.*
- class **LivelinessLostStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS
- class **OfferedDeadlineMissedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS
- class **OfferedIncompatibleQosStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS
- class **PublicationMatchedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS
- class **ReliableReaderActivityChangedStatus**

  *<<**extension**>> (p. 155) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*
- class **ReliableWriterCacheChangedStatus**

  *<<**extension**>> (p. 155) A summary of the state of a data writer's cache of unacknowledged samples written.*
- class **ReliableWriterCacheEventCount**

  *<<**extension**>> (p. 155) The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.*
- class **ServiceRequestAcceptedStatus**

  ***com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1709)*

6.18.1 Detailed Description

com.rti.dds.publication.DataWriter (p. 553) entity and associated elements

6.19 Flow Controllers

<<***extension***>> *(p. 155)* **com.rti.dds.publication.FlowController** (p. 1055) and associated elements

Classes

- interface **FlowController**

  <<***interface***>> *(p. 156) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***com.rti.dds.publication.DataWriter*** *(p. 553) instances are allowed to write data.*
- class **FlowControllerProperty_t**

  Determines the flow control characteristics of the ***com.rti.dds.publication.FlowController*** *(p. 1055).*
- class **FlowControllerSchedulingPolicy**

  Kinds of flow controller scheduling policy.
- class **FlowControllerTokenBucketProperty_t**

  com.rti.dds.publication.FlowController *(p. 1055) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.*

Variables

- static final String **DEFAULT_FLOW_CONTROLLER_NAME**

  [default] *Special value of* ***com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** *(p. 1497) that refers to the built-in default flow controller.*
- static final String **FIXED_RATE_FLOW_CONTROLLER_NAME**

  Special value of ***com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** *(p. 1497) that refers to the built-in fixed-rate flow controller.*
- static final String **ON_DEMAND_FLOW_CONTROLLER_NAME**

  Special value of ***com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** *(p. 1497) that refers to the built-in on-demand flow controller.*

6.19.1 Detailed Description

<<***extension***>> *(p. 155)* **com.rti.dds.publication.FlowController** (p. 1055) and associated elements

com.rti.dds.publication.FlowController (p. 1055) provides the network traffic shaping capability to asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances. For use cases and advantages of publishing asnychronously, please refer to **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496) of **com.rti.dds.publication.Data↩ WriterQos** (p. 612).

See also

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

com.rti.dds.publication.DataWriterQos.publish_mode (p. 620)

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 339)

6.19.2 Variable Documentation

6.19.2.1 DEFAULT_FLOW_CONTROLLER_NAME

```
final String DEFAULT_FLOW_CONTROLLER_NAME  [static]
```

[default] Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1497) that refers to the built-in default flow controller.

RTI Connext provides several built-in **com.rti.dds.publication.FlowController** (p. 1055) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 553). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

By default, flow control is disabled. That is, the built-in **com.rti.dds.publication.FlowController.DEFAULT_FLOW_↩ CONTROLLER_NAME** (p. 75) flow controller does not apply any flow control. Instead, it allows data to be sent asynchronously as soon as it is written by the **com.rti.dds.publication.DataWriter** (p. 553).

Essentially, this is equivalent to a user-created **com.rti.dds.publication.FlowController** (p. 1055) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059) = com.rti.dds.publication.↩ FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `max_tokens` = **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_added_per_period` = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_leaked_per_↩ period` = 0
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `period` = 60 seconds
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `bytes_per_token` = **com.↩ rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

See also

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 721)

com.rti.dds.publication.FlowController.set_property (p. 1057)

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 339)

6.19.2.2 FIXED_RATE_FLOW_CONTROLLER_NAME

```
final String FIXED_RATE_FLOW_CONTROLLER_NAME  [static]
```

Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1497) that refers to the built-in fixed-rate flow controller.

RTI Connext provides several builtin **com.rti.dds.publication.FlowController** (p. 1055) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 553). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **com.rti.dds.publication.FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 75) flow controller shapes the network traffic by allowing data to be sent only once every second. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

Essentially, this is equivalent to a user-created **com.rti.dds.publication.FlowController** (p. 1055) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059) = com.rti.dds.publication.↩ FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `max_tokens` = **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_added_per_period` = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_leaked_per_↩ period` = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `period` = 1 second
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `bytes_per_token` = **com.↩ rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

See also

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 721)

com.rti.dds.publication.FlowController.set_property (p. 1057)

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 339)

6.19.2.3 ON_DEMAND_FLOW_CONTROLLER_NAME

```
final String ON_DEMAND_FLOW_CONTROLLER_NAME  [static]
```

Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1497) that refers to the built-in on-demand flow controller.

RTI Connext provides several builtin **com.rti.dds.publication.FlowController** (p. 1055) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 553). The user can choose to use the built-in flow controllers and optionally modify their properties or can create a custom flow controller.

The built-in **com.rti.dds.publication.FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 76) allows data to be sent only when the user calls **com.rti.dds.publication.FlowController.trigger_flow** (p. 1058). With each trigger, all accumulated data since the previous trigger is sent (across all **com.rti.dds.publication.Publisher** (p. 1466) or **com.rti.dds.publication.DataWriter** (p. 553) instances). In other words, the network traffic shape is fully controlled by the user. Any accumulated samples destined for the same destination are coalesced into as few network packets as possible.

This external trigger source is ideal for users who want to implement some form of closed-loop flow control or who want to only put data on the wire every so many samples (e.g. with the number of samples based on **com.rti.ndds.↩ transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404)).

Essentially, this is equivalent to a user-created **com.rti.dds.publication.FlowController** (p. 1055) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059) = com.rti.dds.publication.↩ FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `max_tokens` = **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_added_per_period` = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `tokens_leaked_per_↩ period` = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `period` = **com.rti.dds.↩ infrastructure.Duration_t.DURATION_INFINITE** (p. 846)
- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) `bytes_per_token` = **com.↩ rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

See also

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 721)

com.rti.dds.publication.FlowController.trigger_flow (p. 1058)

com.rti.dds.publication.FlowController.set_property (p. 1057)

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 339)

6.20 Subscription Module

Contains the **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.subscription.DataReader** (p. 450), **com.↩ rti.dds.subscription.ReadCondition** (p. 1514), **com.rti.dds.subscription.QueryCondition** (p. 1510), and **com.↩ rti.dds.subscription.TopicQuery** (p. 1830) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1755) and **com.rti.dds.subscription.DataReaderListener** (p. 497) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**

  com.rti.dds.subscription.Subscriber *(p. 1730) entity and associated elements*
- **DataReaders**

  com.rti.dds.subscription.DataReader *(p. 450) entity and associated elements*
- **Data Samples**

  com.rti.dds.subscription.SampleInfo *(p. 1634),* ***com.rti.dds.subscription.SampleStateKind*** *(p. 1663),* ***com.rti.↩ dds.subscription.ViewStateKind*** *(p. 1970),* ***com.rti.dds.subscription.InstanceStateKind*** *(p. 1160) and associated elements*

6.20.1 Detailed Description

Contains the **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.subscription.DataReader** (p. 450), **com.↩ rti.dds.subscription.ReadCondition** (p. 1514), **com.rti.dds.subscription.QueryCondition** (p. 1510), and **com.↩ rti.dds.subscription.TopicQuery** (p. 1830) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1755) and **com.rti.dds.subscription.DataReaderListener** (p. 497) interfaces, and more generally, all that is needed on the subscription side.

"DCPS Subscription package"

6.20.2 Access to data samples

Data is made available to the application by the following operations on **com.rti.dds.subscription.DataReader** (p. 450) objects: **com.rti.ndds.example.FooDataReader.read** (p. 1069), **com.rti.ndds.example.FooDataReader.↩ read_w_condition** (p. 1076), **com.rti.ndds.example.FooDataReader.take** (p. 1071), **com.rti.ndds.example.Foo↩ DataReader.take_w_condition** (p. 1077), and the other variants of read() and take().

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

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

Each of these operations returns a collection of `Data` values and associated **com.rti.dds.subscription.SampleInfo** (p. 1634) objects. Each data value represents an atom of data information (i.e., a value for one instance). This collection may contain samples related to the same or different instances (identified by the key). Multiple samples can refer to the same instance if the settings of the **HISTORY** (p. 237) QoS allow for it.

These operations reset the read communication statuses; see **Changes in read communication status** (p. 264).

To return the memory back to the middleware, every read() or take() that retrieves a sequence of samples must be followed with a call to com.rti.ndds.example.FooDataReader.return_loan (p. 1094).

See also

Interpretation of the SampleInfo (p. 1636)

6.20.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **com.rti.dds.subscription.DataReader** (p. 450). These operations return an ordered collection of `DataSamples` consisting of a **com.rti.dds.subscription.SampleInfo** (p. 1634) part and a `Data` part.

The way RTI Connext builds the collection depends on QoS policies set on the **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.subscription.Subscriber** (p. 1730), as well as the `source_timestamp` of the samples, and the parameters passed to the `read()` / `take()` operations, namely:

- the desired sample states (any combination of **com.rti.dds.subscription.SampleStateKind** (p. 1663))
- the desired view states (any combination of **com.rti.dds.subscription.ViewStateKind** (p. 1970))
- the desired instance states (any combination of **com.rti.dds.subscription.InstanceStateKind** (p. 1160))

The `read()` and `take()` operations are non-blocking and just deliver what is currently available that matches the specified states.

The `read_w_condition()` and `take_w_condition()` operations take a **com.rti.dds.subscription.ReadCondition** (p. 1514) object as a parameter instead of sample, view or instance states. The behaviour is that the samples returned will only be those for which the condition is com.rti.dds.infrastructure.true. These operations, in conjunction with **com.rti.dds.subscription.ReadCondition** (p. 1514) objects and a **com.rti.dds.infrastructure.WaitSet** (p. 1973), allow performing waiting reads.

Once the data samples are available to the data readers, they can be read or taken by the application. The basic rule is that the application may do this in any order it wishes. This approach is very flexible and allows the application ultimate control.

To access data coherently, or in order, the **PRESENTATION** (p. 247) QoS must be set properly.

6.21 Subscribers

com.rti.dds.subscription.Subscriber (p. 1730) entity and associated elements

Classes

- interface **Subscriber**

  *<<**interface**>> (p. 156) A subscriber is the object responsible for actually receiving data from a subscription.*
- class **SubscriberAdapter**

  A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)
- interface **SubscriberListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for status about a subscriber.*
- class **SubscriberQos**

  *QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1730) entity.*
- class **SubscriberSeq**

  *Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1730) > .*

Variables

- static final **DataReaderQos** **DATAREADER_QOS_DEFAULT**

  Special value for creating data reader with default QoS.
- static final **DataReaderQos** **DATAREADER_QOS_USE_TOPIC_QOS** = new **DataReaderQos**()

  *Special value for creating **com.rti.dds.subscription.DataReader** (p. 450) with a combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 517) and the **com.rti.dds.topic.TopicQos** (p. 1824).*
- static final **DataReaderQos** **DATAREADER_QOS_PRINT_ALL** = new **DataReaderQos**()

  *Special value which can be supplied to **com.rti.dds.subscription.DataReaderQos.toString(DataReaderQos baseQos, QosPrintFormat format)** (p. 520) indicating that all of the QoS should be printed.*

6.21.1 Detailed Description

com.rti.dds.subscription.Subscriber (p. 1730) entity and associated elements

6.21.2 Variable Documentation

6.21.2.1 DATAREADER_QOS_DEFAULT

```
final DataReaderQos DATAREADER_QOS_DEFAULT [static]
```

Initial value:

```
=
    new DataReaderQos()
```

Special value for creating data reader with default QoS.

When used in **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735), this special value is used to indicate that the **com.rti.dds.subscription.DataReader** (p. 450) should be created with the default **com.rti.dds.subscription.DataReader** (p. 450) QoS by means of the operation get_default_datareader_qos and using the resulting QoS to create the **com.rti.dds.subscription.DataReader** (p. 450).

When used in **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734), this special value is used to indicate that the default QoS should be reset back to the initial value that would be used if the **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734) operation had never been called.

When used in **com.rti.dds.subscription.DataReader.set_qos** (p. 457), this special value is used to indicate that the QoS of the **com.rti.dds.subscription.DataReader** (p. 450) should be changed to match the current default QoS set in the **com.rti.dds.subscription.Subscriber** (p. 1730) that the **com.rti.dds.subscription.DataReader** (p. 450) belongs to.

Warning

This value is a constant and should never be modified. You cannot use this value to *get* the default QoS values for a **DataReader** (p. 450); for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 686).

See also

com.rti.dds.subscription.Subscriber.create_datareader (p. 1735)

com.rti.dds.subscription.Subscriber.set_default_datareader_qos (p. 1734)

com.rti.dds.subscription.DataReader.set_qos (p. 457)

6.21.2.2 DATAREADER_QOS_USE_TOPIC_QOS

```
final DataReaderQos DATAREADER_QOS_USE_TOPIC_QOS = new DataReaderQos() [static]
```

Special value for creating **com.rti.dds.subscription.DataReader** (p. 450) with a combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 517) and the **com.rti.dds.topic.TopicQos** (p. 1824).

The use of this value is equivalent to the application obtaining the default **com.rti.dds.subscription.DataReaderQos** (p. 517) and the **com.rti.dds.topic.TopicQos** (p. 1824) (by means of the operation **com.rti.dds.topic.Topic.get_qos** (p. 1810)) and then combining these two QoS using the operation **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1751) whereby any policy that is set on the **com.rti.dds.topic.TopicQos** (p. 1824) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **com.rti.dds.subscription.DataReader** (p. 450).

This value should only be used in **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735).

See also

com.rti.dds.subscription.Subscriber.create_datareader (p. 1735)

com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1733)

com.rti.dds.topic.Topic.get_qos (p. 1810)

com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1751)

6.21.2.3 DATAREADER_QOS_PRINT_ALL

```
final DataReaderQos DATAREADER_QOS_PRINT_ALL = new DataReaderQos() [static]
```

Special value which can be supplied to **com.rti.dds.subscription.DataReaderQos.toString(DataReaderQos baseQos, QosPrintFormat format)** (p. 520) indicating that all of the QoS should be printed.

The **com.rti.dds.subscription.DataReaderQos.toString(DataReaderQos baseQos, QosPrintFormat format)** (p. 520) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_PRINT_ALL** (p. 81) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_PRINT_ALL** (p. 81) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the print_private field within the **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507) structure.

This value should only be used as the base parameter to the **com.rti.dds.subscription.DataReaderQos.toString(DataReaderQos baseQos, QosPrintFormat format)** (p. 520) API.

Referenced by **DataReaderQos.toString()**.

6.22 DataReaders

com.rti.dds.subscription.DataReader (p. 450) entity and associated elements

Modules

- **Read Conditions**

  com.rti.dds.subscription.ReadCondition *(p. 1514) and associated elements*
- **Query Conditions**

  com.rti.dds.subscription.QueryCondition *(p. 1510) and associated elements*
- **Topic Queries**

  com.rti.dds.subscription.TopicQuery *(p. 1830) and associated elements.*

Classes

- class **FooDataReader**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type-specific data reader.*
- interface **DataReader**

  *<<**interface**>> (p. 156) Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1730).*
- class **DataReaderAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- class **DataReaderCacheStatus**

  *<<**extension**>> (p. 155) The status of the reader's cache.*
- interface **DataReaderListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for reader status.*
- class **DataReaderProtocolStatus**

  *<<**extension**>> (p. 155) The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.*
- class **DataReaderQos**

  *QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 450) entity.*
- class **DataReaderSeq**

  *Declares IDL `sequence` < **com.rti.dds.subscription.DataReader** (p. 450) > .*
- class **LivelinessChangedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS
- class **RequestedDeadlineMissedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS
- class **RequestedIncompatibleQosStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS
- class **SampleLostStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS
- class **SampleLostStatusKind**

  *<<**extension**>> (p. 155) Kinds of reasons why a sample was lost.*
- class **SampleRejectedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS
- class **SampleRejectedStatusKind**

  Kinds of reasons for rejecting a sample.
- class **SubscriptionMatchedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS

6.22.1 Detailed Description

com.rti.dds.subscription.DataReader (p. 450) entity and associated elements

6.23 Read Conditions

com.rti.dds.subscription.ReadCondition (p. 1514) and associated elements

Classes

- interface **ReadCondition**

  *<<**interface**>> (p. 156) Conditions specifically dedicated to read operations and attached to one **com.rti.dds.subscription.DataReader** (p. 450).*
- class **ReadConditionParams**

  *<<**extension**>> (p. 155) Input parameters for **com.rti.dds.subscription.DataReader.create_readcondition_w_params** (p. 455)*

6.23.1 Detailed Description

com.rti.dds.subscription.ReadCondition (p. 1514) and associated elements

6.24 Query Conditions

com.rti.dds.subscription.QueryCondition (p. 1510) and associated elements

Classes

- interface **QueryCondition**

  *<<**interface**>> (p. 156) These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1514) objects that allow the application to also specify a filter on the locally available data.*
- class **QueryConditionParams**

  *<<**extension**>> (p. 155) Input parameters for **com.rti.dds.subscription.DataReader.create_querycondition_w_params** (p. 456)*

6.24.1 Detailed Description

com.rti.dds.subscription.QueryCondition (p. 1510) and associated elements

6.25 Data Samples

com.rti.dds.subscription.SampleInfo (p. 1634), **com.rti.dds.subscription.SampleStateKind** (p. 1663), **com.rti.dds.subscription.ViewStateKind** (p. 1970), **com.rti.dds.subscription.InstanceStateKind** (p. 1160) and associated elements

Modules

- **Sample States**

  com.rti.dds.subscription.SampleStateKind *(p. 1663) and associated elements*
- **View States**

  com.rti.dds.subscription.ViewStateKind *(p. 1970) and associated elements*
- **Instance States**

  com.rti.dds.subscription.InstanceStateKind *(p. 1160) and associated elements*
- **Stream Kinds**

  com.rti.dds.subscription.StreamKind *(p. 1713) and associated elements*

Classes

- class **CoherentSetInfo_t**

  *<<**extension**>> (p. 155) Type definition for a coherent set info.*
- class **SampleInfo**

  Information that accompanies each sample that is `read` *or* `taken`*.*
- class **SampleInfoSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.subscription.SampleInfo*** *(p. 1634)* *>* *.*

6.25.1 Detailed Description

com.rti.dds.subscription.SampleInfo (p. 1634), **com.rti.dds.subscription.SampleStateKind** (p. 1663), **com.rti.dds.subscription.ViewStateKind** (p. 1970), **com.rti.dds.subscription.InstanceStateKind** (p. 1160) and associated elements

6.26 Topic Queries

com.rti.dds.subscription.TopicQuery (p. 1830) and associated elements.

Classes

- interface **TopicQuery**

  <<***extension***>> (p. *155) Allows a* ***com.rti.dds.subscription.DataReader*** (p. *450) to query the sample cache of its matching* ***com.rti.dds.publication.DataWriter*** (p. *553).*
- class **TopicQueryData**

  <<***extension***>> (p. *155) Provides information about a* ***com.rti.dds.subscription.TopicQuery*** (p. *1830)*
- class **TopicQueryHelper**

  Helpers to provide utility operations related to ***com.rti.dds.subscription.TopicQuery*** (p. *1830).*
- class **TopicQuerySelection**

  <<***extension***>> (p. *155) Specifies the data query that defines a* ***com.rti.dds.subscription.TopicQuery*** (p. *1830)*
- class **TopicQuerySelectionKind**

  Kinds of ***TopicQuerySelection*** (p. *1836).*

Variables

- static final **TopicQuerySelection TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER**

  Special value for creating a ***com.rti.dds.subscription.TopicQuery*** (p. *1830) that applies the same filter as the* ***Data↩ Reader*** (p. *450)'s* ***com.rti.dds.topic.ContentFilteredTopic*** (p. *436).*
- static final **TopicQuerySelection TOPIC_QUERY_SELECTION_SELECT_ALL**

  Special value for creating a ***com.rti.dds.subscription.TopicQuery*** (p. *1830) that selects all the samples in a* ***com.rti.↩ dds.publication.DataWriter*** (p. *553) cache.*

6.26.1 Detailed Description

com.rti.dds.subscription.TopicQuery (p. 1830) and associated elements.

TopicQueries allow a **com.rti.dds.subscription.DataReader** (p. 450) to query the sample cache of its matching **com.↩ rti.dds.publication.DataWriter** (p. 553). A user can create a **com.rti.dds.subscription.TopicQuery** (p. 1830) with the **com.rti.dds.subscription.DataReader.create_topic_query** (p. 473) API. When a DataReader creates a TopicQuery, DDS will propagate it to other DomainParticipants and their DataWriters. When a DataWriter matching with the DataReader that created the TopicQuery receives it, it will send the cached samples that pass the TopicQuery's filter.

To configure how to dispatch a TopicQuery, the **com.rti.dds.publication.DataWriterQos** (p. 612) includes the **com.↩ rti.dds.infrastructure.TopicQueryDispatchQosPolicy** (p. 1833) policy. By default, a DataWriter ignores TopicQueries unless they are explicitly enabled using this policy.

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

Late-joining DataWriters will also discover existing TopicQueries. To delete a TopicQuery you must use **com.rti.dds.↩ subscription.DataReader.delete_topic_query** (p. 473).

After deleting a TopicQuery, new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them.

By default, a TopicQuery queries the samples that were in the DataWriter cache at the time the DataWriter receives the TopicQuery. However a TopicQuery can be created in a "continuous" mode. A DataWriter will continue delivering samples that pass a continuous TopicQuery filter until the DataReader application explicitly deletes it.

The samples received in response to a TopicQuery are stored in the associated DataReader's cache. Any of the read/take operations can retrieve TopicQuery samples. The field **com.rti.dds.subscription.SampleInfo.topic_query_guid** (p. 1646) associates each sample to its TopicQuery. If the read sample is not in response to a TopicQuery then this field will be **com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134). Note that the same data may be received several times, depending on how many TopicQueries the DataReader creates and their TopicQuerySelection.

You can choose to read or take only TopicQuery samples, only live samples, or both. To support this ReadConditions and QueryConditions provide the **com.rti.dds.subscription.DataReader.create_querycondition_w_params** (p. 456) and **com.rti.dds.subscription.DataReader.create_readcondition_w_params** (p. 455) APIs.

Each TopicQuery is identified by a GUID that can be accessed using the **com.rti.dds.subscription.TopicQuery.get_guid** (p. 1830) method.

6.26.2 Debugging Topic Queries

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

6.26.2.1 The Built-in ServiceRequest DataReader

TopicQueries are communicated to publishing applications through a built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) channel. The ServiceRequest channel is designed to be generic so that it can be used for many different purposes, one of which is TopicQueries.

When a DataReader creates a TopicQuery, a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) message is sent containing the TopicQuery information. Just as there are built-in DataReaders for **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349), **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762), and **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452), there is a fourth built-in DataReader for ServiceRequests.

The new built-in DataReader can be retrieved using the built-in subscriber and **com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1740). The topic name is **com.rti.dds.topic.builtin.ServiceRequestTypeSupport.SERVICE_REQUEST_TOPIC_NAME** (p. 168). Installing a listener with the **com.rti.dds.subscription.DataReaderListener.on_data_available** (p. 500) callback implemented will allow a publishing application to be notified whenever a TopicQuery has been received from a subscribing application.

The builtin.ServiceRequest.service_id of a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) corresponding to a **com.rti.dds.subscription.TopicQuery** (p. 1830) will be **com.rti.dds.topic.builtin.ServiceRequest.TOPIC_QUERY_SERVICE_ID** (p. 167) and the builtin.ServiceRequest.instance_id will be equal to the GUID of the **com.rti.dds.subscription.TopicQuery** (p. 1830) (see **com.rti.dds.subscription.TopicQuery.get_guid** (p. 1830)).

The builtin.ServiceRequest.request_body is a sequence of bytes containing more information about the TopcQuery. This information can be retrieved using the **com.rti.dds.subscription.TopicQueryHelper.topic_query_data_from_service_request** (p. 1835) function. The resulting **com.rti.dds.subscription.TopicQueryData** (p. 1830) contains the **com.rti.dds.subscription.TopicQuerySelection** (p. 1836) that the **com.rti.dds.subscription.TopicQuery** (p. 1830) was created with, the GUID of the original DataReader that created the TopicQuery, and the topic name of that DataReader. Note: When TopicQueries are propagated through one or more Routing Services, the last DataReader that issued the TopicQuery will be a Routing Service DataReader. The **com.rti.dds.subscription.TopicQueryData.original_related_reader_guid** (p. 1832), however, will be that of the first DataReader to have created the TopicQuery.

6.26.2.2 The on_service_request_accepted DataWriter Listener Callback

It is possible that a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for a **com.rti.dds.subscription.TopicQuery** (p. 1830) is received but is not immediately dispatched to a DataWriter. This can happen, for example, if a DataWriter was not matching with a DataReader at the time that the TopicQuery was received by the publishing application. The **com.rti.dds.publication.DataWriterListener.on_service_request_accepted** (p. 596) callback notifies a DataWriter when a ServiceRequest has been dispatched to that DataWriter. The **com.rti.dds.publication.ServiceRequestAcceptedStatus** (p. 1677) provides information about how many ServiceRequests have been accepted by the DataWriter since the last time that the status was read. The status also includes the **com.rti.dds.publication.ServiceRequestAcceptedStatus.last_request_handle** (p. 1679), which is the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of the last ServiceRequest that was accepted. This instance handle can be used to read samples per instance from the built-in ServiceRequest DataReader and correlate which ServiceRequests have been dispatched to which DataWriters.

6.26.2.3 Reading TopicQuery Samples

Data samples that are received by a DataReader in response to a TopicQuery can be identified with two pieces of information from the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634) to the sample. First, if the **com.rti.dds.subscription.SampleInfo.topic_query_guid** (p. 1646) is not equal to **com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134) then the sample is in response to the TopicQuery with that GUID (see **com.rti.dds.subscription.TopicQuery.get_guid** (p. 1830)). Second, if the sample is in response to a TopicQuery and the **com.rti.dds.subscription.SampleInfo.flag** (p. 1645) **com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_TOPIC_QUERY_SAMPLE** (p. 1631) flag is set then this is not the last sample in response to the TopicQuery for a DataWriter identified by **com.rti.dds.subscription.SampleInfo.original_publication_virtual_guid** (p. 1644). If that flag is not set then there will be no more samples corresponding to that TopicQuery coming from the DataWriter.

6.26.3 Variable Documentation

6.26.3.1 TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER

```
final TopicQuerySelection TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER [static]
```

Special value for creating a **com.rti.dds.subscription.TopicQuery** (p. 1830) that applies the same filter as the **DataReader** (p. 450)'s **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

If the **com.rti.dds.subscription.DataReader** (p. 450) that creates the **com.rti.dds.subscription.TopicQuery** (p. 1830) uses a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), this **com.rti.dds.subscription.TopicQuerySelection** (p. 1836) value indicates that the **TopicQuery** (p. 1830) should use the same filter.

Otherwise it behaves as **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 87).

6.26.3.2 TOPIC_QUERY_SELECTION_SELECT_ALL

```
final TopicQuerySelection TOPIC_QUERY_SELECTION_SELECT_ALL [static]
```

Special value for creating a **com.rti.dds.subscription.TopicQuery** (p. 1830) that selects all the samples in a **com.rti.dds.publication.DataWriter** (p. 553) cache.

6.27 Sample States

com.rti.dds.subscription.SampleStateKind (p. 1663) and associated elements

Classes

- class **SampleStateKind**

  Indicates whether or not a sample has ever been read.

Variables

- static final int **ANY_SAMPLE_STATE** = 0xffff

  Any sample state com.rti.dds.subscription.SampleStateKind.SampleStateKind.READ_SAMPLE_STATE | com.rti.dds.↩ subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE.

6.27.1 Detailed Description

com.rti.dds.subscription.SampleStateKind (p. 1663) and associated elements

6.27.2 Variable Documentation

6.27.2.1 ANY_SAMPLE_STATE

```
final int ANY_SAMPLE_STATE = 0xffff  [static]
```

Any sample state com.rti.dds.subscription.SampleStateKind.SampleStateKind.READ_SAMPLE_STATE | com.rti.dds.↩ subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE.

Referenced by **Requester< TReq, TRep >.readReplies()**, **Requester< TReq, TRep >.readReply()**, **Requester< TReq, TRep >.takeReplies()**, **Requester< TReq, TRep >.takeReply()**, and **Requester< TReq, TRep >.waitFor↩ Replies()**.

6.28 View States

com.rti.dds.subscription.ViewStateKind (p. 1970) and associated elements

Classes

- class **ViewStateKind**

  Indicates whether or not an instance is new.

Variables

- static final int **ANY_VIEW_STATE** = 0xffff

  Any view state com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE | com.rti.dds.subscription.↩ ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE.

6.28.1 Detailed Description

com.rti.dds.subscription.ViewStateKind (p. 1970) and associated elements

6.28.2 Variable Documentation

6.28.2.1 ANY_VIEW_STATE

```
final int ANY_VIEW_STATE = 0xffff  [static]
```

Any view state com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE | com.rti.dds.subscription.↩ ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE.

6.29 Instance States

com.rti.dds.subscription.InstanceStateKind (p. 1160) and associated elements

Classes

- class **InstanceStateKind**

  *Indicates if the samples are from a live **com.rti.dds.publication.DataWriter** (p. 553) or not.*

Variables

- static final int **ANY_INSTANCE_STATE** = 0xffff

  Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE.
- static final int **NOT_ALIVE_INSTANCE_STATE** = 0x006

  Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

6.29.1 Detailed Description

com.rti.dds.subscription.InstanceStateKind (p. 1160) and associated elements

6.29.2 Variable Documentation

6.29.2.1 ANY_INSTANCE_STATE

```
final int ANY_INSTANCE_STATE = 0xffff  [static]
```

Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE.

6.29.2.2 NOT_ALIVE_INSTANCE_STATE

```
final int NOT_ALIVE_INSTANCE_STATE = 0x006  [static]
```

Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_↩ STATE.

6.30 Stream Kinds

com.rti.dds.subscription.StreamKind (p. 1713) and associated elements

Classes

- class **StreamKind**

  *Indicates if the samples are **TopicQuery** (p. 1830) samples or not.*

Variables

- static final int **ANY_STREAM** = 0xffff

  Any stream kind LIVE_STREAM | TOPIC_QUERY_STREAM.

6.30.1 Detailed Description

com.rti.dds.subscription.StreamKind (p. 1713) and associated elements

6.30.2 Variable Documentation

6.30.2.1 ANY_STREAM

```
final int ANY_STREAM = 0xffff  [static]
```

Any stream kind LIVE_STREAM | TOPIC_QUERY_STREAM.

6.31 Infrastructure Module

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

Modules

- **Clock Selection**

  APIs related to clock selection.
- **Builtin Qos Profiles**

  *<<**extension**>> (p. 155) QoS libraries, profiles, and snippets that are automatically built into RTI Connext.*
- **Conditions and WaitSets**

  ***com.rti.dds.infrastructure.Condition** (p. 429) and **com.rti.dds.infrastructure.WaitSet** (p. 1973) and related items.*
- **Time Support**

  Time and duration types and defines.
- **Entity Support**

  ***com.rti.dds.infrastructure.Entity** (p. 1029), **com.rti.dds.infrastructure.Listener** (p. 1236) and related items.*
- **GUID Support**

  *<<**extension**>> (p. 155) GUID type and defines.*
- **Object Support**

  *<<**extension**>> (p. 155) Object related items.*
- **QoS Policies**

  Quality of Service (QoS) policies.
- **Return Codes**

  Types of return codes.
- **Sequence Number Support**

  *<<**extension**>> (p. 155) Sequence number type and defines.*
- **Status Kinds**

  Kinds of communication status.
- **Exception Codes**

  *<<**extension**>> (p. 155) Exception codes.*
- **Sequence Support**

  The com.rti.dds.infrastructure.com.rti.dds.util.Sequence interface allows you to work with variable-length collections of homogeneous data.

Classes

- class **Enum**

  A superclass for all type-safe enumerated types.

6.31.1 Detailed Description

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

"DCPS Infrastructure package"

6.32 Built-in Sequences

Defines sequences of primitive data type. .

Classes

- class **BooleanSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *boolean* >
- class **ByteSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *byte* >
- class **CharSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *char* >
- class **DoubleSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *double* >
- class **FloatSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *float* >
- class **IntSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.infrastructure.↩ int* >
- class **LongDoubleSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.infrastructure.↩ LongDouble* >
- class **LongSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *long* >
- class **ShortSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *short* >
- class **StringSeq**

  Declares IDL `sequence` < *com.rti.dds.infrastructure.String* > .
- class **WcharSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.infrastructure.↩ char* >
- class **WstringSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.infrastructure.↩ char∗* >

6.32.1 Detailed Description

Defines sequences of primitive data type. .

6.33 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

APIs related to Multi-channel DataWriters.

6.33.1 What is a Multi-channel DataWriter?

A Multi-channel **com.rti.dds.publication.DataWriter** (p. 553) is a **com.rti.dds.publication.DataWriter** (p. 553) that is configured to send data over multiple multicast addresses, according to some filtering criteria applied to the data.

To determine which multicast addresses will be used to send the data, the middleware evaluates a set of filters that are configured for the **com.rti.dds.publication.DataWriter** (p. 553). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **com.rti.dds.publication.DataWriter** (p. 553) writes data, the filters are applied. If a filter evaluates to true, the data is sent over that filter's associated channel (set of multicast addresses). We refer to this type of filter as a Channel Guard filter.

6.33.2 Configuration on the Writer Side

To configure a multi-channel **com.rti.dds.publication.DataWriter** (p. 553), simply define a list of all its channels in the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318).

The **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) is propagated along with discovery traffic. The value of this policy is available in builtin.PublicationBuiltinTopicData.locator_filter.

6.33.3 Configuration on the Reader Side

No special changes are required in a subscribing application to get data from a multichannel **com.rti.dds.publication.↩ DataWriter** (p. 553). If you want the **com.rti.dds.subscription.DataReader** (p. 450) to subscribe to only a subset of the channels, use a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

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

6.33.4 Reliability with Multi-Channel DataWriters

6.33.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccess↩ ScopeKind.INSTANCE_PRESENTATION_QOS and the filters in **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) are keyed-only based.

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

6.33.4.2 Reliable Protocol Considerations

Reliability is maintained on a per-channel basis. Each channel has its own reliability channel send queue. The size of that queue is limited by **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) and/or **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628).

The protocol parameters described in **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596) are applied per channel, with the following exceptions:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1607) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1607): The low watermark and high watermark control the queue levels (in number of samples) that determine when to switch between regular and fast heartbeat rates. With MultiChannel DataWriters, high_watermark and low_watermark refer to the DataWriter's queue (not the reliability channel queue). Therefore, periodic heartbeating cannot be controlled on a per-channel basis.

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

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeats_per_max_samples (p. 1611): This field defines the number of heartbeats per send queue. For MultiChannel DataWriters, the value is applied per channel. However, the send queue size that is used to calculate the a piggyback heartbeat rate is defined per DataWriter (see **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592))

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

With batching and MultiChannel DataWriters, the size of the DataWriter's send queue should be configured using **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) instead of max_batches **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628) in order to take advantage of heartbeats_per_max_samples.

6.34 Transports

APIs related to RTI Connext pluggable transports.

Modules

- **Installing Transport Plugins**

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

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

  Developing new transport plugins for RTI Connext.

6.34.1 Detailed Description

APIs related to RTI Connext pluggable transports.

6.34.2 Overview

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

A transport plugin implements the abstract transport API and performs the actual work of sending and receiving messages over a physical transport. A collection of **builtin plugins** (see **Built-in Transport Plugins** (p. 102)) is delivered with RTI Connext for commonly used transports. New transport plugins can easily be created, thus enabling RTI Connext applications to run over transports that may not even be conceived yet. This is a powerful capability and that distinguishes RTI Connext from competing middleware approaches.

RTI Connext also provides a set of APIs for installing and configuring transport plugins to be used in an application. So that RTI Connext applications work out of the box, a subset of the builtin transport plugins is preconfigured by default (see **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843)). You can "turn-off" some or all of the builtin transport plugins. In addition, you can configure other transport plugins for use by the application.

6.34.3 Transport Aliases

In order to use a transport plugin instance in an RTI Connext application, it must be registered with a **com.rti.dds.domain.DomainParticipant** (p. 670). When you register a transport, you specify a sequence of "alias" strings to symbolically refer to the transport plugin. The same alias strings can be used to register more than one transport plugin.

You can register multiple transport plugins with a **com.rti.dds.domain.DomainParticipant** (p. 670). An **alias** symbolically refers to one or more transport plugins registered with the **com.rti.dds.domain.DomainParticipant** (p. 670). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 269)).

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

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

- the transport plugins to use for **discovery** (see **com.rti.dds.infrastructure.DiscoveryQosPolicy.enabled_transports** (p. 667)), and for **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities (see **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1860)).

- the **multicast** addresses on which to receive discovery messages (see **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667)), and the multicast addresses and ports on which to receive user data (see **com.rti.dds.subscription.DataReaderQos.multicast** (p. 524)).

- the **unicast ports** used for user data (see **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1867)) on both **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities.

- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 223) and **NDDS_DISCOVERY_PEERS** (p. 222)).

A **com.rti.dds.domain.DomainParticipant** (p. 670) (and contained its entities) start using a transport plugin after the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled (see **com.rti.dds.infrastructure.Entity.enable** (p. 1032)). An entity will use *all* the transport plugins that match the specified transport QoS policy. All transport plugins are treated uniformly, regardless of how they were created or registered; there is no notion of some transports being more "special" that others.

6.34.4 Transport Lifecycle

A transport plugin is owned by whomever creates it. Thus, if you create and register a transport plugin with a **com.rti.dds.domain.DomainParticipant** (p. 670), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 269)) and transport plugins that are loaded through the **PROPERTY** (p. 248) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 100)) are automatically managed by RTI Connext.

A user-created transport plugin must not be deleted while it is still in use by a **com.rti.dds.domain.DomainParticipant** (p. 670). This generally means that a user-created transport plugin instance can only be deleted after the **com.rti.dds.domain.DomainParticipant** (p. 670) with which it was registered is deleted (see **com.rti.dds.domain.DomainParticipantFactory.delete_participant** (p. 766)). Note that a transport plugin *cannot* be "unregistered" from a **com.rti.dds.domain.DomainParticipant** (p. 670).

A transport plugin instance cannot be registered with more than one **com.rti.dds.domain.DomainParticipant** (p. 670) at a time. This requirement is necessary to guarantee the multi-threaded safety of the transport API.

If the same physical transport resources are to be used with more than one **com.rti.dds.domain.DomainParticipant** (p. 670) in the same address space, the transport plugin should be written in such a way so that it can be instantiated multiple times—once for each **com.rti.dds.domain.DomainParticipant** (p. 670) in the address space. Note that it is always possible to write the transport plugin so that multiple transport plugin instances share the same underlying resources; however the burden (if any) of guaranteeing multi-threaded safety to access shared resource shifts to the transport plugin developer.

6.34.5 Transport Class Attributes

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

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

Every transport plugin must specify the following class attributes.

transport class id (see com.rti.ndds.transport.Transport.Property_t.classid (p. 1403)) Identifies a transport plugin implementation class. It denotes a unique "class" to which the transport plugin instance belongs. The class is used to distinguish between different transport plugin implementations. Thus, a transport plugin vendor should ensure that its transport plugin implementation has a unique class.

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

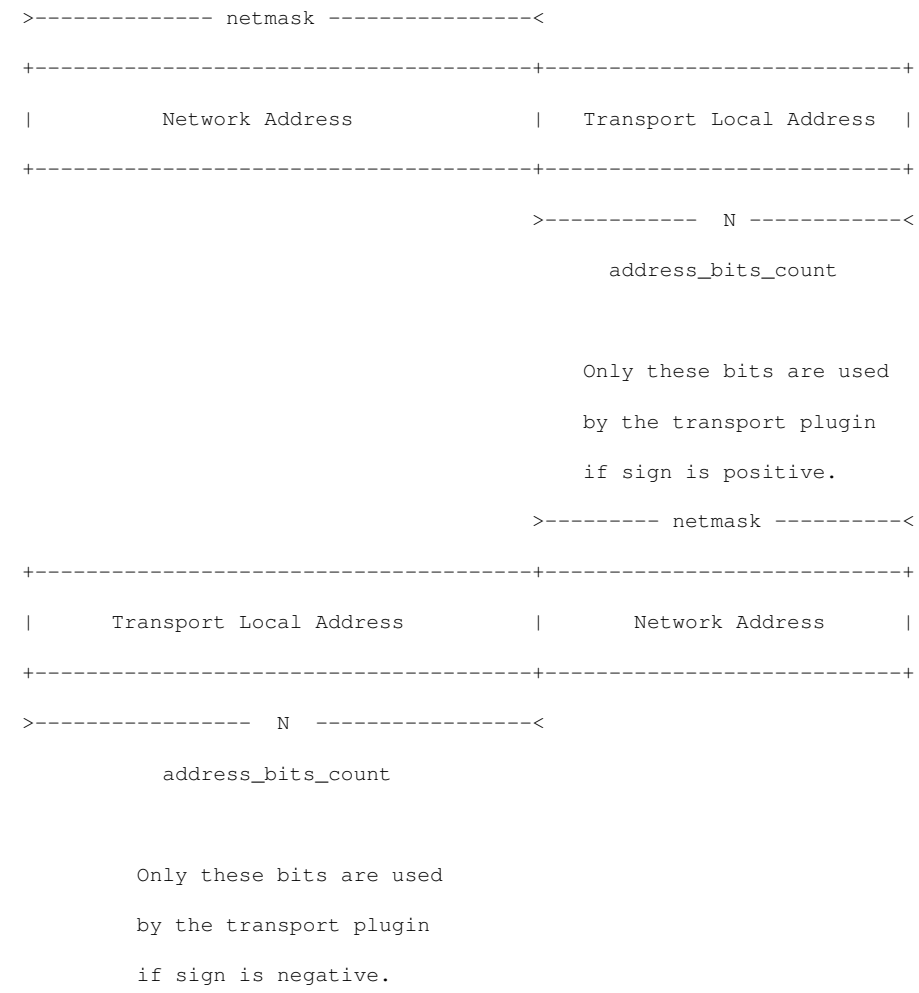
Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Connext application, provided they use different transport class IDs.

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

transport significant address bit count (see com.rti.ndds.transport.Transport.Property_t.address_bit_count (p. 1403))

RTI Connext's addressing is modeled after the IPv6 and uses 128-bit addresses (java.net.InetAddress) to route messages.

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



The remaining bits of an address using the 128 - abs(bit address) representation will be considered as part of the "network address" (see **Transport Network Address** (p. 98)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher (128 - **com.rti.ndds.transport.↩ Transport.Property_t.address_bit_count** (p. 1403)) bits. RTI Connext is free to manipulate those bits freely in the addresses passed in/out to the transport plugin APIs.

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

$$address\_bits\_count >= ceil(log_2(total\_addressable\_transport\_unicast\_interfaces))$$

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

6.34.6 Transport Instance Attributes

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

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

RTI Connext requires that every transport plugin instance must specify the **com.rti.ndds.transport.Transport.↩ Property_t.message_size_max** (p. 1404) and **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer↩ _count_max** (p. 1404).

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

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

6.34.7 Transport Network Address

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

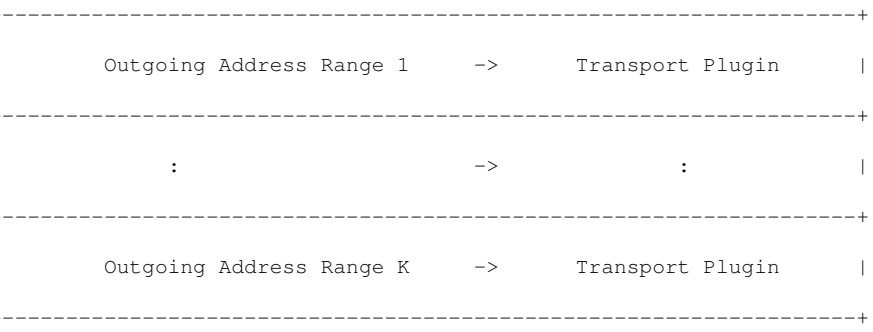
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a transport plugin via the com.rti.ndds.transport.TransportSupport.register_transport()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

The network address for a transport plugin should be chosen such that the resulting fully qualified 128-bit address will be unique in the RTI Connext domain. Thus, if two instances of a transport plugin are registered with a **com.rti.↩ dds.domain.DomainParticipant** (p. 670), they will be at different network addresses in order for their unicast interfaces to have unique fully qualified 128-bit addresses. It is also possible to create multiple transports with the same network address, as it can be useful for certain use cases; note that this will require special entity configuration for most transports to avoid clashes in resource use (e.g. sockets for UDPv4 transport).

6.34.8 Transport Send Route

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

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

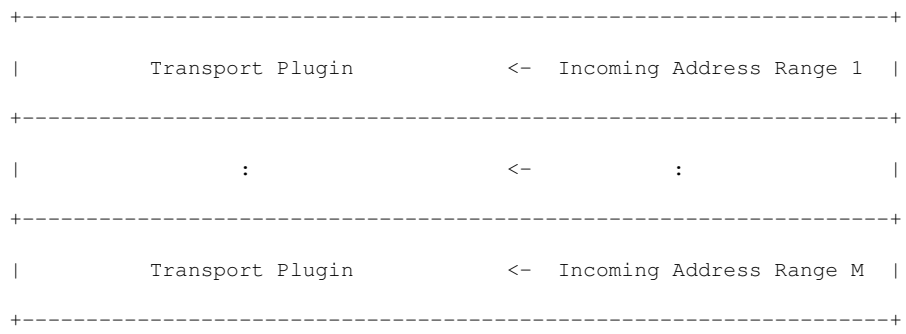


The user can set up a routing table to restrict the use of a transport plugin to send messages to selected addresses ranges.

6.34.9 Transport Receive Route

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

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



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

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

6.35 Installing Transport Plugins

Installing and configuring transports used by RTI Connext.

Classes

- class **TransportSupport**
  *<<**interface**>> (p. 156) The utility class used to configure RTI Connext pluggable transports.*

6.35.1 Detailed Description

Installing and configuring transports used by RTI Connext.

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

- If it is a builtin transport plugin, by specifying a bitmask in **com.rti.dds.infrastructure.TransportBuiltinQos↩ Policy** (p. 1843) (see **Built-in Transport Plugins** (p. 102))
- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 248) QoS policy settings of **com.rti.dds.domain.DomainParticipant** (p. 670) (only supported on architectures that support dynamic libraries, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 100))

The lifecycle of the transport plugin is automatically managed by RTI Connext. See **Transport Lifecycle** (p. 96) for details.

6.35.2 Loading Transport Plugins through Property QoS Policy of Domain Participant

On architectures that support dynamic libraries, a non-builtin transport plugin written in C/C++ and built as a dynamic-link library (∗.dll/∗.so) can be loaded by RTI Connext through the **PROPERTY** (p. 248) QoS policy settings of the **com.rti.dds.domain.DomainParticipant** (p. 670).

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

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

To allow dynamic loading of the transport plugin, the transport plugin must implement the RTI Connext abstract transport API and must provide a function with the signature com.rti.ndds.transport.NDDS_Transport_create_plugin that can be called by RTI Connext to create an instance of the transport plugin. The name of the dynamic library that contains the transport plugin implementation, the name of the function and properties that can be used to create the plugin, and the aliases and network address that are used to register the plugin can all be specified through the **PROPERTY** (p. 248) QoS policy of the **com.rti.dds.domain.DomainParticipant** (p. 670).

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

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

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

Property Name	Description	Required?
<TRANSPORT_PREFIX>.aliases	Used to register the transport plugin returned by com.rti.ndds.transport.NDDS_Transport_create_plugin (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **com.rti.dds.domain.DomainParticipant** (p. 670). Refer to **aliases_in** parameter in com.rti.ndds.transport.TransportSupport.register_transport for details. Aliases should be specified as a comma-separated string, with each comma delimiting an alias. If it is not specified, <TRANSPORT_PREFIX> –without the leading "dds.transport" – is used as the default alias for the plugin.	NO
<TRANSPORT_PREFIX>.network_address	Used to register the transport plugin returned by com.rti.ndds.transport.NDDS_Transport_create_plugin (as specified by <**TRANSPORT_PREFIX**>**.create_function**) to the **com.rti.dds.domain.DomainParticipant** (p. 670). Refer to **network_address_in** parameter in com.rti.ndds.transport.TransportSupport.register_transport for details. If it is not specified, the **network_address_out** output parameter from com.rti.ndds.transport.NDDS_Transport_create_plugin is used. The default value is a zeroed out network address.	NO
<TRANSPORT_PREFIX>.<property_name>	Property that is passed into com.rti.ndds.transport.NDDS_Transport_create_plugin (as specified by <**TRANSPORT_PREFIX**>**.create_function**) for creating the transport plugin. This property name-value pair will be passed to com.rti.ndds.transport.NDDS_Transport_create_plugin after stripping out <TRANSPORT_PREFIX> from the property name. The parsing of this property and configuring the transport using this property should be handled by the implementation of each transport plugin. Multiple <TRANSPORT_PREFIX>.<property_name> can be specified. Note: "library", "create_function", "aliases" and "network_address" cannot be used as the <property_name> due to conflicts with other builtin property names.	NO

A transport plugin is dynamically created and registered to the **com.rti.dds.domain.DomainParticipant** (p. 670) by RTI Connext when:

- the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled,
- the first DataWriter/DataReader is created, or

- you lookup a builtin DataReader (**com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1740)),

whichever happens first.

Any changes to the transport plugin related properties in the **PROPERTY** (p. 248) QoS policy after the transport plugin has been registered with the **com.rti.dds.domain.DomainParticipant** (p. 670) will have no effect.

6.36 Built-in Transport Plugins

Transport plugins delivered with RTI Connext.

Classes

- interface **ShmemTransport**

  Built-in transport plug-in for inter-process communications using shared memory.
- interface **UDPv4Transport**

  Transport *(*p. *1841) plug-in using UDP/IPv4.*
- interface **UDPv4WanTransport**

  Transport *(*p. *1841) plug-in using UDP/IPv4 for WAN communications..*
- interface **UDPv6Transport**

  Transport *(*p. *1841) plug-in using UDP/IPv6.*

6.36.1 Detailed Description

Transport plugins delivered with RTI Connext.

The **TRANSPORT_BUILTIN** (p. 269) specifies the collection of transport plugins that can be automatically configured and managed by RTI Connext as a convenience to the user.

These transport plugins can simply be turned "on" or "off" by a specifying a bitmask in **com.rti.dds.infrastructure.↩ TransportBuiltinQosPolicy** (p. 1843), thus bypassing the steps for setting up a transport plugin. RTI Connext preconfigures the transport plugin properties, the network address, and the aliases to "factory defined" values.

If a builtin transport plugin is turned "on" in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843), the plugin is implicitly created and registered to the corresponding **com.rti.dds.domain.DomainParticipant** (p. 670) by RTI Connext when:

- the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1740)),

whichever happens first.

Each builtin transport contains its own set of properties. For example, the **com.rti.ndds.transport.UDPv4Transport** (p. 1943) allows the application to specify whether or not multicast is supported, the maximum size of the message, and provides a mechanism for the application to filter out network interfaces.

The builtin transport plugin properties can be changed by the method **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property()** (p. 1863) or by using the **PROPERTY** (p. 248) QoS policy associated with the **com.rti.dds.domain.DomainParticipant** (p. 670). Builtin transport plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) always overwrite the ones specified through **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property()** (p. 1863). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 248) QoS policy.

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

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863) **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438)

6.37 Creating New Transport Plugins

Developing new transport plugins for RTI Connext.

Classes

- interface **Transport**

  RTI Connext's abstract pluggable transport interface.

6.37.1 Detailed Description

Developing new transport plugins for RTI Connext.

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

6.38 Queries and Filters Syntax

6.38.1 Syntax for DDS Queries and Filters

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

- The `filter_expression` in the **com.rti.dds.topic.ContentFilteredTopic** (p. 436)
- The `query_expression` in the **com.rti.dds.subscription.QueryCondition** (p. 1510)
- <<***extension***>> *(p. 155)* The `filter_expression` in the **com.rti.dds.subscription.TopicQuerySelection** (p. 1836)
- <<***extension***>> *(p. 155)* The `filter_expression` in the **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411)

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

The following notational conventions are made:

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

6.38.2 SQL grammar in BNF

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

6.38.3 Token expression

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

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

Formal notation:

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

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

Formal notation:

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

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

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

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

Formal notation:

```
TOPICNAME : IDENTIFIER
```

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

Formal notation:

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

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

Formal notation:

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

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

Formal notation:

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

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

Formal notation:

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

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

Formal notation:

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

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

Formal notation (case insensitive):

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

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

Formal notation (case insensitive):

```
NULLVALUE : "null"
```

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

Formal notation:

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

6.38.4 String Parameters

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

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

whereas the following expression will not compile:

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

6.38.5 Type compatability in Predicate

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

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

- (∗1) See **SQL Extension: Regular Expression Matching** (p. 108)
- (∗2) Because the formal notation of the Enumeration values, they are compatible with string and char literals, but they are not compatible with string or char variables, i.e., "MyEnum='EnumValue'" would be correct, but "My↩ Enum=MyString" is not allowed.
- (∗3) Only for same type Enums.

6.38.6 SQL Extension: Regular Expression Matching

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

MATCH is case-sensitive.

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

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

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

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

6.38.7 Character Encoding

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

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

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

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

6.38.8 Unicode Normalization

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

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

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

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

The possible values of the normalization property are:

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

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

6.38.9 Examples

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

Example of a **filter_expression** (for **com.rti.dds.topic.ContentFilteredTopic** (p. 436)) or a **query_expression** (for **com.rti.dds.subscription.QueryCondition** (p. 1510)):

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

Examples of a **filter_expression** using **MATCH** (for **com.rti.dds.topic.ContentFilteredTopic** (p. 436)) operator:

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

6.39 Logging and Version

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

Modules

- **Logging**

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

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

6.39.1 Detailed Description

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

6.40 General Utilities and Compliance Configuration

API of general utilities used in the RTI Connext distribution.

Modules

- **Heap Monitoring**

  Monitor memory allocations done by the middleware on the native heap.
- **Network Capture**

  Save network traffic into a capture file for further analysis.
- **Other Utilities**

  Other Utilities, such as com.rti.ndds.utility.Utility.spin.

6.40.1 Detailed Description

API of general utilities used in the RTI Connext distribution.

6.41 Observability

API of RTI Connext Observability Framework.

Modules

- **Observability Library**
  RTI Monitoring Library 2.0.

6.41.1 Detailed Description

API of RTI Connext Observability Framework.

6.42 Request-Reply Pattern

Support for the request-reply communication pattern.

Modules

- **Requester**
  com.rti.connext.requestreply.Requester<TReq,TRep> and associated elements
- **Replier**
  com.rti.connext.requestreply.Replier<TReq,TRep>, com.rti.connext.requestreply.SimpleReplier<TReq,TRep> and associated elements

6.42.1 Detailed Description

Support for the request-reply communication pattern.

There are two basic entities that enable this pattern:

- com.rti.connext.requestreply.Requester<TReq,TRep>
- com.rti.connext.requestreply.Replier<TReq,TRep> (and a simpler version com.rti.connext.requestreply.Simple↩ Replier<TReq,TRep>)

This functionality is built on top of RTI Connext.

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

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

See also

Request-Reply Examples (p. 146).

6.43 Requester

com.rti.connext.requestreply.Requester<TReq,TRep> and associated elements

Classes

- class **RequesterParams**

  Contains the parameters for creating a com.rti.connext.requestreply.Requester<TReq,TRep>
- class **Requester< TReq, TRep >**

  Allows sending requests and receiving replies.

6.43.1 Detailed Description

com.rti.connext.requestreply.Requester<TReq,TRep> and associated elements

6.44 Replier

com.rti.connext.requestreply.Replier<TReq,TRep>, com.rti.connext.requestreply.SimpleReplier<TReq,TRep> and associated elements

Classes

- interface **ReplierListener< TReq, TRep >**

  Called when a com.rti.connext.requestreply.Replier<TReq,TRep> has new available requests.
- class **ReplierParams< TReq, TRep >**

  Contains the parameters for creating a com.rti.connext.requestreply.Replier<TReq,TRep>.
- interface **SimpleReplierListener< TReq, TRep >**

  *The listener called by a **SimpleReplier** (p. 1692).*
- class **SimpleReplierParams< TReq, TRep >**

  Contains the parameters for creating a com.rti.connext.requestreply.SimpleReplier<TReq,TRep>
- class **Replier< TReq, TRep >**

  Allows receiving requests and sending replies.
- class **SimpleReplier< TReq, TRep >**

  A callback-based replier.

6.44.1 Detailed Description

com.rti.connext.requestreply.Replier<TReq,TRep>, com.rti.connext.requestreply.SimpleReplier<TReq,TRep> and associated elements

6.45 Infrastructure

Infrastructure types for RTI Connext Messaging.

Modules

- **RTI Connext Exceptions**

  RTI Connext return-code exceptions.

Classes

- interface **Sample**< **T** >

  A data sample and related info received from the middleware.
- interface **Sample**< **T** >**.Iterator**< **T** >

  Provides access to a collection of middleware-loaned samples.
- interface **SampleData**< **T** >

  Sample (p. *1627) base type that contains data and has an identity.*
- interface **WriteSample**< **T** >

  A sample for writing data.

6.45.1 Detailed Description

Infrastructure types for RTI Connext Messaging.

This section describes the sample types used for reading and writing. These types encapsulate data and related information:

- com.rti.connext.infrastructure.Sample<T> (used for reading)
- com.rti.connext.infrastructure.WriteSample<T> (used for writing)

There are different operations that provide loaned samples from the middleware. They return a com.rti.connext.↩ infrastructure.Sample<T>.Iterator<T>.

6.46 Utilities

Utilities for the RTI Connext Messaging module.

Utilities for the RTI Connext Messaging module.

Connext Messaging Utilities

6.47 Durability and Persistence

APIs related to RTI Connext Durability and Persistence.

APIs related to RTI Connext Durability and Persistence.

RTI Connext offers the following mechanisms for achieving durability and persistence:

- **Durable Writer History** (p. 114)
- **Durable Reader State** (p. 114)
- **Data Durability** (p. 115)

To use the first two features, you need a relational database, which is not included with RTI Connext. Supported databases are listed in the `Release Notes`.

The third feature, provided by RTI Persistence Service, can use the filesystem or a relational database to persist information.

These three features can be used separately or in combination.

6.47.1 Durable Writer History

This feature allows a **com.rti.dds.publication.DataWriter** (p. 553) to locally persist its local history cache so that it can survive shutdowns, crashes and restarts. When an application restarts, each **com.rti.dds.publication.DataWriter** (p. 553) that has been configured to have durable writer history automatically loads all the data in its history cache from disk and can carry on sending data as if it had never stopped executing. To the rest of the system, it will appear as if the **com.rti.dds.publication.DataWriter** (p. 553) had been temporarily disconnected from the network and then reappeared.

See also

Configuring Durable Writer History (p. 116)

6.47.2 Durable Reader State

This feature allows a **com.rti.dds.subscription.DataReader** (p. 450) to locally persists its state and remember the sequence numbers it has already received. When an application restarts, each **com.rti.dds.subscription.DataReader** (p. 450) that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the **com.rti.dds.↩ subscription.DataReader** (p. 450) before the restart will will not be provided to the application again.

See also

Configuring Durable Reader State (p. 118)

6.47.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 230) QoS lets an application configure a **com.rti.dds.publication.DataWriter** (p. 553) such that the information written by the **com.rti.dds.publication.DataWriter** (p. 553) survives beyond the lifetime of the **com.rti.dds.publication.DataWriter** (p. 553). In this manner, a late-joining **com.rti.dds.subscription.DataReader** (p. 450) can subscribe and receive the information even after the **com.rti.dds.publication.DataWriter** (p. 553) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

6.47.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **com.rti.dds.publication.DataWriter** (p. 553) is identified by a pair (virtual GUID, sequence number).

- The virtual GUID (Global Unique Identifier) is a 16-byte character identifier associated with a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450); it is used to uniquely identify this entity in the global data space.
- The sequence number is a 64-bit identifier that identifies changes published by a specific **com.rti.dds.publication.DataWriter** (p. 553).

Several **com.rti.dds.publication.DataWriter** (p. 553) entities can be configured with the same virtual GUID. If each of these **com.rti.dds.publication.DataWriter** (p. 553) entities publishes a sample with sequence number '0', the sample will only be received once by the **com.rti.dds.subscription.DataReader** (p. 450) entities subscribing to the content published by the **com.rti.dds.publication.DataWriter** (p. 553) entities.

RTI Connext also uses the virtual GUID (Global Unique Identifier) to associate a persisted state (state in permanent storage) to the corresponding DDS entity.

For example, the history of a **com.rti.dds.publication.DataWriter** (p. 553) will be persisted in a database table with a name generated from the virtual GUID of the **com.rti.dds.publication.DataWriter** (p. 553). If the **com.rti.dds.publication.DataWriter** (p. 553) is restarted, it must have associated the same virtual GUID to restore its previous history.

Likewise, the state of a **com.rti.dds.subscription.DataReader** (p. 450) will be persisted in a database table whose name is generated from the **com.rti.dds.subscription.DataReader** (p. 450) virtual GUID

A **com.rti.dds.publication.DataWriter** (p. 553)'s virtual GUID can be configured using **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid** (p. 598). Similarly, a **com.rti.dds.subscription.DataReader** (p. 450)'s virtual GUID can be configured using **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 502).

The **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) structures include the virtual GUID associated with the discovered publication or subscription.

Refer to the `User's Manual` for additional use cases.

See also

com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid (p. 598) **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 502).

6.47.5 Configuring Durable Writer History

To configure a **com.rti.dds.publication.DataWriter** (p. 553) to have durable writer history, use the **PROPERTY** (p. 248) QoS policy associated with the **com.rti.dds.publication.DataWriter** (p. 553) or the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Properties defined for the **com.rti.dds.domain.DomainParticipant** (p. 670) will be applied to all the **com.rti.dds.↩ publication.DataWriter** (p. 553) objects belonging to the **com.rti.dds.domain.DomainParticipant** (p. 670), unless the property is overwritten by the **com.rti.dds.publication.DataWriter** (p. 553).

See also

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1438)

The following table lists the supported durable writer history properties.

Table 6.9 ***Durable Writer History Properties***

Property	Description
dds.data_writer.history.plugin_name	Must be set to "dds.data_writer.history.odbc_plugin.↩ builtin" to enable durable writer history in the DataWriter. This property is required.
dds.data_writer.history.odbc_plugin.builtin.dsn	The ODBC DSN (Data Source Name) associated with the database where the writer history must be persisted. This property is required.
dds.data_writer.history.odbc_plugin.builtin.driver	This property tells RTI Connext which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library↩ : UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems.
dds.data_writer.history.odbc_plugin.builtin.username	Configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.password	Configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_writer.history.odbc_plugin.builtin.shared	If set to 1, RTI Connext creates a single connection per DSN that will be shared across DataWriters within the same Publisher. If set to 0 (the default), a **com.rti.↩ dds.publication.DataWriter** (p. 553) will create its own database connection. Default: 0

Property	Description
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_max_size	These properties configure the resource limits associated with the ODBC writer history caches. To minimize the number of accesses to the database, RTI Connext uses two caches, one for samples and one for instances. The initial and maximum sizes of these caches are configured using these properties. The resource limits initial_instances, max_instances, initial↩ _samples, max_samples and max_samples_per_↩ instance in the **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy** (p. 1590) are used to configure the maximum number of samples and instances that can be stored in the relational database. Default↩ : **com.rti.dds.infrastructure.ResourceLimitsQos↩ Policy.max_instances** (p. 1592)
dds.data_writer.history.odbc_plugin.builtin.instance_↩ cache_init_size	See description above. Default: **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.initial_↩ instances** (p. 1593)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_max_size	See description above. Default: 32 (the minimum)
dds.data_writer.history.odbc_plugin.builtin.sample_↩ cache_init_size	See description above. Default: 32
dds.data_writer.history.odbc_plugin.builtin.restore	This property indicates whether or not the persisted writer history must be restored once the **com.rti.↩ dds.publication.DataWriter** (p. 553) is restarted. If the value is 0, the content of the database associated with the **com.rti.dds.publication.DataWriter** (p. 553) being restarted will be deleted. If the value is 1, the **com.rti.↩ dds.publication.DataWriter** (p. 553) will restore its previous state from the database content. Default: 1
dds.data_writer.history.odbc_plugin.builtin.in_memory↩ _state	This property determines how much state will be kept in memory by the ODBC writer history in order to avoid accessing the database. When in_memory_state is equal to 1, instance_↩ cache_max_size is always equal to **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.max_↩ instances** (p. 1592) (it cannot be changed). In addition, the ODBC writer history will keep in memory a fixed state overhead of 24 bytes per sample. In this operating mode, the ODBC writer history provides the best performance. However, the restore operation will be slower and the maximum number of samples that the writer history can manage will be limited by the available physical memory. If in_memory_state is equal to 0, all the state will be kept in the underlying database. In this operating mode, the maximum number of samples in the writer history will not be limited by the physical memory available unless the underlying database is an in-memory database (Times↩ Ten). Default: 1

6.47.6 Configuring Durable Reader State

To configure a **com.rti.dds.subscription.DataReader** (p. 450) with durable reader state, use the **PROPERTY** (p. 248) QoS policy associated with the **com.rti.dds.subscription.DataReader** (p. 450) or **com.rti.dds.domain.DomainParticipant** (p. 670).

A property defined in the **com.rti.dds.domain.DomainParticipant** (p. 670) will be applicable to all the **com.rti.dds.subscription.DataReader** (p. 450) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 670) unless it is overwritten by the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1438)

The following table lists the supported durable reader state properties.

Table 6.10 ***Durable Reader State Properties***

Property	Description
dds.data_reader.state.odbc.dsn	The ODBC DSN (Data Source Name) associated with the database where the **com.rti.dds.subscription.DataReader** (p. 450) state must be persisted. This property is required.
dds.data_reader.state.filter_redundant_samples	To enable durable reader state, this property must be set to 1. Otherwise, the reader state will not be kept and/or persisted. When the reader state is not maintained, RTI Connext does not filter duplicate samples that may be coming from the same virtual writer. By default, this property is set to 1.
dds.data_reader.state.odbc.driver	This property is used to indicate which ODBC driver to load. If the property is not specified, RTI Connext will try to use the standard ODBC driver manager library: UnixOdbc (odbc32.dll) on Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems).
dds.data_reader.state.odbc.username	This property configures the username used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.odbc.password	This property configures the password used to connect to the database. This property is not used if it is unspecified. There is no default value.
dds.data_reader.state.restore	This property indicates if the persisted **com.rti.dds.subscription.DataReader** (p. 450) state must be restored or not once the **com.rti.dds.subscription.DataReader** (p. 450) is restarted. If this property is 0, the previous state will be deleted from the database. If it is 1, the **com.rti.dds.subscription.DataReader** (p. 450) will restore its previous state from the database content. Default: 1

Property	Description
dds.data_reader.state.checkpoint_frequency	This property controls how often the reader state is stored in the database. A value of N means to store the state once every N samples. A high value will provide better performance. However, if the reader is restarted it may receive some duplicate samples. These samples will be filtered by the middleware and they will not be propagated to the application. Default: 1
dds.data_reader.state.persistence_service.request_↩ depth	This property indicates the number of most recent historical samples that the persisted **com.rti.dds.**↩ **subscription.DataReader** (p. 450) wants to receive when it starts up. Default: 0

6.47.7 Configuring Data Durability

RTI Connext implements com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_↩ DURABILITY_QOS and com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.PERSISTENT_↩ DURABILITY_QOS durability using RTI Persistence Service, available for purchase as a separate RTI product.

For more information on RTI Persistence Service, refer to the `User's Manual`, or the RTI Persistence Service API Reference HTML documentation.

See also

DURABILITY (p. 230)

6.48 System Properties

System Properties.

System Properties.

RTI Connext uses the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) of a DomainParticipant to maintain a set of properties that provide system information such as hostname.

Unless the default **com.rti.dds.domain.DomainParticipantQos** (p. 795) value is overwritten, the system properties are automatically set in the **com.rti.dds.domain.DomainParticipantQos** (p. 795) obtained by calling the method **com.rti.**↩ **dds.domain.DomainParticipantFactory.get_default_participant_qos** (p. 767) or using the constant **com.rti.dds.**↩ **domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42).

System properties are also automatically set in the **com.rti.dds.domain.DomainParticipantQos** (p. 795) loaded from an XML QoS profile unless you disable property inheritance using the attribute **inherit** in the XML tag <**property**>.

By default, the system properties are propagated to other DomainParticipants in the system and can be accessed through builtin.ParticipantBuiltinTopicData.property.

You can disable the propagation of individual properties by setting the flag **com.rti.dds.infrastructure.Property_t.**↩ **propagate** (p. 1398) to com.rti.dds.infrastructure.false or by removing the property using the method **com.rti.dds.**↩ **infrastructure.PropertyQosPolicyHelper.remove_property** (p. 1443).

The number of system properties set on the **com.rti.dds.domain.DomainParticipantQos** (p. 795) is platform specific.

6.48.1 System Properties List

The following table lists the supported system properties. For more information, see `System Properties`, in the Core Libraries User's Manual.

Property Name	Description
dds.sys_info.hostname	Name of the host where the process is running
dds.sys_info.process_id	Process ID
dds.sys_info.username	Name of the user running the process
dds.sys_info.execution_timestamp	Time when the execution started
dds.sys_info.creation_timestamp	Time when the executable was created
dds.sys_info.executable_filepath	Name and full path of the executable
dds.sys_info.target	Architecture for which the library was compiled

6.48.2 System Resource Consideration

System properties are affected by the resource limits **com.rti.dds.infrastructure.DomainParticipantResource↩ LimitsQosPolicy.participant_property_list_max_length** (p. 819) and **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.participant_property_string_max_length** (p. 819).

6.49 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

APIs related to XML QoS Profiles.

6.49.1 Loading QoS Profiles from XML Resources

A 'QoS profile' is a group of QoS settings, specified in XML format. By using QoS profiles, you can change QoS settings without recompiling the application.

The Qos profiles are loaded the first time any of the following operations are called:

- **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 765)
- **com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile** (p. 782)
- **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 768)
- **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos** (p. 767)
- **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)

- **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)
- **com.rti.dds.domain.DomainParticipantFactory.get_participant_qos_from_profile** (p. 776)
- **com.rti.dds.domain.DomainParticipantFactory.get_topic_qos_from_profile** (p. 780)
- **com.rti.dds.domain.DomainParticipantFactory.get_topic_qos_from_profile_w_topic_name** (p. 781)
- **com.rti.dds.domain.DomainParticipantFactory.get_publisher_qos_from_profile** (p. 776)
- **com.rti.dds.domain.DomainParticipantFactory.get_subscriber_qos_from_profile** (p. 777)
- **com.rti.dds.domain.DomainParticipantFactory.get_datawriter_qos_from_profile** (p. 778)
- **com.rti.dds.domain.DomainParticipantFactory.get_datawriter_qos_from_profile_w_topic_name** (p. 778)
- **com.rti.dds.domain.DomainParticipantFactory.get_datareader_qos_from_profile** (p. 779)
- **com.rti.dds.domain.DomainParticipantFactory.get_datareader_qos_from_profile_w_topic_name** (p. 779)
- **com.rti.dds.domain.DomainParticipantFactory.get_qos_profile_libraries** (p. 781)
- **com.rti.dds.domain.DomainParticipantFactory.get_qos_profiles** (p. 782)
- **com.rti.dds.domain.DomainParticipantFactory.load_profiles** (p. 771)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **com.rti.dds.domain.DomainParticipantFactory.set_qos** (p. 770)
- **com.rti.dds.domain.DomainParticipantFactory.reload_profiles** (p. 771)

The **com.rti.dds.domain.DomainParticipantFactory.unload_profiles** (p. 772) operation will free the resources associated with the XML QoS profiles.

There are five ways to configure the XML resources (listed by load order):

- The file NDDS_QOS_PROFILES.xml in $NDDSHOME/resource/xml is loaded if it exists and **com.rti.dds.↩ infrastructure.ProfileQosPolicy.ignore_resource_profile** (p. 1395) in **com.rti.dds.infrastructure.Profile↩ QosPolicy** (p. 1393) is set to com.rti.dds.infrastructure.false (first to be loaded). An example file, `NDDS_QOS↩ _PROFILES.example.xml`, is available for reference.

- The URL groups separated by semicolons referenced by the enviroment variable NDDS_QOS_PROFILES are loaded if they exist and **com.rti.dds.infrastructure.ProfileQosPolicy.ignore_environment_profile** (p. 1395) in **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1393) is set to com.rti.dds.infrastructure.false.

- The file USER_QOS_PROFILES.xml in the working directory will be loaded if it exists and **com.rti.dds.↩ infrastructure.ProfileQosPolicy.ignore_user_profile** (p. 1395) in **com.rti.dds.infrastructure.ProfileQos↩ Policy** (p. 1393) is set to com.rti.dds.infrastructure.false.

- The URL groups referenced by **com.rti.dds.infrastructure.ProfileQosPolicy.url_profile** (p. 1394) in **com.rti.↩ dds.infrastructure.ProfileQosPolicy** (p. 1393) will be loaded if specified.

- The sequence of XML strings referenced by **com.rti.dds.infrastructure.ProfileQosPolicy.string_profile** (p. 1394) will be loaded if specified (last to be loaded).

The above methods can be combined together.

6.49.2 URL

The location of the XML resources (only files and strings are supported) is specified using a URL (Uniform Resource Locator) format. For example:

File Specification: `file:///usr/local/default_dds.xml`

String Specification: str://"<dds><qos_library> . . . lt;/qos_library>&lt/dds>"

If the URL schema name is omitted, RTI Connext will assume a file name. For example:

File Specification: /usr/local/default_dds.xml

6.49.2.1 URL groups

To provide redundancy and fault tolerance, you can specify multiple locations for a single XML document via URL groups. The syntax of a URL group is as follows:

[URL1 | URL2 | URL2 | . . .| URLn]

For example:

[`file:///usr/local/default_dds.xml` | `file:///usr/local/alternative_default_dds.xml`]

Only one of the elements in the group will be loaded by RTI Connext, starting from the left.

Brackets are not required for groups with a single URL.

6.49.2.2 NDDS_QOS_PROFILES environment variable

The environment variable NDDS_QOS_PROFILES contains a list of URL groups separated by ';'

The URL groups referenced by the environment variable are loaded if they exist and **com.rti.dds.infrastructure.ProfileQosPolicy.ignore_environment_profile** (p. 1395) is set to com.rti.dds.infrastructure.false

6.49.2.3 Built-In QoS Profiles

There are also a number of built-in QoS profiles that can be used without having to load any configurations from outside XML resources. For more information on these built-in profiles see **Built-in Qos Profiles** (p. 171).

For more information on XML Configuration, refer to the `User's Manual`.

6.50 Publication Example

A data publication example.

A data publication example.

6.50.1 A typical publication example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 125)
- **Set up participant** (p. 125)
- **Set up publisher** (p. 129)
- **Register user data type(s)** (p. 126)
- **Set up topic(s)** (p. 126)
- **Set up data writer(s)** (p. 130)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 136)

Send data

- **Send data** (p. 130)

Tear down

- **Tear down data writer(s)** (p. 131)
- **Tear down topic(s)** (p. 126)
- **Tear down publisher** (p. 129)
- **Tear down participant** (p. 126)

6.51 Subscription Example

A data subscription example.

A data subscription example.

6.51.1 A typical subscription example

Prep

- Create user data types using rtiddsgen. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 125)
- **Set up participant** (p. 125)
- **Set up subscriber** (p. 131)
- **Register user data type(s)** (p. 126)
- **Set up topic(s)** (p. 126)
- **Set up data reader(s)** (p. 133)

- **Set up data reader** (p. 133) **OR Set up subscriber** (p. 131) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 136)

Receive data

- Access received data either **via a reader** (p. 134) **OR via a subscriber** (p. 132) (possibly in a **ordered or coherent** (p. 132) manner)

Tear down

- **Tear down data reader(s)** (p. 135)
- **Tear down topic(s)** (p. 126)
- **Tear down subscriber** (p. 132)
- **Tear down participant** (p. 126)

6.52 Participant Use Cases

Working with domain participants.

Working with domain participants.

6.52.1 Turning off auto-enable of newly created participant(s)

- **Get the factory** (p. 125)

- Change the value of the **ENTITY_FACTORY** (p. 234) for the **com.rti.dds.domain.DomainParticipantFactory** (p. 761)

```
DomainParticipantFactoryQos factory_qos = new DomainParticipantFactoryQos();

try {
    factory.get_qos(factory_qos);

    /* Change the QosPolicy to create disabled participants */
    factory_qos.entity_factory.autoenable_created_entities = false;

    factory.set_qos(factory_qos);

} catch (RETCODE_ERROR err) {
    System.out.println(
            "***Error: changing domain participant factory qos\n");
}
```

6.52.2 Getting the factory

- Get the DDSDomainParticipantFactory instance:

```
DomainParticipantFactory factory = null;
factory = DomainParticipantFactory.get_instance();
```

6.52.3 Setting up a participant

- **Get the factory** (p. 125)

- Create DDSDomainParticipant:

```
int domain_id = 10;
DomainParticipantQos participant_qos = new DomainParticipantQos();
// initialize participant_qos with default values
factory.get_default_participant_qos(participant_qos);
/* Set the peer hosts. These list all the computers the application
 * may communicate with along with the maximum maximum participant
 * index of the participants that can concurrently run on that
 * computer. This list only needs to be a superset of the actual list
 * of computers and participants that will be running at any time.
 */
/* To run this example across multiple nodes, modify the following
 * IP addresses to match your network configuration.
 */
 final String[] NDDS_DISCOVERY_INITIAL_PEERS = {
                "1@udpv4://10.10.1.192",
                "1@udpv4://10.10.1.190",
                "1@udpv4://10.10.1.152"
  };
  participant_qos.discovery.initial_peers.
                                ensureCapacity(NDDS_DISCOVERY_INITIAL_PEERS.length);
  for (int i = 0; i < NDDS_DISCOVERY_INITIAL_PEERS.length; ++i) {
          participant_qos.discovery.initial_peers.add(
                NDDS_DISCOVERY_INITIAL_PEERS[i]);
  }

  // Initialize listener if desired
  DomainParticipantListener participant_listener = null;
  // Create the participant
  DomainParticipant participant = null;
  try {
          participant = factory.create_participant(
                        domain_id, participant_qos,
                                            participant_listener, StatusKind.STATUS_MASK_NONE);
  } catch (RETCODE_ERROR err) {
    // participant couldn't be created
  }
```

6.52.4 Tearing down a participant

- **Get the factory** (p. 125)
- Delete DDSDomainParticipant:

```
try {
    factory.delete_participant(participant);
} catch (RETCODE_ERROR err) {
    // unable to delete
}
```

6.53 Topic Use Cases

Working with topics.

Working with topics.

6.53.1 Registering a user data type

- **Set up participant** (p. 125)
- Register user data type of type `Foo` under the name "My_Type"

```
String type_name = "My_Type";
FooTypeSupport.register_type(participant, type_name);
```

6.53.2 Setting up a topic

- **Set up participant** (p. 125)
- **Ensure user data type is registered** (p. 126)
- Create a **com.rti.dds.topic.Topic** (p. 1807) under the name "my_topic"

```
String topic_name = "my_topic";
String type_type = "My_Type";   // user data type
TopicQos topic_qos = new TopicQos();
// MyTopicListener is user defined and
// implements TopicListener
TopicListener topic_listener = new MyTopicListener(); // or = null
participant.get_default_topic_qos(topic_qos);
Topic topic = null;
try {
    topic = participant.create_topic(topic_name, type_name,
                                     topic_qos, topic_listener,
                                StatusKind.STATUS_MASK_ALL);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.53.3 Tearing down a topic

- Delete Topic:

```
try {
    participant.delete_topic(topic);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.54 FlowController Use Cases

Working with flow controllers.

Working with flow controllers.

6.54.1 Creating a flow controller

- **Set up participant** (p. 125)
- Create a flow controller

```
FlowController controller = null;
FlowControllerProperty_t property = new FlowControllerProperty_t();
retcode = participant.get_default_flowcontroller_property(property);
// optionally modify flow controller property values
try {
    controller = participant.create_flowcontroller(
        "my flow controller name", property);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.54.2 Flow controlling a data writer

- **Set up participant** (p. 125)
- **Create flow controller** (p. 127)
- Create an asynchronous data writer, FooDataWriter, of user data type `Foo`:

```
DataWriterQos writer_qos = new DataWriterQos();
// MyWriterListener is user defined and
// implements DataWriterListener
MyWriterListener writer_listener = new MyWriterListener(); // or = null
publisher.get_default_datawriter_qos(writer_qos);
/* Change the writer QoS to publish asnychronously */
writer_qos.publish_mode.kind = PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS;
/* Setup to use the previously created flow controller */
writer_qos.publish_mode.flow_controller_name = "my flow controller name";
/* Samples queued for asynchronous write are subject to the History Qos policy */
writer_qos.history.kind = HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS;
FooDataWriter writer = null;
try {
    writer = (FooDataWriter) publisher.create_datawriter(topic, writer_qos,
                                                         writer_listener,
                                                         StatusKind.STATUS_MASK_ALL);
    /* Send data asynchronously... */

    /* Wait for asynchronous send completes, if desired */
    writer.wait_for_asynchronous_publishing(timout);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.54.3 Using the built-in flow controllers

RTI Connext provides several built-in flow controllers.

The **com.rti.dds.publication.FlowController.DEFAULT_FLOW_CONTROLLER_NAME** (p. 75) built-in flow controller provides the basic asynchronous writer behavior. When calling **com.rti.ndds.example.FooDataWriter.write** (p. 1105),

the call signals the **com.rti.dds.publication.Publisher** (p. 1466) asynchronous publishing thread (**com.rti.dds.↩ publication.PublisherQos.asynchronous_publisher** (p. 1494)) to send the actual data. As with any com.rti.dds.↩ infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 553), the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) call returns immediately afterwards. The data is sent immediately in the context of the **com.rti.dds.publication.Publisher** (p. 1466) asynchronous publishing thread.

When using the **com.rti.dds.publication.FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 75) flow controller, data is also sent in the context of the **com.rti.dds.publication.Publisher** (p. 1466) asynchronous publishing thread, but at a regular fixed interval. The thread accumulates samples from different **com.rti.dds.publication.Data↩ Writer** (p. 553) instances and generates data on the wire only once per **com.rti.dds.publication.FlowController↩ TokenBucketProperty_t.period** (p. 1065).

In contrast, the **com.rti.dds.publication.FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 76) flow controller permits flow only when **com.rti.dds.publication.FlowController.trigger_flow** (p. 1058) is called. The data is still sent in the context of the **com.rti.dds.publication.Publisher** (p. 1466) asynchronous publishing thread. The thread accumulates samples from different **com.rti.dds.publication.DataWriter** (p. 553) instances (across any **com.rti.dds.↩ publication.Publisher** (p. 1466)) and sends all data since the previous trigger.

The properties of the built-in **com.rti.dds.publication.FlowController** (p. 1055) instances can be adjusted.

- **Set up participant** (p. 125)
- Lookup built-in flow controller

```
FlowController controller = null;
try {
    controller = participant.lookup_flowcontroller(
        FlowController.DEFAULT_FLOW_CONTROLLER_NAME);
} catch (RETCODE_ERROR err) {
    // This should never happen, built-in flow controllers are always created
    // handle exception
}
```

- Change property of built-in flow controller, if desired

```
FlowControllerProperty_t property = new FlowControllerProperty_t();
/* Get the property of the looked-up default flow controller */
controller.get_property(property);
/* Change the property value as desired */
property.token_bucket.period.sec = 2;
property.token_bucket.period.nanosec = 0;
/* Update the flow controller property */
controller.set_property(property);
```

- **Create a data writer using the correct flow controller name** (p. 127)

6.54.4 Shaping the network traffic for a particular transport

- **Set up participant** (p. 125)
- **Create the transports** (p. 138)
- **Create a separate flow controller for each transport** (p. 127)
- Configure **com.rti.dds.publication.DataWriter** (p. 553) instances to only use a single transport
- **Associate all data writers using the same transport to the corresponding flow controller** (p. 127)
- For each transport, the corresponding flow controller limits the network traffic based on the token bucket properties

6.54.5 Coalescing multiple samples in a single network packet

- **Set up participant** (p. 125)
- **Create a flow controller with a desired token bucket period** (p. 127)
- **Associate the data writer with the flow controller** (p. 127)
- Multiple samples written within the specified period will be coalesced into a single network packet (provided that `tokens_added_per_period` and `bytes_per_token` permit).

6.55 Publisher Use Cases

Working with publishers.

Working with publishers.

6.55.1 Setting up a publisher

- **Set up participant** (p. 125)
- Create a DDSPublisher

```
PublisherQos publisher_qos = new PublisherQos();
// MyPublisherListener is user defined and
// extends DDSPublisherListener
PublisherListener publisher_listener
    = new MyPublisherListener(); // or = null
participant.get_default_publisher_qos(publisher_qos);
Publisher publisher = null;
try {
    publisher = participant.create_publisher(publisher_qos,
                                             publisher_listener,
                                        StatusKind.STATUS_MASK_ALL);
} catch (RETCODE_ERROR err) {
    // respond to exception
}
```

6.55.2 Tearing down a publisher

- Delete DDSPublisher:

```
try {
    participant.delete_publisher(publisher);
} catch (RETCODE_ERROR err) {
    // respond to exception
}
```

6.56 DataWriter Use Cases

Working with data writers.

Working with data writers.

6.56.1 Setting up a data writer

- **Set up publisher** (p. 129)
- **Set up a topic** (p. 126)
- Create a data writer, FooDataWriter, of user data type `Foo`:

```
DataWriterQos writer_qos = new DataWriterQos();
// MyWriterListener is user defined and
// implements DataWriterListener
MyWriterListener writer_listener = new MyWriterListener(); // or = null
publisher.get_default_datawriter_qos(writer_qos);
FooDataWriter writer = null;
try {
    writer = (FooDataWriter) publisher.create_datawriter(topic, writer_qos,
                                                         writer_listener,
                                                    StatusKind.STATUS_MASK_ALL);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.56.2 Managing instances

- Getting an instance "key" value of user data type `Foo`

```
Foo data = ...;     // user data
try {
    writer.get_key_value(data, instance_handle);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

- Registering an instance of type `Foo`

```
InstanceHandle_t instance_handle = InstanceHandle_t.HANDLE_NIL;
instance_handle = writer->register_instance(data);
```

- Unregistering an instance of type `Foo`

```
try {
    writer.unregister_instance(data, instance_handle);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

- Disposing of an instance of type `Foo`

```
try {
    writer.dispose(data, instance_handle);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

6.56.3 Sending data

- **Set up data writer** (p. 130)
- **Register instance** (p. 130)
- Write instance of type `Foo`

```
Foo data = new Foo();     // user data
InstanceHandle_t instance_handle
    = InstanceHandle_t.HANDLE_NIL; // or a valid registered handle
try {
    writer.write(data, instance_handle);
} catch (RETCODE_ERR err) {
    // ... check for cause of failure
}
```

6.56.4 Tearing down a data writer

- Delete DataWriter:

```
try {
    publisher.delete_datawriter(writer);
} catch (RETCODE_ERR err) {
    // ... check for cause of failure
}
```

6.57 Subscriber Use Cases

Working with subscribers.

Working with subscribers.

6.57.1 Setting up a subscriber

- **Set up participant** (p. 125)
- Create a Subscriber

```
SubscriberQos subscriber_qos = new SubscriberQos();
// MySubscriberListener is user defined and
// implements SubscriberListener
SubscriberListener subscriber_listener
    = new MySubscriberListener(); // or = null
participant.get_default_subscriber_qos(subscriber_qos);
Subscriber subscriber = null;
try {
    subscriber = participant.create_subscriber(subscriber_qos,
                                               subscriber_listener,
                                          StatusKind.STATUS_MASK_ALL);
} catch (RETCODE_ERROR err) {
    // respond to exception
}
```

6.57.2 Set up subscriber to access received data

- **Set up subscriber** (p. 131)
- Set up to handle the DATA_ON_READERS_STATUS status, in one or both of the following two ways.
- **Enable DATA_ON_READERS_STATUS for the SubscriberListener associated with the subscriber** (p. 136)
  - The processing to handle the status change is done in the **com.rti.dds.subscription.Subscriber↩ Listener.on_data_on_readers** (p. 1756) method of the attached listener.
  - Typical processing will **access the received data** (p. 132), either in arbitrary order or in a **coherent and ordered manner** (p. 132).
- **Enable DATA_ON_READERS_STATUS for the StatusCondition associated with the subscriber** (p. 137)
  - The processing to handle the status change is done **when the subscriber's attached status condition is triggered** (p. 138) and the DATA_ON_READERS_STATUS status on the subscriber is changed.
  - Typical processing will **access the received data** (p. 132), either in an arbitrary order or in a **coherent and ordered manner** (p. 132).

6.57.3 Access received data via a subscriber

- **Ensure subscriber is set up to access received data** (p. 131)
- Get the list of readers that have data samples available:
```
DataReaderSeq reader_seq = new DataReaderSeq(); // list of readers
int max_samples = DataReader.LENGTH_UNLIMITED;
int  sample_state_mask = SampleStateKind.NOT_READ_SAMPLE_STATE;
int  view_state_mask = ViewStateKind.ANY_VIEW_STATE;
int  instance_state_mask = InstanceStateKind.ANY_INSTANCE_STATE;

try {
    subscriber.get_datareaders(reader_seq,
                               sample_state_mask,
                               view_state_mask,
                               instance_state_mask);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```
- Upon successfully getting the list of readers with data, process the data readers to either:
  - **Read the data in each reader** (p. 134), **OR**
  - **Take the data in each reader** (p. 134)

  If the intent is to access the data **coherently or in order** (p. 132), the list of data readers *must* be processed in the order returned:
```
for (int i = 0; i < reader_seq.size(); ++i) {
    FooDataReader reader = (FooDataReader) reader_seq.get(i);
    // Take the data from reader,
    //             OR
    // Read the data from reader
}
```
- **Alternatively**, call **com.rti.dds.subscription.Subscriber.notify_datareaders** (p. 1742) to invoke the Data↩ReaderListener for each of the data readers.
```
subscriber.notify_datareaders();
```

6.57.4 Access received data coherently and/or in order

To access the received data coherently and/or in an ordered manner, according to the settings of the **com.rti.dds.↩infrastructure.PresentationQosPolicy** (p. 1379) attached to a **com.rti.dds.subscription.Subscriber** (p. 1730):

- **Ensure subscriber is set up to access received data** (p. 131)
- Indicate that data will be accessed via the subscriber:
```
subscriber.begin_access();
```
- **Access received data via the subscriber, making sure that the data readers are processed in the order returned.** (p. 132)
- Indicate that the data access via the subscriber is done:
```
subscriber.end_access();
```

6.57.5 Tearing down a subscriber

- Delete Subscriber:
```
try {
    participant.delete_subscriber(subscriber);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.58 DataReader Use Cases

Working with data readers.

Working with data readers.

6.58.1 Setting up a data reader

- **Set up subscriber** (p. 131)
- **Set up a topic** (p. 126)
- Create a data reader, FooDataReader, of user data type `Foo`:

```
DataReaderQos reader_qos = new DataReaderQos();
// MyReaderListener is user defined and
// implements DataReaderListener
DataReaderListener reader_listener
    = new MyReaderListener(); // or = null
subscriber.get_default_datareader_qos(reader_qos);
FooDataReader reader = null;
try {
    reader = (FooDataReader) subscriber.create_datareader(topic,
                                                          reader_qos,
                                                          reader_listener,
                                                      StatusKind.STATUS_MASK_ALL);
} catch (RETCODE_ERROR err) {
    // respond to exception
}
```

6.58.2 Managing instances

- Given a data reader

```
FooDataReader reader = ...;
```

- Getting an instance "key" value of user data type `Foo`

```
Foo data = new Foo();     // user data of type Foo
// ...
try {
    reader.get_key_value(data, instance_handle);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

6.58.3 Set up reader to access received data

- **Set up data reader** (p. 133)
- Set up to handle the DATA_AVAILABLE_STATUS status, in one or both of the following two ways.
- **Enable DATA_AVAILABLE_STATUS for the DataReaderListener associated with the data reader** (p. 136)
  - The processing to handle the status change is done in the **com.rti.dds.subscription.DataReader↩ Listener.on_data_available** (p. 500) method of the attached listener.
  - Typical processing will **access the received data** (p. 134).
- **Enable DATA_AVAILABLE_STATUS for the StatusCondition associated with the data reader** (p. 137)
  - The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 138) and the DATA_AVAILABLE_STATUS status on the data reader is changed.
  - Typical processing will **access the received data** (p. 134).

6.58.4 Access received data via a reader

- **Ensure reader is set up to access received data** (p. 133)
- Access the received data, by either:
  - **Taking the received data in the reader** (p. 134), **OR**
  - **Reading the received data in the reader** (p. 134)

6.58.5 Taking data

- **Ensure reader is set up to access received data** (p. 133)
- Take samples of user data type `Foo`. The samples are removed from the Service. The caller is responsible for deallocating the buffers.

```
FooSeq         data_seq = new FooSeq();        // holder for sequence of user data type Foo
SampleInfoSeq info_seq = new SampleInfoSeq(); // holder for sequence of DDS_SampleInfo
int            max_samples;
int            sample_state_mask = SampleStateMask.ANY_SAMPLE_STATE;
int            view_state_mask = ViewStateMask.ANY_VIEW_STATE;
int            instance_state_mask = InstanceStateMask.ANY_INSTANCE_STATE;
try {
    reader.take(data_seq, info_seq,
                max_samples,
                sample_state_mask,
                view_state_mask,
                instance_state_mask);
} catch (RETCODE_ERROR) {
    // ... check for cause of failure
}
```

- Use the received data

```
// Use the received data samples 'data_seq' and associated
// information 'info_seq'
for (int i = 0; i < data_seq.size(); ++i) {
    // use... data_seq.get(i) ...
    // use... info_seq.get(i) ...
}
```

- Return the data samples and the information buffers back to the middleware. *IMPORTANT*: Once this call returns, you must not retain any pointers to any part of any sample or sample info object.

```
reader.return_loan(data_seq, info_seq);
```

6.58.6 Reading data

- **Ensure reader is set up to access received data** (p. 133)
- Read samples of user data type `Foo`. The samples are not removed from the Service. It remains responsible for deallocating the buffers.

```
FooSeq         data_seq = new FooSeq();        // holder for sequence of user data type Foo
SampleInfoSeq info_seq = new SampleInfoSeq)_; // holder for sequence of DDS_SampleInfo
int            max_samples;
int            sample_state_mask = SampleStateMask.ANY_SAMPLE_STATE;
int            view_state_mask = ViewStateMask.ANY_VIEW_STATE;
int            instance_state_mask = InstanceStateMask.ANY_INSTANCE_STATE;
try {
    reader.read(data_seq, info_seq,
                max_samples,
                sample_state_mask,
                view_state_mask,
                instance_state_mask);
} catch (RETCODE_ERROR) {
    // ... check for cause of failure
}
```

- Use the received data
```
// Use the received data samples 'data_seq' and associated
// information 'info_seq'
for (int i = 0; i < data_seq.size(); ++i) {
    // use... data_seq.get(i) ...
    // use... info_seq.get(i) ...
}
```

- Return the data samples and the information buffers back to the middleware
```
reader.return_loan(data_seq, info_seq);
```

6.58.7 Tearing down a data reader

- Delete DDSDataReader:
```
try {
    subscriber.delete_datareader(reader);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

6.59 Entity Use Cases

Working with entities.

Working with entities.

6.59.1 Enabling an entity

- To enable an **com.rti.dds.infrastructure.Entity** (p. 1029)
```
try {
    entity.enable();
} catch (RETCODE_ERROR err) {
    System.out.println(
        "*** Error: failed to enable entity");
}
```

6.59.2 Checking if a status changed on an entity.

- Given an **com.rti.dds.infrastructure.Entity** (p. 1029) and a **com.rti.dds.infrastructure.StatusKind** (p. 1701) to check for, get the list of statuses that have changed since the last time they were respectively cleared.
```
int status_changes_list = entity.get_status_changes();
```

- Check if `status_kind` was changed since the last time it was cleared. A plain communication status change is cleared when the status is read using the entity's `get_<plain communication status>()` method. A read communication status change is cleared when the data is taken from the middleware via a TDataReader_↩ take() call [see **Changes in Status** (p. 264) for details].
```
if ((status_changes_list & status_kind) != 0) {
    return true;  /* ... YES, status_kind changed ... */
} else {
    return false;  /* ... NO, status_kind did NOT change ... */
}
```

6.59.3 Changing the QoS for an entity

The QoS for an entity can be specified at the entity creation time. Once an entity has been created, its QoS can be manipulated as follows.

- Get an entity's QoS settings using **get_qos (abstract)** (p. 1031)
```
try {
    entity.get_qos(qos);
} catch (RETCODE_ERROR err) {
    System.out.println("***Error: failed to get qos\n");
}
```
- Change the desired qos policy fields
```
/* Change the desired qos policies */
/* qos.policy.field = ... */
```
- Set the qos using **set_qos (abstract)** (p. 1030).
```
try {
    entity.set_qos(qos);
} catch (RETCODE_IMMUTABLE_POLICY immutable) {
    System.out.println(
            "***Error: tried changing a policy that can only be" +
            "          set at entity creation time\n");
} catch (RETCODE_INCONSISTENT_POLICY inconsistent) {
    System.out.println(
            "***Error: tried changing a policy to a value inconsistent" +
            "          with other policy settings\n");
} catch (RETCODE_ERROR other) {
    System.out.println(
            "***Error: tried changing a policy that can only be" +
            "          set at entity creation time\n");
}
```

6.59.4 Changing the listener and enabling/disabling statuses associated with it

The listener for an entity can be specified at the entity creation time. By default the listener is *enabled* for all the statuses supported by the entity.

Once an entity has been created, its listener and/or the statuses for which it is enabled can be manipulated as follows.

- User defines entity listener methods
```
/* ... methods defined by EntityListener ... */
public class MyEntityListener implements Listener {
    // ... methods defined by EntityListener ...
}
```
- Get an entity's listener using **get_listener (abstract)** (p. 1032)
```
entity_listener = entity.get_listener();
```
- Enable `status_kind` for the `listener`
```
enabled_status_list |= status_kind;
```
- Disable `status_kind` for the `listener`
```
enabled_status_list &= ~status_kind;
```
- Set an entity's listener to `entity_listener` using **set_listener (abstract)** (p. 1031). Only enable the listener for the statuses specified by the `enabled_status_list`.
```
try {
    entity.set_listener(entity_listener, enabled_status_list);
} catch (RETCODE_ERROR err) {
    // respond to failure
}
```

6.59.5 Enabling/Disabling statuses associated with a status condition

Upon entity creation, by default, all the statuses are *enabled* for the DDS_StatusCondition associated with the entity.

Once an entity has been created, the list of statuses for which the DDS_StatusCondition is triggered can be manipulated as follows.

- Given an `entity`, a `status_kind`, and the associated `status_condition`:
  ```
  statuscondition = entity.get_statuscondition();
  ```
  - Get the list of statuses enabled for the `status_condition`
    ```
    enabled_status_list = statuscondition.get_enabled_statuses();
    ```
- Check if the given `status_kind` is enabled for the `status_condition`
  ```
  if ((enabled_status_list & status_kind) > 0) {
      /*... YES, status_kind is enabled ... */
  } else {
      /* ... NO, status_kind is NOT enabled ... */
  }
  ```
- Enable `status_kind` for the `status_condition`
  ```
  try {
      statuscondition.set_enabled_statuses(enabled_status_list | status_kind);
  } catch (RETCODE_ERROR err) {
      // ... check for cause of failure
  }
  ```
- Disable `status_kind` for the `status_condition`
  ```
  try {
      statuscondition.set_enabled_statuses(enabled_status_list & ~status_kind);
  } catch (RETCODE_ERROR err) {
      // ... check for cause of failure
  }
  ```

6.60 Waitset Use Cases

Using wait-sets and conditions.

Using wait-sets and conditions.

6.60.1 Setting up a wait-set

- Create a wait-set
  ```
  WaitSet waitset = new WaitSet();
  ```
- Attach conditions
  ```
  Condition cond1 = ...;
  Condition cond2 = entity.get_statuscondition();
  Condition cond3 = reader.create_readcondition(
      SampleStateKind.NOT_READ_SAMPLE_STATE,
      ViewStateKind.ANY_VIEW_STATE,
      InstanceStateKind.ANY_INSTANCE_STATE);
  Condition cond4 = new GuardCondition();
  Condition cond5 = ...;
  waitset.attach_condition(cond1);
  waitset.attach_condition(cond2);
  waitset.attach_condition(cond3);
  waitset.attach_condition(cond4);
  waitset.attach_condition(cond5);
  ```

6.60.2 Waiting for condition(s) to trigger

- **Set up a wait-set** (p. 137)
- Wait for a condition to trigger or `timeout`, whichever occurs first

```
Duration_t timeout = new Duration_t(0, 1000000); // 1ms
ConditionSeq active_conditions = new ConditionSeq(); // list of active conditions
boolean is_cond1_triggered = false;
boolean is_cond2_triggered = false;
try {
    waitset.wait(active_conditions, timeout);
    // check if "cond1" or "cond2" are triggered:
    for (int i = 0; i < active_conditions.size(); ++i) {
        if (active_conditions.get(i) == cond1) {
            System.out.println("Cond1 was triggered!");
            is_cond1_triggered = true;
        }
        if (active_conditions.get(i) == cond2) {
            System.out.println("Cond2 was triggered!");
            is_cond2_triggered = true;
        }
    }
    if (is_cond1_triggered) {
        // ... do something because "cond1" was triggered ...
    }
    if (is_cond2_triggered) {
        // ... do something because "cond2" was triggered ...
    }
} catch (RETCODE_TIMEOUT timed_out) {
    // timeout!
    System.out.println(
        "Wait timed out!! None of the conditions was triggered.");
} catch (RETCODE_ERROR ex) {
    // ... check for cause of failure
     throw ex;
}
```

6.60.3 Tearing down a wait-set

- Delete the wait-set

```
waitset.delete();
waitset = null;
// let the wait set be garbage collected
```

6.61 Transport Use Cases

Working with pluggable transports.

Working with pluggable transports.

6.61.1 Changing the automatically registered built-in transports

- The **com.rti.dds.infrastructure.TransportBuiltinKind.MASK_DEFAULT** (p. 271) specifies the transport plugins that will be automatically registered with a newly created **com.rti.dds.domain.DomainParticipant** (p. 670) by default.
- This default can be changed by changing the value of the value of **TRANSPORT_BUILTIN** (p. 269) Qos Policy on the **com.rti.dds.domain.DomainParticipant** (p. 670)
- To change the **com.rti.dds.domain.DomainParticipantQos.transport_builtin** (p. 800) Qos Policy:

```
DomainParticipantQos participant_qos = new DomainParticipantQos();

factory.get_default_participant_qos(participant_qos);

participant_qos.transport_builtin.mask = TransportBuiltinKind.SHMEM |
                                         TransportBuiltinKind.UDPv4;
```

6.61.2 Changing the properties of the automatically registered builtin transports

The behavior of the automatically registered builtin transports can be altered by changing their properties.

- Tell the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 125)
- Get the property of the desired builtin transport plugin, say **com.rti.ndds.transport.UDPv4Transport** (p. 1943)

```
UDPv4Transport.Property_t property = new UDPv4Transport.Property_t();

TransportSupport.get_builtin_transport_property(participant, property);
```

- Change the property fields as desired. Note that the properties should be changed carefully, as inappropriate values may prevent communications. For example, the **com.rti.ndds.transport.UDPv4Transport** (p. 1943) properties can be changed to support **large messages** (assuming the underlying operating system's UDPv4 stack supports the large message size). Note: if message_size_max is increased from the default for any of the built-in transports, then the DDS_ReceiverPoolQosPolicy::buffer_size on the DomainParticipant should also be changed.

```
/* Decrease the UDPv4 maximum message size to 64K (small messages). */
property.message_size_max        =  9216;
property.recv_socket_buffer_size =  9216;
property.send_socket_buffer_size =  9216;
```

- Set the property of the desired builtin transport plugin, say **com.rti.ndds.transport.UDPv4Transport** (p. 1943)

```
TransportSupport.set_builtin_transport_property(participant, property);
```

- **Enable the participant** (p. 135) to turn on communications with other participants in the domain using the new properties for the automatically registered builtin transport plugins.

6.62 Filter Use Cases

Working with data filters.

Working with data filters.

6.62.1 Introduction

RTI Connext supports filtering data either during the exchange from **com.rti.dds.publication.DataWriter** (p. 553) to **com.rti.dds.subscription.DataReader** (p. 450), or after the data has been stored at the **com.rti.dds.subscription.DataReader** (p. 450).

Filtering during the exchange process is performed by a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), which is created by the **com.rti.dds.subscription.DataReader** (p. 450) as a way of specifying a subset of the data samples that it wishes to receive.

Filtering samples that have already been received by the **com.rti.dds.subscription.DataReader** (p. 450) is performed by creating a **com.rti.dds.subscription.QueryCondition** (p. 1510), which can then used to check for matching samples, be alerted when matching samples arrive, or retrieve matching samples through use of the **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) or **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077) functions. (Conditions may also be used with the APIs **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) and **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092).)

Filtering may be performed on any topic, either keyed or un-keyed, except the **Built-in Topics** (p. 51). Filtering may be perfomed on any field, subset of fields, or combination of fields, subject only to the limitations of the filter syntax, and some restrictions against filtering some *sparse value types* of the **Dynamic Data** (p. 65) API.

RTI Connext contains built in support for filtering using SQL syntax, described in the **Queries and Filters Syntax** (p. 104) module.

6.62.1.1 Overview of ContentFilteredTopic

Each **com.rti.dds.topic.ContentFilteredTopic** (p. 436) is created based on an existing **com.rti.dds.topic.Topic** (p. 1807). The **com.rti.dds.topic.Topic** (p. 1807) specifies the **field_names** and **field_types** of the data contained within the topic. The **com.rti.dds.topic.ContentFilteredTopic** (p. 436), by means of its `filter_expression` and `expression_parameters`, further specifies the *values* of the data which the **com.rti.dds.subscription.Data↩ Reader** (p. 450) wishes to receive.

Custom filters may also be constructed and utilized as described in the **Creating Custom Content Filters** (p. 142) module.

Once the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) has been created, a **com.rti.dds.subscription.Data↩ Reader** (p. 450) can be created using the filtered topic. The filter's characteristics are exchanged between the **com.↩ rti.dds.subscription.DataReader** (p. 450) and any matching **com.rti.dds.publication.DataWriter** (p. 553) during the discovery process.

If the **com.rti.dds.publication.DataWriter** (p. 553) allows (by **com.rti.dds.infrastructure.DataWriterResource↩ LimitsQosPolicy.max_remote_reader_filters** (p. 628)), and the **com.rti.dds.subscription.DataReader** (p. 450) 's **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853) is empty, then the **com.rti.dds.publication.↩ DataWriter** (p. 553) will filter out samples that don't meet the filtering criteria.

If disallowed by the **com.rti.dds.publication.DataWriter** (p. 553), or the **com.rti.dds.subscription.DataReader** (p. 450) has set the **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853), then the **com.rti.dds.↩ publication.DataWriter** (p. 553) sends all samples to the **com.rti.dds.subscription.DataReader** (p. 450), and the **com.rti.dds.subscription.DataReader** (p. 450) discards any samples that do not meet the filtering criteria.

The `expression_parameters` can be modified using **com.rti.dds.topic.ContentFilteredTopic.set_↩ expression_parameters** (p. 438). Any changes made to the filtering criteria by means of **com.rti.dds.topic.Content↩ FilteredTopic.set_expression_parameters** (p. 438), will be conveyed to any connected **com.rti.dds.publication.↩ DataWriter** (p. 553). New samples will be subject to the modified filtering criteria, but samples that have already been accepted or rejected are unaffected. However, if the **com.rti.dds.subscription.DataReader** (p. 450) connects to a **com.rti.dds.publication.DataWriter** (p. 553) that *re-sends* its data, the re-sent samples will be subjected to the new filtering criteria.

6.62.1.2 Overview of QueryCondition

com.rti.dds.subscription.QueryCondition (p. 1510) combine aspects of the content filtering capabilities of **com.rti.↩ dds.topic.ContentFilteredTopic** (p. 436) with state filtering capabilities of **com.rti.dds.subscription.ReadCondition** (p. 1514) to create a reconfigurable means of filtering or searching data in the **com.rti.dds.subscription.DataReader** (p. 450) queue.

com.rti.dds.subscription.QueryCondition (p. 1510) may be created on a disabled **com.rti.dds.subscription.Data↩ Reader** (p. 450), or after the **com.rti.dds.subscription.DataReader** (p. 450) has been enabled. If the **com.rti.dds.↩ subscription.DataReader** (p. 450) is enabled, and has already recevied and stored samples in its queue, then all data samples in the are filtered against the **com.rti.dds.subscription.QueryCondition** (p. 1510) filter criteria at the time that the **com.rti.dds.subscription.QueryCondition** (p. 1510) is created. (Note that an exclusive lock is held on the **com.rti.dds.subscription.DataReader** (p. 450) sample queue for the duration of the **com.rti.dds.subscription.↩ QueryCondition** (p. 1510) creation).

Once created, incoming samples are filtered against all **com.rti.dds.subscription.QueryCondition** (p. 1510) filter criteria at the time of their arrival and storage into the **com.rti.dds.subscription.DataReader** (p. 450) queue.

The number of **com.rti.dds.subscription.QueryCondition** (p. 1510) filters that an individual **com.rti.dds.↩ subscription.DataReader** (p. 450) may create is set by **com.rti.dds.infrastructure.DataReaderResourceLimits↩ QosPolicy.max_query_condition_filters** (p. 539), to an upper maximum of 32.

6.62.2 Filtering with ContentFilteredTopic

- **Set up subscriber** (p. 131)
- **Set up a topic** (p. 126)
- Create a ContentFilteredTopic, of user data type `Foo`:

```
String cft_param_list[] = {"1", "100"};
StringSeq cft_parameters = new StringSeq(java.util.Arrays.asList(cft_param_list));
ContentFilteredTopic cft = participant.create_contentfilteredtopic(
                                "ContentFilteredTopic",
                                topic,
                                "value > %0 AND value < %1",
                                cft_parameters);
if (cft == null) {
    System.err.println("create_contentfilteredtopic error\n");
    return;
}
```

- Create a FooReader using the ContentFilteredTopic:

```
FooDataReader reader = (FooDataReader)
                           subscriber.create_datareader(
                               cft,
                               datareader_qos, // or Subscriber.DATAREADER_QOS_DEFAULT //
                               listener, // or null //
                               StatusKind.STATUS_MASK_ALL);
if (reader == null) {
    System.err.println("create_datareader error\n");
    return;
}
```

  Once setup, reading samples with a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) is exactly the same as normal reads or takes, as decribed in **DataReader Use Cases** (p. 133).

- Changing filter crieria using set_expression_parameters:

```
cft_parameters.set(0, "5");
cft_parameters.set(1, "9");
cft.set_expression_parameters(cft_parameters);
```

6.62.3 Filtering with Query Conditions

- Given a data reader of type Foo

```
FooDataReader reader = ...;
```

- Creating a QueryCondition

```
QueryCondition queryCondition = null;
String qc_param_list[] = {"1", "100"};
StringSeq qc_parameters = new StringSeq(java.util.Arrays.asList(qc_param_list));
queryCondition = reader.create_querycondition(SampleStateKind.NOT_READ_SAMPLE_STATE,
                                              ViewStateKind.ANY_VIEW_STATE,
                                              InstanceStateKind.ANY_INSTANCE_STATE,
                                              "value > %0 AND value < %1",
                                              qc_parameters);
if (queryCondition == null) {
    System.err.println("create_querycondition error\n");
    return;
}
```

- Reading matching samples with a **com.rti.dds.subscription.QueryCondition** (p. 1510)

```
FooSeq _dataSeq = new FooSeq();
SampleInfoSeq _infoSeq = new SampleInfoSeq();
try {
    reader.read_w_condition(_dataSeq, _infoSeq,
                            ResourceLimitsQosPolicy.LENGTH_UNLIMITED,
                            queryCondition);
    for(int i = 0; i < _dataSeq.size(); ++i) {
        SampleInfo info = (SampleInfo)_infoSeq.get(i);
        if (info.valid_data) {
            // --- Process data here --- //
        }
    }
```

```
} catch (RETCODE_NO_DATA noData) {
    // No data to process
} finally {
    reader.return_loan(_dataSeq, _infoSeq);
}
```

- **com.rti.dds.subscription.QueryCondition.set_query_parameters** (p. 1511) is used similarly to **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 438), and the same coding techniques can be used.
- Any **com.rti.dds.subscription.QueryCondition** (p. 1510) that have been created must be deleted before the **com.rti.dds.subscription.DataReader** (p. 450) can be deleted. This can be done using **com.rti.dds.subscription.DataReader.delete_contained_entities** (p. 469) or manually as in:
```
retcode = reader.delete_readcondition(queryCondition);
```

6.62.4 Filtering Performance

Although RTI Connext supports filtering on any field or combination of fields using the SQL syntax of the built-in filter, filters for keyed topics that filter solely on the contents of key fields have the potential for much higher performance. This is because for key-only filters, the **com.rti.dds.subscription.DataReader** (p. 450) caches the results of the filter (pass or not pass) for each instance. When another sample of the same instance is seen at the **com.rti.dds.subscription.DataReader** (p. 450), the filter results are retrieved from the cache, dispensing with the need to call the filter function.

This optimization applies to all filtering using the built-in SQL filter, performed by the **com.rti.dds.subscription.DataReader** (p. 450), for either **com.rti.dds.topic.ContentFilteredTopic** (p. 436) or **com.rti.dds.subscription.QueryCondition** (p. 1510). This does *not* apply to filtering perfomed for **com.rti.dds.topic.ContentFilteredTopic** (p. 436) by the **com.rti.dds.publication.DataWriter** (p. 553).

6.63 Creating Custom Content Filters

Working with custom content filters.

Working with custom content filters.

6.63.1 Introduction

By default, RTI Connext creates content filters with the DDS_SQL_FILTER, which implements a superset of the DDS-specified SQL WHERE clause. However, in many cases this filter may not be what you want. Some examples are:

- The default filter can only filter based on the content of a sample, not on a computation on the content of a sample. You can use a custom filter that is customized for a specific type and can filter based on a computation of the type members.
- You want to use a different filter language then SQL

This HOWTO explains how to write your own custom filter and is divided into the following sections:

- **The Custom Content Filter API** (p. 143)
- **Example Using C format strings** (p. 143)

6.63.2 The Custom Content Filter API

A custom content filter is created by calling the **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740) function with a **com.rti.dds.topic.ContentFilter** (p. 433) that contains a **compile**, an **evaluate** function and a **finalize** function. **com.rti.dds.topic.ContentFilteredTopic** (p. 436) can be created with **com.rti.dds.domain.Domain↩ Participant.create_contentfilteredtopic_with_filter** (p. 711) to use this filter.

A custom content filter is used by RTI Connext at the following times during the life-time of a **com.rti.dds.topic.↩ ContentFilteredTopic** (p. 436) (the function called is shown in parenthesis).

- When a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) is created (**compile** (p. 143))
- When the filter parameters are changed on the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) (**compile** (p. 143)) with **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 438)
- When a sample is filtered (**evaluate** (p. 143)). This function is called by the RTI Connext core with a de-serialized sample
- When a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) is deleted (**finalize** (p. 143))

6.63.2.1 The compile function

The **compile** (p. 144) function is used to **compile** a filter expression and expression parameters. Please note that the term **compile** is intentionally loosely defined. It is up to the user to decide what this function should do and return.

See **com.rti.dds.topic.ContentFilter.compile** (p. 434) for details.

6.63.2.2 The evaluate function

The **evaluate** (p. 144) function is called each time a sample is received to determine if a sample should be filtered out and discarded.

See **com.rti.dds.topic.ContentFilter.evaluate** (p. 435) for details.

6.63.2.3 The finalize function

The **finalize** (p. 144) function is called when an instance of the custom content filter is no longer needed. When this function is called, it is safe to free all resources used by this particular instance of the custom content filter.

See **com.rti.dds.topic.ContentFilter.finalize** (p. 435) for details.

6.63.3 Example Using C format strings

Assume that you have a type Foo.

You want to write a custom filter function that will drop all samples where the value of Foo.x $>$ x and x is a value determined by an expression parameter. The filter will **only** be used to filter samples of type Foo.

6.63.3.1 Writing the Compile Function

The first thing to note is that we can ignore the filter expression, since we already know what the expression is. The second is that x is a parameter that can be changed. By using this information, the compile function is very easy to implement. Simply return the parameter string. This string will then be passed to the evaluate function every time a sample of this type is filtered.

Below is the entire **compile** (p. 143) function.

```
public void compile(
    ObjectHolder new_compile_data, String expression,
    StringSeq parameters, TypeCode type_code, String type_class_name,
    Object old_compile_data) {
    new_compile_data.value = parameters.get(0);
}
```

6.63.3.2 Writing the Evaluate Function

The next step is to implement the **evaluate** function. The evaluate function receives the parameter string with the actual value to test against. Thus the evaluate function must read the actual value from the parameter string before evaluating the expression. Below is the entire **evaluate** (p. 143) function.

```
public boolean evaluate(
    Object compile_data, Object sample, FilterSampleInfo meta_data) {
    String parameter = (String)compile_data;
    int x;

    Foo foo_sample = (Foo)sample;

    x = Integer.parseInt(parameter);
    return (foo_sample.x > x ? false : true);
}
```

6.63.3.3 Writing the Finalize Function

The last function to write is the finalize function. It is safe to free all resources used by this particular instance of the custom content filter that is allocated in **compile**. Below is the entire **finalize** (p. 143) function.

```
public void finalize(
    Object compile_data) {
    /* nothing to do since no resource are allocated */
}
```

6.63.3.4 Registering the Filter

Before the custom filter can be used, it must be registered with RTI Connext:

```
ContentFilter myCustomFilter = new MyContentFilter();
participant.register_contentfilter("MyCustomFilter", myCustomFilter);
```

6.63.3.5 Unregistering the Filter

When the filter is no longer needed, it can be unregistered from RTI Connext:

```
participant.unregister_contentfilter("MyCustomFilter");
```

6.64 Large Data Use Cases

Working with large data types.

Working with large data types.

6.64.1 Introduction

RTI Connext supports data types whose size exceeds the maximum message size of the underlying transports. A **com.rti.dds.publication.DataWriter** (p. 553) will fragment data samples when required. Fragments are automatically reassembled at the receiving end.

Once all fragments of a sample have been received, the new sample is passed to the **com.rti.dds.subscription.↩ DataReader** (p. 450) which can then make it available to the user. Note that the new sample is treated as a regular sample at that point and its availability depends on standard QoS settings such as **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.max_samples** (p. 1592) and com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQos↩ PolicyKind.KEEP_LAST_HISTORY_QOS.

The large data feature is fully supported by all DDS API's, so its use is mostly transparent. Some additional considerations apply as explained below.

6.64.2 Writing Large Data

In order to use the large data feature with the **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE↩ _RELIABILITY_QOS** (p. 1532) setting, the **com.rti.dds.publication.DataWriter** (p. 553) must be configured as an asynchronous writer (com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.↩ ASYNCHRONOUS_PUBLISH_MODE_QOS) with associated **com.rti.dds.publication.FlowController** (p. 1055).

While the use of an asynchronous writer and flow controller is optional when using the **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532) setting, most large data use cases will benefit from the use of a flow controller to prevent flooding the network when fragments are being sent.

- **Set up writer** (p. 130)
- **Add flow control** (p. 127)

6.64.3 Receiving Large Data

Large data is supported by default and in most cases, no further changes are required.

The **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529) allows tuning the resources available to the **com.rti.dds.subscription.DataReader** (p. 450) for reassembling fragmented large data.

- **Set up reader** (p. 133)

6.65 Request-Reply Examples

Examples on how to use the request-reply API .

Examples on how to use the request-reply API .

Request-Reply code examples.

6.65.1 Request-Reply Examples

Requesters and Repliers provide a way to use the Request-Reply communication pattern on top of the DDS entities. An application uses a Requester to send requests to a Replier; another application using a Replier receives a request and can send one or more replies for that request. The Requester that sent the request (and only that one) will receive the reply (or replies).

DDS Types

RTI Connext uses DDS data types for sending and receiving requests and replies. Valid types are those generated by the rtiddsgen code generator, the DDS built-in types, and `DynamicData`. Refer to the `Core Libraries User's Manual` and the following links for more information:

- `Code Generator User's Manual`,
- **Using the DDS built-in types** (p. 61),
- **Using DynamicData** (p. 65)

Set up

- **Create a DomainParticipant** (p. 125)
- **Create a requester** (p. 147)
- **Create a requester with parameters** (p. 148)
- **Create a replier** (p. 151)

Requester: sending requests and receiving replies

- **Basic Requester example** (p. 148)
- **Taking loaned samples** (p. 149)
- **Taking samples by copy** (p. 150)
- **Correlation between requests and replies** (p. 150)

Replier: receiving requests and sending replies

- **Basic Replier example** (p. 152)
- **SimpleReplier example** (p. 152)
- **Taking loaned samples** (p. 149)
- **Taking samples by copy** (p. 150)

Error handling

- **Error handling example** (p. 153)

Note

To use Request-Reply you need to include the additional `rticonnextmsg.jar` to build your Java application.

6.65.2 Creating a Requester

- **Setting up a participant** (p. 125)
- Creating a Requester

```
// Note: error checking omitted for simplicity

// Create a DomainParticipant
DomainParticipant participant =
        DomainParticipantFactory.get_instance().create_participant(
            domainId, DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
            null, StatusKind.STATUS_MASK_NONE);

// Create a Requester (if creation fails an exception is thrown)
// Foo is the request type and FooTypeSupport is its DDS TypeSupport
// Bar is the reply type and BarTypeSupport is its DDS TypeSupport
Requester<Foo, Bar> requester =
        new Requester<Foo, Bar>(
                participant, "TestService",
                FooTypeSupport.get_instance(),
                BarTypeSupport.get_instance());
```

- Finalizing a Requester

```
requester.close();
```

See also

Creating a Requester with optional parameters (p. 148)

Configuring Request-Reply QoS profiles (p. 153)

6.65.3 Creating a Requester with optional parameters

- **Setting up a participant** (p. 125)
- Creating a Requester with optional parameters

```
// Note: error checking omitted for simplicity

// Create a DomainParticipant
DomainParticipant participant =
        DomainParticipantFactory.get_instance().create_participant(
                0, DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null, StatusKind.STATUS_MASK_NONE);

// Create a Requester with a QoS profile (located for example in
// USER_QOS_PROFILES.xml, in the current working directory)
Requester<Foo, Bar> requester =
        new Requester<Foo, Bar>(
            new RequesterParams(
                participant,
                FooTypeSupport.get_instance(),
                BarTypeSupport.get_instance())
            .setServiceName("TestService")
            .setQosProfile("RequestReplyExampleProfiles",
                           "RequesterExampleProfile"));
```

See also

Creating a Requester (p. 147)

6.65.4 Basic Requester example

- **Creating a Requester** (p. 147)
- **Creating a Requester with optional parameters** (p. 148)
- Basic Requester example

```
// Send request
Foo request = new Foo();
request.message = "A request";
requester.sendRequest(request);

// Create reply Sample
Sample<Bar> reply = requester.createReplySample();

// Receive reply (wait for it and get the sample)
boolean received = requester.receiveReply(reply , MAX_WAIT);

if (received) {
    if (reply.getInfo().valid_data) {
        System.out.println("Received reply: " + reply.getData().message);
    } else {
        System.out.println("Received invalid reply");
    }
} else {
    System.out.println("Reply not received");
}
```

- Basic Requester example using the **String built-in type** (p. 285)

```
// Create a requester using the String
// type with the DDS StringTypeSupport
Requester<String, String> requester =
        new Requester<String, String>(
            new RequesterParams(
                participant,
                StringTypeSupport.get_instance(),
                StringTypeSupport.get_instance())
            .setServiceName("TestServiceString")
```

```
            .setQosProfile("MyLibrary", "RequesterProfile"));

// Send request
requester.sendRequest("A request");

// Create reply Sample
Sample<String> reply = requester.createReplySample();

// Receive reply (wait for it and get the sample)
boolean received = requester.receiveReply(reply , MAX_WAIT);

if (received && reply.getInfo().valid_data) {
    System.out.println("Received reply: " + reply.getData());
}
```

- Basic Requester example using **DynamicData** (p. 65).

```
// Instantiate DynamicData type support. For simplicity we create them
// from existing, generated TypeCode objects (Foo and Bar)
DynamicDataTypeSupport fooDynamicTypeSupport = new DynamicDataTypeSupport(
        FooTypeCode.VALUE, new DynamicDataTypeProperty_t());
DynamicDataTypeSupport barDynamicTypeSupport = new DynamicDataTypeSupport(
        BarTypeCode.VALUE, new DynamicDataTypeProperty_t());

// Create a requester using a DynamicData type support
Requester<DynamicData, DynamicData> requester =
        new Requester<DynamicData, DynamicData>(
                participant, "TestService",
                fooDynamicTypeSupport,
                barDynamicTypeSupport);

// Create a Foo instance
DynamicData foo = (DynamicData) fooDynamicTypeSupport.create_data();
foo.set_string("message", DynamicData.MEMBER_ID_UNSPECIFIED,
        "A dynamic request");

// Send request
requester.sendRequest(foo);

// Create reply Sample
Sample<DynamicData> reply = requester.createReplySample();

// Receive reply (wait for it and get the sample)
boolean received = requester.receiveReply(reply , MAX_WAIT);

if (received && reply.getInfo().valid_data) {
    System.out.println("Received reply: " +
        reply.getData().get_string("message", DynamicData.MEMBER_ID_UNSPECIFIED));
}
```

See also

Basic Replier example (p. 152)

6.65.5 Taking loaned samples

- Get iterator to loaned replies (no copies)

```
// Take loaned samples
// Unlike receiveReplies(), takeReplies() doesn't block. It returns an
// empty collection if there are no replies at that moment.
// (before that we can call requester.wait_for_replies)
Sample.Iterator<Bar> replies = requester.takeReplies();
try {
    while (replies.hasNext()) {
        Sample<Bar> reply = replies.next();
        System.out.println("Received: " + reply.getData().message);
    }
} finally {
    replies.close();
}
```

A more compact version using a `try-with-resources` block (since Java 7):

```
try (Sample.Iterator<String> replies = requester.getReplies()) {
    while (replies.hasNext()) {
        Sample<String> reply = replies.next();
        System.out.println("Received: " + reply.getData());
    }
}
```

See also

Basic Requester example (p. 148)

Basic Replier example (p. 152)

Taking samples by copy (p. 150)

6.65.6 Taking samples by copy

- Get copies of the replies:

```
ArrayList<Sample<Bar» replies = new ArrayList<Sample<Bar»();

// Add to existing list, reusing and overwriting any existing elements
requester.takeReplies(replies, 10);
for (Sample<Bar> reply : replies) {
    System.out.println("Received: " + reply.getData().message);
}

// Get replies in a newly created list
List<Sample<Bar» newList = requester.takeReplies(null, 10);
for (Sample<Bar> reply : newList) {
    System.out.println("Received: " + reply.getData().message);
}
```

See also

Basic Replier example (p. 152)

Taking loaned samples (p. 149)

6.65.7 Correlating requests and replies

- **Creating a Requester** (p. 147)
- Example 1) Waiting for a reply for a specific request

```
// Create requests
WriteSample<Foo> request1 = requester.createRequestSample();
request1.getData().message = "Request 1";
WriteSample<Foo> request2 = requester.createRequestSample();
request2.getData().message = "Request 2";

// Send requests using WriteSample
requester.sendRequest(request1);
requester.sendRequest(request2);
// Wait for a reply to the second request
Sample<Bar> reply = requester.createReplySample();
requester.waitForReplies(1, MAX_WAIT, request2.getIdentity());
requester.takeReply(reply, request2.getIdentity());

assert reply.getRelatedIdentity().equals(request2.getIdentity());
```

```
System.out.println("Received reply: " + reply.getData().message);
// Wait for a reply to the first request
requester.waitForReplies(1, MAX_WAIT, request1.getIdentity());
requester.takeReply(reply, request1.getIdentity());

assert reply.getRelatedIdentity().equals(request1.getIdentity());
System.out.println("Received reply: " + reply.getData().message);
```

- Example 2) Correlate reply after receiving it

```
// Create requests
WriteSample<Foo> request1 = requester.createRequestSample();
request1.getData().message = "Request 1";
WriteSample<Foo> request2 = requester.createRequestSample();
request2.getData().message = "Request 2";

// Send requests using WriteSample
requester.sendRequest(request1);
requester.sendRequest(request2);
// Wait for two replies
boolean received = requester.waitForReplies(2, MAX_WAIT);
if (!received) {
    System.out.println("Replies not received");
    return;
}

Sample.Iterator<Bar> replies = requester.takeReplies();
try {
    while(replies.hasNext()) {
        Sample<Bar> reply = replies.next();
        if (reply.getRelatedIdentity().equals(
                request1.getIdentity())) {
            System.out.println(
                "Received reply for request 1: " + reply.getData().message);
        } else if (reply.getRelatedIdentity().equals(
                request2.getIdentity())) {
            System.out.println(
                "Received reply for request 2: " + reply.getData().message);
        } else {
            // Should not happen
            System.out.println("Received unexpected reply");
        }
    }
} finally {
    // Return the loan
    replies.close();
}
```

See also

Basic Requester example (p. 148)

Basic Replier example (p. 152)

6.65.8 Creating a Replier

- **Setting up a participant** (p. 125)
- Creating a Replier

```
// Create a DomainParticipant
DomainParticipant participant =
        DomainParticipantFactory.get_instance().create_participant(
                domain_id, DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null, StatusKind.STATUS_MASK_NONE);

// Create a Replier
Replier<Foo, Bar> replier =
        new Replier<Foo, Bar>(
                participant, "TestService",
                FooTypeSupport.get_instance(),
                BarTypeSupport.get_instance());
```

- Finalizing a Replier

```
replier.close();
```

6.65.9 Basic Replier example

- **Creating a Replier** (p. 151)
- Basic Replier example

```
// Create a Sample for reading
Sample<Foo> request = replier.createRequestSample();

// Receive one request
boolean received = replier.receiveRequest(request, MAX_WAIT);

if (!received) {
    System.out.println("Request not received");
    return false;
}

if (request.getInfo().valid_data) {
    // Send a reply for that request
    Bar reply = new Bar();
    reply.message = "Reply for " + request.getData().message;
    replier.sendReply(reply, request.getIdentity());
}

// Note: a replier can send more than one reply for the same request
```

See also

Basic Requester example (p. 148)

6.65.10 SimpleReplier example

- Implement a listener

```
class MySimpleReplierListener implements SimpleReplierListener<Foo, Bar> {
    private Bar reply = new Bar();

    public Bar onRequestAvailable(Sample<Foo> request) {
        if (request.getInfo().valid_data) {
            // Return a reply
            reply.message = "Simple reply for " + request.getData().message;
            return reply;
        } else {
            // Do not reply to dispose/unregister samples
            return null;
        }
    }
    public void returnLoan(Bar reply) {
        // nothing to do
    }

}
```

- And create a SimpleReplier with the listener:

```
// Create a SimpleReplier using a DomainParticipant, service name and
// the a SimpleReplierListener implementation
SimpleReplier<Foo, Bar> replier =
        new SimpleReplier<Foo, Bar>(
                participant, "TestService",
                new MySimpleReplierListener(),
                FooTypeSupport.get_instance(),
                BarTypeSupport.get_instance());

// After creation the SimpleReplier is already active and may call
// the listener
```

See also

Basic Requester example (p. 148)

6.65.11 Error handling example

- Catching an exception from the request-reply API

```
// All the operations in the request-reply API
// throw exceptions in case of error.
// You can also set the verbosity level to warning for
// additional information
Logger.get_instance().set_verbosity(
        LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING);
//
// For example, this replier is sending a reply for a nonexistent request
//
WriteSample<Bar> reply = replier.createReplySample();
SampleIdentity_t invalidRequestId = new SampleIdentity_t();


try {
    replier.sendReply(reply, invalidRequestId);
} catch (RETCODE_BAD_PARAMETER ex) {
    // You can also catch RETCODE_ERROR, which is more generic
    System.out.println("Exception while sending reply: " + ex);
}
```

6.65.12 Configuring Request-Reply QoS profiles

If you do not specify your own quality of service parameters (in **com.rti.connext.requestreply.Requester↩ Params** (p. 1586) and com.rti.connext.requestreply.ReplierParams<TReq,TRep>), a com.rti.connext.requestreply.↩ Requester<TReq,TRep> and com.rti.connext.requestreply.Replier<TReq,TRep> are created using a default configuration. That configuration is equivalent to the one in the following QoS profile called "default":

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Description
XML QoS Profile for HelloWorld

(c) Copyright, Real-Time Innovations, 2012.  All rights reserved.
RTI grants Licensee a license to use, modify, compile, and create derivative
works of the software solely for use with RTI Connext DDS. Licensee may
redistribute copies of the software provided that all such copies are
subject to this license. The software is provided "as is", with no warranty
of any type, including any warranty for fitness for any purpose. RTI is
under no obligation to maintain or support the software. RTI shall not be
liable for any incidental or consequential damages arising out of the use
or inability to use the software.

The QoS configuration of the DDS entities in the generated example is loaded
from this file.

This file is used only when it is in the current working directory or when the
environment variable NDDS_QOS_PROFILES is defined and points to this file.

The profile in this file inherits from the builtin QoS profile
BuiltinQosLib::Generic.StrictReliable. That profile, along with all of the
other built-in QoS profiles can be found in the
BuiltinProfiles.documentationONLY.xml file located in the
$NDDSHOME/resource/xml/ directory.

You may use any of these QoS configurations in your application simply by
referring to them by the name shown in the
BuiltinProfiles.documentationONLY.xml file.

Also, following the QoS Profile composition pattern you can use QoS Snippets
to easily create your final QoS Profile. For further information visit:
https://community.rti.com/best-practices/qos-profile-inheritance-and-composition-guidance

There is a QoS Snippet library that contains a collection of
QoS Snippets that set some specific features or configurations. You can find
```

```
them in the BuiltinProfiles.documentationONLY.xml file as well.

You should not edit the file BuiltinProfiles.documentationONLY.xml directly.
However, if you wish to modify any of the values in a built-in profile, the
recommendation is to create a profile of your own and inherit from the built-in
profile you wish to modify. The NDDS_QOS_PROFILES.example.xml file (contained in
the same directory as the BuiltinProfiles.documentationONLY.xml file) shows how
to inherit from the built-in profiles.

For more information about XML QoS Profiles see the "Configuring QoS with
XML" chapter in the RTI Connext DDS Core Libraries User's Manual.
-->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:noNamespaceSchemaLocation="file:////rti/jenkins/workspace/connextdds_htmldocs_release_connextdds_7.3.0/build/nddsgen.2
    <!-- QoS Library containing the QoS profile used in the generated example.

         A QoS library is a named set of QoS profiles.
    -->
    <qos_library name="HelloWorld_Library">
        <!-- QoS profile used to configure reliable communication between the DataWriter
             and DataReader created in the example code.

             A QoS profile groups a set of related QoS.
        -->
        <qos_profile name="HelloWorld_Profile" base_name="BuiltinQosLib::Generic.StrictReliable"
       is_default_qos="true">
            <!-- QoS used to configure the data writer created in the example code -->
            <datawriter_qos>
                <publication_name>
                    <name>HelloWorldDataWriter</name>
                </publication_name>
            </datawriter_qos>
            <!-- QoS used to configure the data reader created in the example code -->
            <datareader_qos>
                <subscription_name>
                    <name>HelloWorldDataReader</name>
                </subscription_name>
            </datareader_qos>
            <domain_participant_qos>
                <!--
                The participant name, if it is set, will be displayed in the
                RTI tools, making it easier for you to tell one
                application from another when you're debugging.
                -->
                <participant_name>
                    <name>HelloWorldParticipant</name>
                    <role_name>HelloWorldParticipantRole</role_name>
                </participant_name>
            </domain_participant_qos>
        </qos_profile>
    </qos_library>
</dds>
```

You can use the profile called "RequesterExampleProfile", which modifies some parameters from the default. The example **Creating a Requester with optional parameters** (p. 148) shows how to create a com.rti.connext.requestreply.↩ Requester<TReq,TRep> using this profile.

See also

Creating a Requester with optional parameters (p. 148)

Configuring QoS Profiles with XML (p. 120)

6.66 Documentation Roadmap

This section contains a roadmap for the new user with pointers on what to read first.

If you are new to RTI Connext, we recommend starting in the following order:

- See the `Getting Started Guide`. This document provides download and installation instructions. It also lays out the core value and concepts behind the product and takes you step-by-step through the creation of a simple example application.
- The `User's Manual` describes the features of the product and how to use them. It is organized around the structure of the DDS APIs and certain common high-level tasks.

- The documentation in the **RTI Connext DDS API Reference** (p. 157) provides an overview of API classes and modules for the DDS data-centric publish-subscribe (DCPS) package from a programmer's perspective. Start by reading the documentation on the main page.
- After reading the high level module documentation, look at the **Publication Example** (p. 122) and **Subscription Example** (p. 123) for step-by-step examples of creating a publication and subscription. These are hyperlinked code snippets to the full API documentation, and provide a good place to begin learning the APIs.

- Next, work through your own application using the example code files generated by rtiddsgen. See the `Code Generator User's Manual`.
- To integrate similar code into your own application and build system, you will likely need to refer to the `Platform Notes`.

6.67 Conventions

This section describes the conventions used in the API documentation.

6.67.1 Unsupported Features

[Not supported (optional)] This note means that the optional feature from the DDS specification is not supported in the current release.

6.67.2 API Documentation Terms

In the API documentation, the term module refers to a logical grouping of documentation and elements in the API.

6.67.3 Stereotypes

Commonly used stereotypes in the API documentation include the following.

6.67.3.1 Extensions

- *<<**extension**>> (p. 155)*
  - An RTI Connext product extension to the DDS standard specification.
  - The extension APIs complement the standard APIs specified by the OMG DDS specification. They are provided to improve product usability and enable access to product-specific features such as pluggable transports.

6.67.3.2 Experimental

- *<<**experimental**>> (p. 156)*
  - RTI Connext experimental features are used to evaluate new features and get user feedback.
  - These features are not guaranteed to be fully supported and might be implemented only of some of the programming languages supported bt RTI Connext
  - The functional APIs corresponding to experimental features can be distinguished from other APIs by the suffix '_exp'.
  - Experimental features may or may not appear in future product releases.
  - The name of the experimental features APIs will change if they become officially supported. At the very least the suffix '_exp' will be removed.
  - Experimental features should not be used in production.

6.67.3.3 Types

- *<<**interface**>> (p. 156)*
  - Pure interface type with *no state*.
  - Languages such as Java natively support the concept of an *interface* type, which is a collection of method signatures devoid of any dynamic state.
  - In C++, this is achieved via a class with all *pure virtual* methods and devoid of any instance variables (ie no dynamic state).
  - Interfaces are generally organized into a type hierarchy. Static typecasting along the interface type hierarchy is "safe" for valid objects.
- *<<**generic**>> (p. 156)*
  - A *generic* type is a *skeleton* class written in terms of generic parameters. Type-specific instantiations of such types are conventionally referred to in this documentation in terms of the hypothetical type "Foo"; for example: FooSeq, FooDataType, FooDataWriter, and FooDataReader.
- *<<**singleton**>> (p. 156)*
  - Singleton class. There is a single instance of the class.
  - Generally acccessed via a `get_instance()` static method.

6.67.3.4 Method Parameters

- *<<**in**>> (p. 156)*
  - An *input* parameter.
- *<<**out**>> (p. 156)*
  - An *output* parameter.
- *<<**inout**>> (p. 156)*
  - An *input* and *output* parameter.

6.68 RTI Connext DDS API Reference

RTI Connext modules following the DDS module definitions.

Modules

- **Domain Module**

  *Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.*
- **Topic Module**

  *Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.*
- **Publication Module**

  *Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.*
- **Subscription Module**

  *Contains the **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.subscription.ReadCondition** (p. 1514), **com.rti.dds.subscription.QueryCondition** (p. 1510), and **com.rti.dds.subscription.TopicQuery** (p. 1830) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1755) and **com.rti.dds.subscription.DataReaderListener** (p. 497) interfaces, and more generally, all that is needed on the subscription side.*
- **Infrastructure Module**

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

  APIs related to RTI Connext pluggable transports.
- **Queries and Filters Syntax**
- **Logging and Version**

  APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.
- **General Utilities and Compliance Configuration**

  API of general utilities used in the RTI Connext distribution.
- **Observability**

  API of RTI Connext Observability Framework.
- **Durability and Persistence**

  APIs related to RTI Connext Durability and Persistence.
- **System Properties**

  System Properties.
- **Configuring QoS Profiles with XML**

  APIs related to XML QoS Profiles.
- **Additional RTI Connext Communication Patterns**

  Extensions to the RTI Connext publish-subscribe functionality.

6.68.1 Detailed Description

RTI Connext modules following the DDS module definitions.

6.68.2 Overview

Information flows with the aid of the following constructs: **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.publication.DataWriter** (p. 553) on the sending side, **com.rti.dds.subscription.Subscriber** (p. 1730) and **com.rti.dds.subscription.DataReader** (p. 450) on the receiving side.

- A **com.rti.dds.publication.Publisher** (p. 1466) is an object responsible for data distribution. It may publish data of different data types. A TDataWriter acts as a *typed* (i.e. each **com.rti.dds.publication.DataWriter** (p. 553) object is dedicated to one application data type) accessor to a publisher. A **com.rti.dds.publication.DataWriter** (p. 553) is the object the application must use to communicate to a publisher the existence and value of data objects of a given type. When data object values have been communicated to the publisher through the appropriate data-writer, it is the publisher's responsibility to perform the distribution (the publisher will do this according to its own QoS, or the QoS attached to the corresponding data-writer). A *publication* is defined by the association of a data-writer to a publisher. This association expresses the intent of the application to publish the data described by the data-writer in the context provided by the publisher.
- A **com.rti.dds.subscription.Subscriber** (p. 1730) is an object responsible for receiving published data and making it available (according to the Subscriber's QoS) to the receiving application. It may receive and dispatch data of different specified types. To access the received data, the application must use a *typed* TDataReader attached to the subscriber. Thus, a *subscription* is defined by the association of a data-reader with a subscriber. This association expresses the intent of the application to subscribe to the data described by the data-reader in the context provided by the subscriber.

com.rti.dds.topic.Topic (p. 1807) objects conceptually fit between publications and subscriptions. Publications must be known in such a way that subscriptions can refer to them unambiguously. A **com.rti.dds.topic.Topic** (p. 1807) is meant to fulfill that purpose: it associates a name (unique in the domain i.e. the set of applications that are communicating with each other), a data type, and QoS related to the data itself. In addition to the topic QoS, the QoS of the **com.rti.dds.publication.DataWriter** (p. 553) associated with that Topic and the QoS of the **com.rti.dds.publication.Publisher** (p. 1466) associated to the **com.rti.dds.publication.DataWriter** (p. 553) control the behavior on the publisher's side, while the corresponding **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.subscription.Subscriber** (p. 1730) QoS control the behavior on the subscriber's side.

When an application wishes to publish data of a given type, it must create a **com.rti.dds.publication.Publisher** (p. 1466) (or reuse an already created one) and a **com.rti.dds.publication.DataWriter** (p. 553) with all the characteristics of the desired publication. Similarly, when an application wishes to receive data, it must create a **com.rti.dds.subscription.Subscriber** (p. 1730) (or reuse an already created one) and a **com.rti.dds.subscription.DataReader** (p. 450) to define the subscription.

6.68.3 Conceptual Model

The overall conceptual model is shown below.

Notice that all the main communication objects (the specializations of Entity) follow unified patterns of:

- Supporting QoS (made up of several QosPolicy); QoS provides a generic mechanism for the application to control the behavior of the Service and tailor it to its needs. Each **com.rti.dds.infrastructure.Entity** (p. 1029) supports its own specialized kind of QoS policies (see **QoS Policies** (p. 250)).

- Accepting a **com.rti.dds.infrastructure.Listener** (p. 1236); listeners provide a generic mechanism for the middleware to notify the application of relevant asynchronous events, such as arrival of data corresponding to a subscription, violation of a QoS setting, etc. Each **com.rti.dds.infrastructure.Entity** (p. 1029) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 262)).

  Note that only one Listener per entity is allowed (instead of a list of them). The reason for that choice is that this allows a much simpler (and, thus, more efficient) implementation as far as the middleware is concerned. Moreover, if it were required, the application could easily implement a listener that, when triggered, triggers in return attached 'sub-listeners'.

- Accepting a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) (and a set of **com.rti.dds.subscription.↩ ReadCondition** (p. 1514) objects for the **com.rti.dds.subscription.DataReader** (p. 450)); conditions (in conjunction with **com.rti.dds.infrastructure.WaitSet** (p. 1973) objects) provide support for an alternate communication style between the middleware and the application (i.e., wait-based rather than notification-based).

All DCPS entities are attached to a **com.rti.dds.domain.DomainParticipant** (p. 670). A domain participant represents the local membership of the application in a domain. A *domain* is a distributed concept that links all the applications able to communicate with each other. It represents a communication plane: only the publishers and the subscribers attached to the same domain may interact.

com.rti.dds.infrastructure.DomainEntity (p. 669) is an intermediate object whose only purpose is to state that a DomainParticipant cannot contain other domain participants.

At the DCPS level, data types represent information that is sent atomically. For performance reasons, only plain data structures are handled by this level.

By default, each data modification is propagated individually, independently, and uncorrelated with other modifications. However, an application may request that several modifications be sent as a whole and interpreted as such at the recipient side. This functionality is offered on a Publisher/Subscriber basis. That is, these relationships can only be specified among **com.rti.dds.publication.DataWriter** (p. 553) objects attached to the same **com.rti.dds.publication.Publisher** (p. 1466) and retrieved among **com.rti.dds.subscription.DataReader** (p. 450) objects attached to the same **com.rti.↩ dds.subscription.Subscriber** (p. 1730).

By definition, a **com.rti.dds.topic.Topic** (p. 1807) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **com.rti.dds.topic.Topic** (p. 1807) identifies data of a single type, ranging from one single instance to a whole collection of instances of that given type. This is shown below for the hypothetical data type `Foo`.

In case a set of instances is gathered under the same topic, different instances must be distinguishable. This is achieved by means of the values of some data fields that form the **key** to that data set. The *key description* (i.e., the list of data fields whose value forms the key) has to be indicated to the middleware. The rule is simple: *different data samples with the same key value represent successive values for the same instance, while different data samples with different key values represent different instances*. If no key is provided, the data set associated with the **com.rti.dds.topic.Topic** (p. 1807) is restricted to a *single instance*.

Topics need to be known by the middleware and potentially propagated. Topic objects are created using the create operations provided by **com.rti.dds.domain.DomainParticipant** (p. 670).

The interaction style is straightforward on the publisher's side: when the application decides that it wants to make data available for publication, it calls the appropriate operation on the related **com.rti.dds.publication.DataWriter** (p. 553) (this, in turn, will trigger its **com.rti.dds.publication.Publisher** (p. 1466)).

On the subscriber's side however, there are more choices: relevant information may arrive when the application is busy doing something else or when the application is just waiting for that information. Therefore, depending on the way the application is designed, asynchronous notifications or synchronous access may be more appropriate. Both interaction

modes are allowed, a **com.rti.dds.infrastructure.Listener** (p. 1236) is used to provide a callback for synchronous access and a **com.rti.dds.infrastructure.WaitSet** (p. 1973) associated with one or several **com.rti.dds.infrastructure.Condition** (p. 429) objects provides asynchronous data access.

The same synchronous and asynchronous interaction modes can also be used to access changes that affect the middleware communication status (see **Status Kinds** (p. 262)). For instance, this may occur when the middleware asynchronously detects an inconsistency. In addition, other middleware information that may be relevant to the application (such as the list of the existing topics) is made available by means of **built-in topics** (p. 51) that the application can access as plain application data, using built-in data-readers.

6.68.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 91) defines the abstract classes and the interfaces that are refined by the other modules. It also provides support for the two interaction styles (notification-based and wait-based) with the middleware.
- **Domain module** (p. 41) contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of the Service and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up the Service.
- **Topic module** (p. 54) contains the **com.rti.dds.topic.Topic** (p. 1807) class, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by the application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.
- **Publication module** (p. 69) contains the **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.
- **Subscription module** (p. 78) contains the **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.subscription.ReadCondition** (p. 1514), and **com.rti.dds.subscription.QueryCondition** (p. 1510) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1755) and **com.rti.dds.subscription.DataReaderListener** (p. 497) interfaces, and more generally, all that is needed on the subscription side.

6.69 Additional RTI Connext Communication Patterns

Extensions to the RTI Connext publish-subscribe functionality.

Modules

- **Request-Reply Pattern**

  Support for the request-reply communication pattern.
- **Infrastructure**

  Infrastructure types for RTI Connext Messaging.
- **Utilities**

  Utilities for the RTI Connext Messaging module.

6.69.1 Detailed Description

Extensions to the RTI Connext publish-subscribe functionality.

This section includes the **Request-Reply API** (p. 111).

6.70 Programming How-To's

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

Modules

- **Publication Example**
  A data publication example.
- **Subscription Example**
  A data subscription example.
- **Participant Use Cases**
  Working with domain participants.
- **Topic Use Cases**
  Working with topics.
- **FlowController Use Cases**
  Working with flow controllers.
- **Publisher Use Cases**
  Working with publishers.
- **DataWriter Use Cases**
  Working with data writers.
- **Subscriber Use Cases**
  Working with subscribers.
- **DataReader Use Cases**
  Working with data readers.
- **Entity Use Cases**
  Working with entities.
- **Waitset Use Cases**
  Using wait-sets and conditions.
- **Transport Use Cases**
  Working with pluggable transports.
- **Filter Use Cases**
  Working with data filters.
- **Creating Custom Content Filters**
  Working with custom content filters.
- **Large Data Use Cases**
  Working with large data types.
- **Request-Reply Examples**
  Examples on how to use the request-reply API .

6.70.1 Detailed Description

These "How To"s illustrate how to apply RTI Connext APIs to common use cases.

These are a good starting point to familiarize yourself with DDS. You can use these code fragments as "templates" for writing your own code.

6.71 Participant Built-in Topics

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

Classes

- class **ParticipantBuiltinTopicData**

  *Entry created when a **DomainParticipant** (p. 670) object is discovered.*
- class **ParticipantBuiltinTopicDataDataReader**

  Instantiates `DataReader` < ***com.rti.dds.domain.builtin.ParticipantBuiltinTopicData*** *(p. 1349)* > .
- class **ParticipantBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < ***com.rti.dds.domain.builtin.ParticipantBuiltinTopicData*** *(p. 1349)* > .
- class **ParticipantBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` < ***com.rti.dds.domain.builtin.ParticipantBuiltinTopicData*** *(p. 1349)* > .

Variables

- static final String **PARTICIPANT_TOPIC_NAME** = DDS_PARTICIPANT_TOPIC_NAME()

  Participant topic name.

6.71.1 Detailed Description

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

6.71.2 Variable Documentation

6.71.2.1 PARTICIPANT_TOPIC_NAME

```
final String PARTICIPANT_TOPIC_NAME = DDS_PARTICIPANT_TOPIC_NAME()  [static]
```

Participant topic name.

Topic name of **builtin.ParticipantBuiltinTopicDataDataReader** (p. 1354)

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

builtin.ParticipantBuiltinTopicDataDataReader (p. 1354)

6.72 Publication Built-in Topics

Builtin topic for accessing information about the Publications discovered by RTI Connext.

Classes

- class **PublicationBuiltinTopicData**

  *Entry created when a **com.rti.dds.publication.DataWriter** (p. 553) is discovered in association with its **Publisher** (p. 1466).*
- class **PublicationBuiltinTopicDataDataReader**

  Instantiates `DataReader` *<* ***com.rti.dds.publication.builtin.PublicationBuiltinTopicData*** *(p. 1452) >.*
- class **PublicationBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *<* ***com.rti.dds.publication.builtin.PublicationBuiltinTopicData*** *(p. 1452) >.*
- class **PublicationBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` *<* ***com.rti.dds.publication.builtin.PublicationBuiltinTopicData*** *(p. 1452) >.*

Variables

- static final String **PUBLICATION_TOPIC_NAME** = DDS_PUBLICATION_TOPIC_NAME()

  Publication topic name.

6.72.1 Detailed Description

Builtin topic for accessing information about the Publications discovered by RTI Connext.

6.72.2 Variable Documentation

6.72.2.1 PUBLICATION_TOPIC_NAME

```
final String PUBLICATION_TOPIC_NAME = DDS_PUBLICATION_TOPIC_NAME()  [static]
```

Publication topic name.

Topic name of **builtin.PublicationBuiltinTopicDataDataReader** (p. 1461)

See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

builtin.PublicationBuiltinTopicDataDataReader (p. 1461)

6.73 Subscription Built-in Topics

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

Classes

- class **SubscriptionBuiltinTopicData**

  *Entry created when a **com.rti.dds.subscription.DataReader** (p. 450) is discovered in association with its **Subscriber** (p. 1730).*
- class **SubscriptionBuiltinTopicDataDataReader**

  *Instantiates **DataReader** (p. 450) < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) > .*
- class **SubscriptionBuiltinTopicDataSeq**

  *Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) > .*
- class **SubscriptionBuiltinTopicDataTypeSupport**

  *Instantiates TypeSupport < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) > .*

Variables

- static final String **SUBSCRIPTION_TOPIC_NAME** = DDS_SUBSCRIPTION_TOPIC_NAME()

  Subscription topic name.

6.73.1 Detailed Description

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

6.73.2 Variable Documentation

6.73.2.1 SUBSCRIPTION_TOPIC_NAME

```
final String SUBSCRIPTION_TOPIC_NAME = DDS_SUBSCRIPTION_TOPIC_NAME()  [static]
```

Subscription topic name.

Topic name of **builtin.SubscriptionBuiltinTopicDataDataReader** (p. 1771)

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

builtin.SubscriptionBuiltinTopicDataDataReader (p. 1771)

6.74 Topic Built-in Topics

Builtin topic for accessing information about the Topics discovered by RTI Connext.

Classes

- class **TopicBuiltinTopicData**

  *Entry created when a **Topic** (p. 1807) object discovered.*
- class **TopicBuiltinTopicDataDataReader**

  Instantiates `DataReader` < ***builtin.TopicBuiltinTopicData*** *(p. 1813)* > .
- class **TopicBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < ***builtin.TopicBuiltin↩ TopicData*** *(p. 1813)* > .
- class **TopicBuiltinTopicDataTypeSupport**

  Instantiates ***`TypeSupport`*** *(p. 1941)* < ***builtin.TopicBuiltinTopicData*** *(p. 1813)* > .

Variables

- static final String **TOPIC_TOPIC_NAME** = DDS_TOPIC_TOPIC_NAME()

  Topic *(p. 1807) topic name.*

6.74.1 Detailed Description

Builtin topic for accessing information about the Topics discovered by RTI Connext.

6.74.2 Variable Documentation

6.74.2.1 TOPIC_TOPIC_NAME

```
final String TOPIC_TOPIC_NAME = DDS_TOPIC_TOPIC_NAME()  [static]
```

Topic (p. 1807) topic name.

Topic (p. 1807) name of **builtin.TopicBuiltinTopicDataDataReader** (p. 1817)

See also

builtin.TopicBuiltinTopicData (p. 1813)

builtin.TopicBuiltinTopicDataDataReader (p. 1817)

6.75 ServiceRequest Built-in Topic

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

Classes

- class **ServiceRequest**

  A request coming from one of the built-in services.
- class **ServiceRequestDataReader**

  Instantiates `DataReader` < ***com.rti.dds.topic.builtin.ServiceRequest*** *(p. 1675)* > .
- class **ServiceRequestSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < ***com.rti.dds.topic.builtin.ServiceRequest*** *(p. 1675)* > .
- class **ServiceRequestTypeSupport**

  Instantiates ***`TypeSupport`*** *(p. 1941)* < ***com.rti.dds.topic.builtin.ServiceRequest*** *(p. 1675)* > .

Variables

- static final int **UNKNOWN_SERVICE_ID**

  An invalid Service Id.
- static final int **TOPIC_QUERY_SERVICE_ID**

  Service Id for the ***com.rti.dds.subscription.TopicQuery*** *(p. 1830) Service.*
- static final int **LOCATOR_REACHABILITY_SERVICE_ID**

  Service Id for the Locator Reachability Service.
- static final int **INSTANCE_STATE_SERVICE_ID**

  Service Id for the Instance State Request service.
- static final int **MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Command Service Request.
- static final int **MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Reply Service Requests.
- static final String **SERVICE_REQUEST_TOPIC_NAME** = DDS_SERVICE_REQUEST_TOPIC_NAME()

  Service Request topic name.

6.75.1 Detailed Description

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

Currently, the **com.rti.dds.subscription.TopicQuery** (p. 1830), Locator Reachability Instance State Consistency and Controlability (part of Observability) all rely on this topic.

See also

Topic Queries (p. 84) for an explanation of how TopicQueries use ServiceRequests and how you can access the ServiceRequests for debugging purposes in the section **The Built-in ServiceRequest DataReader** (p. 86).

6.75.2 Variable Documentation

6.75.2.1 UNKNOWN_SERVICE_ID

```
final int UNKNOWN_SERVICE_ID  [static]
```

An invalid Service Id.

6.75.2.2 TOPIC_QUERY_SERVICE_ID

```
final int TOPIC_QUERY_SERVICE_ID  [static]
```

Service Id for the **com.rti.dds.subscription.TopicQuery** (p. 1830) Service.

6.75.2.3 LOCATOR_REACHABILITY_SERVICE_ID

```
final int LOCATOR_REACHABILITY_SERVICE_ID  [static]
```

Service Id for the Locator Reachability Service.

6.75.2.4 INSTANCE_STATE_SERVICE_ID

```
final int INSTANCE_STATE_SERVICE_ID  [static]
```

Service Id for the Instance State Request service.

6.75.2.5 MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID

```
final int MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID  [static]
```

Service Id for RTI Monitoring Library 2.0 Command Service Request.

RTI Monitoring Library 2.0 remote commands are sent using this service.

6.75.2.6 MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID

```
final int MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID  [static]
```

Service Id for RTI Monitoring Library 2.0 Reply Service Requests.

Replies to RTI Monitoring Library 2.0 remote commands are sent using this service.

6.75.2.7 SERVICE_REQUEST_TOPIC_NAME

```
final String SERVICE_REQUEST_TOPIC_NAME = DDS_SERVICE_REQUEST_TOPIC_NAME()  [static]
```

Service Request topic name.

Topic (p. 1807) name of the **builtin.ServiceRequestDataReader** (p. 1680)

See also

com.rti.dds.topic.builtin.ServiceRequest (p. 1675)

builtin.ServiceRequestDataReader (p. 1680)

6.76 Common types and functions

Types and functions related to the built-in topics.

Classes

- class **ContentFilterProperty_t**

  *<<**extension**>> (p. 155) Type used to provide all the required information to enable content filtering.*
- class **Locator_t**

  *<<**extension**>> (p. 155) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*
- class **LocatorSeq**

  *Declares IDL `sequence` < **com.rti.dds.infrastructure.Locator_t** (p. 1253) >*
- class **ProductVersion_t**

  *<<**extension**>> (p. 155) Type used to represent the current version of RTI Connext.*
- class **ProtocolVersion_t**

  *<<**extension**>> (p. 155) Type used to represent the version of the RTPS protocol.*
- class **TransportInfo_t**

  Contains the class_id and message_size_max of an installed transport.
- class **TransportInfoSeq**

  *Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.↩ TransportInfo_t** (p. 1845) > .*
- class **VendorId_t**

  *<<**extension**>> (p. 155) Type used to represent the vendor of the service implementing the RTPS protocol.*
- class **BuiltinTopicKey_t**

  The key type of the built-in topic types.

6.76.1 Detailed Description

Types and functions related to the built-in topics.

6.77 ASYNCHRONOUS_PUBLISHER

<<***extension***>> *(p. 155)* Specifies the asynchronous publishing settings of the **com.rti.dds.publication.Publisher** (p. 1466) instances.

Classes

- class **AsynchronousPublisherQosPolicy**

  Configures the mechanism that sends user data in an external middleware thread.

Variables

- static final **QosPolicyId_t ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy*** *(p. 339)*

6.77.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies the asynchronous publishing settings of the **com.rti.dds.publication.Publisher** (p. 1466) instances.

6.77.2 Variable Documentation

6.77.2.1 ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID

```
final  QosPolicyId_t ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy** (p. 339)

6.78 AVAILABILITY

<<***extension***>> *(p. 155)* Configures the availability of data.

Classes

- class **AvailabilityQosPolicy**

  Configures the availability of data.
- class **EndpointGroup_t**

  Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.
- class **EndpointGroupSeq**

  *A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022).*

Variables

- static final **QosPolicyId_t AVAILABILITY_QOS_POLICY_ID**

  <<***extension***>> (p. 155) Identifier for **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343)

6.78.1 Detailed Description

<<***extension***>> (p. 155) Configures the availability of data.

6.78.2 Variable Documentation

6.78.2.1 AVAILABILITY_QOS_POLICY_ID

```
final  QosPolicyId_t AVAILABILITY_QOS_POLICY_ID  [static]
```

<<***extension***>> (p. 155) Identifier for **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343)

6.79 BATCH

<<***extension***>> (p. 155) Batch QoS policy used to enable batching in **com.rti.dds.publication.DataWriter** (p. 553) instances.

Classes

- class **BatchQosPolicy**

  Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

Variables

- static final **QosPolicyId_t BATCH_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.BatchQosPolicy*** *(p. 355)*

6.79.1 Detailed Description

<<***extension***>> *(p. 155)* Batch QoS policy used to enable batching in **com.rti.dds.publication.DataWriter** (p. 553) instances.

6.79.2 Variable Documentation

6.79.2.1 BATCH_QOS_POLICY_ID

```
final  QosPolicyId_t BATCH_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.BatchQosPolicy** (p. 355)

6.80 Builtin Qos Profiles

<<***extension***>> *(p. 155)* QoS libraries, profiles, and snippets that are automatically built into RTI Connext.

Classes

- class **BuiltinQosProfiles**

  The available built-in QoS libraries, profiles, and snippets.

Variables

- static final String **BUILTIN_QOS_LIB**

  A library of non-experimental QoS profiles.
- static final String **PROFILE_BASELINE_ROOT**

  The root baseline QoS values from which all other Baseline.X.X.X profiles inherit.
- static final String **PROFILE_BASELINE**

  The most up-to-date QoS default values.
- static final String **PROFILE_BASELINE_5_0_0**

  The QoS default values for version 5.0.0.
- static final String **PROFILE_BASELINE_5_1_0**

  The QoS default values for version 5.1.0.
- static final String **PROFILE_BASELINE_5_2_0**

  The QoS default values for version 5.2.0.
- static final String **PROFILE_BASELINE_5_3_0**

  The QoS default values for version 5.3.0.
- static final String **PROFILE_BASELINE_6_0_0**

  The QoS default values for version 6.0.0.
- static final String **PROFILE_BASELINE_6_1_0**

  The QoS default values for version 6.1.0.
- static final String **PROFILE_BASELINE_7_0_0**

  The QoS default values for version 7.0.0.
- static final String **PROFILE_BASELINE_7_1_0**

  The QoS default values for version 7.1.0.
- static final String **PROFILE_GENERIC_COMMON**

  A common Participant base profile.
- static final String **PROFILE_GENERIC_MONITORING_COMMON**

  Enables RTI Monitoring Library.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY**

  Sets the values necessary to communicate with RTI Connext Micro.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.
- static final String **PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY**

  Sets the values necessary to interoperate with other DDS vendors.
- static final String **PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY**

  Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.
- static final String **PROFILE_GENERIC_SECURITY**

  Loads the DDS Secure builtin plugins.
- static final String **BUILTIN_QOS_LIB_EXP**

  A library of experimental QoS profiles.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE**

  Enables strict reliability.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE**

  Enables keep-last reliability.
- static final String **PROFILE_GENERIC_BEST_EFFORT**

Enables best-effort reliability kind.

- static final String **PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT**

  A profile that can be used to achieve high throughput.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY**

  A profile that can be used to achieve low latency.
- static final String **PROFILE_GENERIC_PARTICIPANT_LARGE_DATA**

  A common Participant base profile to facilitate sending large data.
- static final String **PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING**

  Configures Participants for large data and monitoring.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with strict reliability.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with keep-last reliability.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures strictly reliable communication for large data with a fast flow controller.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures strictly reliable communication for large data with a medium flow controller.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures strictly reliable communication for large data with a slow flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures keep-last reliable communication for large data with a fast flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures keep-last reliable communication for large data with a medium flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures keep-last reliable communication for large data with a slow flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL**

  Persists the samples of a DataWriter as long as the entity exists.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT**

  Persists samples using RTI Persistence Service.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT**

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- static final String **PROFILE_GENERIC_AUTO_TUNING**

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- static final String **PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT**

  Uses a set of QoS which reduces the memory footprint of the application.
- static final String **PROFILE_GENERIC_MONITORING2**

  The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.
- static final String **PROFILE_PATTERN_PERIODIC_DATA**

  Used for applications that expect periodic data.
- static final String **PROFILE_PATTERN_STREAMING**

  Used for applications that stream data.
- static final String **PROFILE_PATTERN_RELIABLE_STREAMING**

  Used for applications that stream data and require reliable communication.
- static final String **PROFILE_PATTERN_EVENT**

  Used for applications that handle events.
- static final String **PROFILE_PATTERN_ALARM_EVENT**

  Used for applications that handle alarm events.

- static final String **PROFILE_PATTERN_STATUS**

  Used for applications whose samples represent statuses.
- static final String **PROFILE_PATTERN_ALARM_STATUS**

  Used for applications in which samples represent alarm statuses.
- static final String **PROFILE_PATTERN_LAST_VALUE_CACHE**

  Used for applications that only need the last published value.
- static final String **BUILTIN_QOS_SNIPPET_LIB**

  A library of QoS Snippets.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON**

  QoS Snippet that configures the reliability protocol with a common configuration.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL**

  QoS Snippet that configures the reliability protocol for KEEP_ALL.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST**

  QoS Snippet that configures the reliability protocol for KEEP_LAST.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE**

  QoS Snippet that configures the reliability protocol for sending data at a high rate.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY**

  QoS Snippet that configures the reliability protocol for sending data at low latency.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA**

  QoS Snippet that configures the reliability protocol for large data.
- static final String **SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC**

  Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_COMMON**

  QoS Snippet that optimizes discovery with a common configuration.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT**

  QoS Snippet that optimizes the Participant QoS to send less discovery information.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST**

  QoS Snippet that optimizes the Endpoint Discovery to be faster.
- static final String **SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS**

  QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.
- static final String **SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE**

  QoS Snippet that sets RELIABILITY QoS to RELIABLE.
- static final String **SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT**

  QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.
- static final String **SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1**

  QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.
- static final String **SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL**

  *QoS Snippet that sets HISTORY **QosPolicy** (p. 1501) to KEEP_ALL kind.*
- static final String **SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS**

  *QoS Snippet that sets PUBLISH_MODE **QosPolicy** (p. 1501) to ASYNCHRONOUS kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL**

  *QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to TRANSIENT_LOCAL kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT**

  *QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to TRANSIENT kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT**

  *QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to PERSISTENT kind.*
- static final String **SNIPPET_QOS_POLICY_BATCHING_ENABLE**

*QoS Snippet that sets BATCH **QosPolicy** (p. 1501) to true.*

- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS**

  QoS Snippet that configures and set a FlowController of 838 Mbps.
- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS**

  QoS Snippet that configures and sets a FlowController of 209 Mbps.
- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS**

  QoS Snippet that configures and sets a FlowController of 52 Mbps.
- static final String **SNIPPET_FEATURE_AUTO_TUNING_ENABLE**

  QoS Snippet that enables auto_throttle and turbo_mode to true.
- static final String **SNIPPET_FEATURE_MONITORING_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library.
- static final String **SNIPPET_FEATURE_MONITORING2_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library 2.0.
- static final String **SNIPPET_FEATURE_SECURITY_ENABLE**

  QoS Snippet that enables security using the Builtin Security Plugins.
- static final String **SNIPPET_FEATURE_TOPIC_QUERY_ENABLE**

  QoS Snippet that enables Topic Query.
- static final String **SNIPPET_TRANSPORT_TCP_LAN_CLIENT** = "Transport.TCP.LAN.Client"

  QoS Snippet that configures a TCP LAN Client over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT**

  QoS Snippet that configures a symmetric WAN TCP Client over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER**

  QoS Snippet that an asymmetric WAN TCP Server over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT**

  QoS Snippet that configures an asymmetric WAN TCP Client over DDS.
- static final String **SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION**

  QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.
- static final String **SNIPPET_TRANSPORT_UDP_WAN**

  QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).
- static final String **SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3**

  QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.
- static final String **SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.
- static final String **SNIPPET_5_1_0_TRANSPORT_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

6.80.1 Detailed Description

<<***extension***>> *(p. 155)* QoS libraries, profiles, and snippets that are automatically built into RTI Connext.

The built-in profiles can be accessed in QoS XML configuration files and by using any of the APIs that accept library and profiles names by using the constants or string versions as documented on this page.

The built-in profiles are provided as a way to quickly and easily configure RTI Connext applications with a set of QoS values aimed at achieving a specific behavior.

There are three built-in QoS libraries:

- BuiltinQosLib: A library containing built-in QoS Profiles.

- BuiltinQosLibExp: A library containing experimental QoS Profiles. Experimental QoS Profiles are new QoS Profiles that have been tested internally but have not gone through an extensive validation period. Therefore, some of the settings may change in future releases based on customer and internal feedback. After validation, experimental QoS Profiles will be moved into the non-experimental library.

- BuiltinQosSnippetLib: A library containing QoS Snippets that are ready to use as elements for the QoS Profile composition pattern. For further information about this pattern visit the following article: https://community.rti.com/best-practices/qos-profile-inheritance-and-composition-guidance

There are three types of profiles:

- Baseline.X.X.X: These profiles represent the QoS defaults for /ndds version X.X.X. To access the defaults for the latest RTI Connext version, use the BuiltinQosLib::Baseline profile.
- Generic.X: These profiles allow you to easily configure different features and communication use-cases with RTI Connext. For example, there is a Generic.StrictReliable profile for use when your application has a requirement for no data loss.
- Pattern.X: These profiles inherit from the Generic.X profiles and allow you to configure various domain-specific communication use cases. For example, there is a Pattern.Alarm profile that can be used to manage the generation and consumption of alarm events.

There are several types of QoS Snippets. These are the current QoS Snippets available:

- Optimization.X: these QoS Snippets optimize one or more parameters related to the X QoS Policy or a specific use-case.
- QosPolicy.X.Y: these QoS Snippets set a specific QoS Policy X to the value Y.
- Feature.X: these QoS Snippets set all the needed QoS values to enable/modify a specific feature.
- Transport.X: these QoS Snippets set a specific transport defined by X. This transport may have specific scenarios that are also specified in the name.
- Compatibility.X: these QoS Snippets change the specific QoS policies to ensure compatibility with specific products or versions specified by X.

These QoS Profiles can be used as base profiles in XML configuration, then QoS Snippets can modify specific aspects of this base QoS Profile, and finally files and values can be modified to fit a specific system's needs. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.Common">

   <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

     <base_name>

       <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

     </base_name>

     <!-- Override and add values -->

   </domain_participant_qos>

</qos_profile>
```

The QoS Profiles can also be used in any APIs that call for a QoS library and profile name, for example, the create_*↩
_with_profile() APIs. To create a **com.rti.dds.publication.DataWriter** (p. 553) configured to send large data:

```
writer = DDS_DomainParticipant_create_datawriter_with_profile(

                participant, topic,

                com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB,

      com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW,

                NULL, com.rti.dds.infrastructure.StatusKind.STATUS_MASK_NONE);
```

All the built-in QoS Profiles and QoS Snippets are documented in the BaselineRoot.documentationONLY.xml and BuiltinProfiles.documentationONLY.xml files that are included in the NDDSHOME/xml directory of the RTI Connext installation.

- BaselineRoot.documentationONLY.xml contains the root baseline QoS profile that corresponds to the default values of RTI Connext 5.0.0.

- BuiltinProfiles.documentationONLY.xml contains the rest of the built-in **Qos** (p. 1500) Profiles and QoS Snippets.

6.80.2 Variable Documentation

6.80.2.1 BUILTIN_QOS_LIB

```
final String BUILTIN_QOS_LIB  [static]
```

A library of non-experimental QoS profiles.

String-version: "BuiltinQosLib"

6.80.2.2 PROFILE_BASELINE_ROOT

```
final String PROFILE_BASELINE_ROOT  [static]
```

The root baseline QoS values from which all other Baseline.X.X.X profiles inherit.

This profile contains the root baseline QoS values from which all other Baseline.X.X.X profiles inherit. These values correspond to the default values for RTI Connext 5.0.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.Root"

6.80.2.3 PROFILE_BASELINE

```
final String PROFILE_BASELINE  [static]
```

The most up-to-date QoS default values.

You can use this profile if you want your application to pick up and use any new QoS default settings each time a new RTI Connext version is released – without changing your application code.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline"

6.80.2.4 PROFILE_BASELINE_5_0_0

```
final String PROFILE_BASELINE_5_0_0  [static]
```

The QoS default values for version 5.0.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.5.0.0"

6.80.2.5 PROFILE_BASELINE_5_1_0

```
final String PROFILE_BASELINE_5_1_0  [static]
```

The QoS default values for version 5.1.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.5.1.0"

6.80.2.6 PROFILE_BASELINE_5_2_0

```
final String PROFILE_BASELINE_5_2_0  [static]
```

The QoS default values for version 5.2.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.5.2.0"

6.80.2.7 PROFILE_BASELINE_5_3_0

```
final String PROFILE_BASELINE_5_3_0  [static]
```

The QoS default values for version 5.3.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.5.3.0"

6.80.2.8 PROFILE_BASELINE_6_0_0

```
final String PROFILE_BASELINE_6_0_0  [static]
```

The QoS default values for version 6.0.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.6.0.0"

6.80.2.9 PROFILE_BASELINE_6_1_0

```
final String PROFILE_BASELINE_6_1_0  [static]
```

The QoS default values for version 6.1.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.6.1.0"

6.80.2.10 PROFILE_BASELINE_7_0_0

```
final String PROFILE_BASELINE_7_0_0  [static]
```

The QoS default values for version 7.0.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.7.0.0"

6.80.2.11 PROFILE_BASELINE_7_1_0

```
final String PROFILE_BASELINE_7_1_0  [static]
```

The QoS default values for version 7.1.0.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Baseline.7.1.0"

6.80.2.12 PROFILE_GENERIC_COMMON

```
final String PROFILE_GENERIC_COMMON  [static]
```

A common Participant base profile.

All Generic.X and Pattern.X profiles inherit from this profile.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.Common"

6.80.2.13 PROFILE_GENERIC_MONITORING_COMMON

```
final String PROFILE_GENERIC_MONITORING_COMMON  [static]
```

Enables RTI Monitoring Library.

Generic Base participant QoS Profile that enables RTI Monitoring Library.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Monitoring.Common", apply the QoS Snippet "BuiltinQos↩ SnippetLib::Feature.Monitoring.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Monitoring.Common">

</qos_profile>
```

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.Monitoring.Common"

6.80.2.14 PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY

```
final String PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY  [static]
```

Sets the values necessary to communicate with RTI Connext Micro.

This profile will always represent the QoS values required for interoperability between the most recent version of RTI Connext Micro at the time of release of the most recent version of RTI Connext.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.ConnextMicroCompatibility"

6.80.2.15 PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9

```
final String PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9 [static]
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.

At the time of the release of RTI Connext 5.3.0 it was not necessary to set any QoS values in order to interoperate with RTI Connext Micro. This applies to RTI Connext Micro versions 2.4.4 and later. The most recent version of RTI Connext Micro at the time of release of RTI Connext 5.3.0 was 2.4.9. There is no guarantee that this profile will interoperate with versions of RTI Connext Micro after 2.4.9.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.ConnextMicroCompatibility.2.4.9"

6.80.2.16 PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3

```
final String PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3 [static]
```

Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.

RTI Connext Micro versions 2.4.3 and earlier only supported the com.rti.dds.infrastructure.LivelinessQosPolicyKind.↩ LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS LivelinessQos kind. In order to be compatible with these versions of RTI Connext Micro, the **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.↩ publication.DataWriter** (p. 553) must have their Liveliness kind changed to this value because the default kind in RTI Connext is com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_↩ QOS.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.ConnextMicroCompatibility.2.4.3"

6.80.2.17 PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY

```
final String PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY [static]
```

Sets the values necessary to interoperate with other DDS vendors.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.OtherDDSVendorCompatibility"

6.80.2.18 PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY

```
final String PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY [static]
```

Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.510TransportCompatibility"

6.80.2.19 PROFILE_GENERIC_SECURITY

```
final String PROFILE_GENERIC_SECURITY  [static]
```

Loads the DDS Secure builtin plugins.

Generic Base Participant Profile that enables the builtin DDS Security Plugins.

Use of this QoS Profile is deprecated. It is included for backwards compatibility.

Instead of using the (deprecated) "BuiltinQosLib::Generic.Security", apply the QoS Snippet "BuiltinQosSnippetLib::↩ Feature.Security.Enable" via composition. For example:

```
<qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosSnippetLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

Legacy applications may use this QoS Profile via inheritance. To do this, create a derived QoS Profile that inherits from this one. For example:

```
<qos_profile name="MyProfile" base_name="BuiltinQosLib::Generic.StrictReliable">

    <domain_participant_qos base_name="BuiltinQosLib::Generic.Security">

</qos_profile>
```

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177)

String-version: "Generic.Security"

6.80.2.20 BUILTIN_QOS_LIB_EXP

```
final String BUILTIN_QOS_LIB_EXP  [static]
```

A library of experimental QoS profiles.

Experimental profiles are new profiles that have been tested internally but have not gone through an extensive validation period. Therefore some of the settings may change in future releases based on customer and internal feedback. After validation, experimental profiles will be moved into the non-experimental library.

QoS Profiles in this library are deprecated. They have been moved to "BuiltinQosLib". You should use the QoS Profiles from "BuiltinQosLib" instead of the ones in "BuiltinQosLibExp". The experimental profiles are still defined here to avoid backward compatibility issues.

String-version: "BuiltinQosLibExp"

6.80.2.21 PROFILE_GENERIC_STRICT_RELIABLE

```
final String PROFILE_GENERIC_STRICT_RELIABLE  [static]
```

Enables strict reliability.

Configures communication to be "strict reliable" where every sample is reliably delivered.

Combines the use of the **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) for **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) with a com.rti.dds.infrastructure.HistoryQos↩ PolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS for the **com.rti.dds.infrastructure.HistoryQosPolicy↩ Kind** (p. 1147).

This QoS Profile also optimizes the reliability protocol setting for this configuration.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable"

6.80.2.22 PROFILE_GENERIC_KEEP_LAST_RELIABLE

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE  [static]
```

Enables keep-last reliability.

Like the Generic.StrictReliable profile, this profile ensures in-order delivery of samples. However, new data can overwrite data that has not yet been acknowledged by the reader, therefore causing possible sample loss.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable"

6.80.2.23 PROFILE_GENERIC_BEST_EFFORT

```
final String PROFILE_GENERIC_BEST_EFFORT  [static]
```

Enables best-effort reliability kind.

This profile enables best-effort communication. No effort or resources are spent to track whether or not sent samples are received. Minimal resources are used. This is the most deterministic method of sending data since there is no indeterministic delay that can be introduced by resending data. Data samples may be lost. This setting is good for periodic data.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.BestEffort"

6.80.2.24 PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT

```
final String PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT [static]
```

A profile that can be used to achieve high throughput.

This QoS Profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_↩ RELIABLE** (p. 182) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_RELIABLE** (p. 182) QoS Profile..

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.HighThroughput"

6.80.2.25 PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY

```
final String PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY [static]
```

A profile that can be used to achieve low latency.

This QoS Profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_↩ RELIABLE** (p. 182) QoS Profile to perform additional, fine-grained performance tuning specific to applications that send continuously streaming data. The parameters specified here add to and/or override the parameters specified in the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_RELIABLE** (p. 182) QoS Profile.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.LowLatency"

6.80.2.26 PROFILE_GENERIC_PARTICIPANT_LARGE_DATA

```
final String PROFILE_GENERIC_PARTICIPANT_LARGE_DATA [static]
```

A common Participant base profile to facilitate sending large data.

This is a common Participant base QoS Profile that configures 3 different flow controllers: 838, 209, and 52 Mbps that can each be used to throttle application data flow at different rates.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.Participant.LargeData"

6.80.2.27 PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING

```
final String PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING [static]
```

Configures Participants for large data *and* monitoring.

This is a common base Participant QoS Profile to configure Participants to both handle large data and use RTI Monitoring Library.

This QoS Profile is deprecated. It is included for backwards compatibility.

It is recommended that instead of inheriting from this QoS Profile, new applications apply the following QoS Snippets to their application-specific QoS Profiles:

Enable Monitoring

- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_FEATURE_MONITORING_ENABLE** (p. 204)

Enable Large Data + optimizations

- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS** (p. 200)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_↩ LARGE_DATA** (p. 195)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_↩ DYNAMICMEMALLOC** (p. 196) In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.Participant.LargeData.Monitoring"

See also

com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_PARTICIPANT_LARGE_DATA (p. 184)

com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_MONITORING_COMMON (p. 180)

6.80.2.28 PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA

```
final String PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA [static]
```

Configures endpoints for sending large data with strict reliability.

This QoS Profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_PARTICIPANT_↩ LARGE_DATA** (p. 184) QoS Profile to handle sending large samples. This QoS Profile optimizes memory usage per sample within RTI Connext, but it does not do any flow control. You can use this QoS Profile directly, which enables asynchronous publication with the default flow controller (i.e. no flow control) or you can use one of the three QoS Profiles below (Generic.StrictReliable.LargeData.∗Flow), which uses this QoS Profile as a common base QoS Profile. Each of these three QoS Profiles uses one of the three flow controllers defined in **com.rti.dds.infrastructure.Builtin↩ QosProfiles.PROFILE_GENERIC_PARTICIPANT_LARGE_DATA** (p. 184) in order to throttle application data flow.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.LargeData"

6.80.2.29 PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA [static]
```

Configures endpoints for sending large data with keep-last reliability.

This QoS Profile is similar to the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_↩ PARTICIPANT_LARGE_DATA** (p. 184) QoS Profile, but also adds QoS Snippets to handle sending large data. You can use this QoS Profile directly, which enables the default flow controller (i.e., no flow control) or you can choose one of the three QoS Profiles below (Generic.KeepLastReliable.LargeData.∗Flow) which uses this QoS Profile as a common base QoS Profile. Each of these three QoS Profiles uses one of the three flow controllers defined in **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_PARTICIPANT_LARGE_DATA** (p. 184) in order to throttle application data flow.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.LargeData"

6.80.2.30 PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW

```
final String PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW [static]
```

Configures strictly reliable communication for large data with a fast flow controller.

Strictly reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.LargeData.FastFlow"

6.80.2.31 PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW

```
final String PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW [static]
```

Configures strictly reliable communication for large data with a medium flow controller.

Strictly reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.LargeData.MediumFlow"

6.80.2.32 PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW

```
final String PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW [static]
```

Configures strictly reliable communication for large data with a slow flow controller.

Strictly reliable communication for large data with a 52 MB/sec ($\sim$ 6.25 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.StrictReliable.LargeData.SlowFlow"

6.80.2.33 PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW [static]
```

Configures keep-last reliable communication for large data with a fast flow controller.

Keep-last reliable communication for large data with a 838 Mbps ($\sim$ 100 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.LargeData.FastFlow"

6.80.2.34 PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW [static]
```

Configures keep-last reliable communication for large data with a medium flow controller.

Keep-last reliable communication for large data with a 209Mbps ($\sim$ 25 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.LargeData.MediumFlow"

6.80.2.35 PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW [static]
```

Configures keep-last reliable communication for large data with a slow flow controller.

Keep-last reliable communication for large data with a 52 MB/sec ($\sim$ 6.25 MB/sec) flow controller.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.LargeData.SlowFlow"

6.80.2.36 PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL [static]
```

Persists the samples of a DataWriter as long as the entity exists.

This profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_KEEP_LAST_↩ RELIABLE** (p. 183) profile, but persists the samples of a **com.rti.dds.publication.DataWriter** (p. 553) as long as the entity exists in order to deliver them to late-joining DataReaders.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.TransientLocal"

6.80.2.37 PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT [static]
```

Persists samples using RTI Persistence Service.

This profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_KEEP_LAST_↩ RELIABLE** (p. 183) profile, but persists samples using Persistence Service in order to deliver them to late-joining DataReaders.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.Transient"

6.80.2.38 PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT

```
final String PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT [static]
```

Persists samples in permanent storage, like a disk, using RTI Persistence Service.

This profile extends the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_KEEP_LAST_↩ RELIABLE** (p. 183) profile, but persists samples in permanent storage, such as a disk, using Persistence Service in order to deliver them to late-joining DataReaders.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.KeepLastReliable.Persistent"

6.80.2.39 PROFILE_GENERIC_AUTO_TUNING

```
final String PROFILE_GENERIC_AUTO_TUNING  [static]
```

Enables the Turbo Mode batching and Auto Throttle experimental features.

Turbo Mode batching adjusts the maximum number of bytes of a batch based on how frequently samples are being written. Auto Throttle auto-adjusts the speed at which a writer will write samples, based on the number of unacknowledged samples in its queue.

These features are designed to auto-adjust the publishing behavior within a system in order to achieve the best possible performance with regards to throughput and latency.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.AutoTuning"

6.80.2.40 PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT

```
final String PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT  [static]
```

Uses a set of QoS which reduces the memory footprint of the application.

Uses a set of QoS which reduces the memory footprint of the application.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Generic.MinimalMemoryFootprint"

6.80.2.41 PROFILE_GENERIC_MONITORING2

```
final String PROFILE_GENERIC_MONITORING2  [static]
```

The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.

This profile inherits from **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_MINIMAL_MEMORY↩ _FOOTPRINT** (p. 189).

The following DomainParticipant QoS policy of this profile cannot be modified and it is overwritten by the RTI Monitoring Library 2.0:

- Discovery Config Built-in Channel Kind.

The following DataWriter and DataReader QoS policies of this profile cannot be modified and they are overwritten by the RTI Monitoring Library 2.0:

- Reliability Kind.
- Durability Kind.
- History Kind.
- Publish Mode Kind.
- Protocol -> RTPS Reliable Writer -> Max Heartbeat Retries.

This profile uses Topic Filters to select the DataWriter and DataReader QoS depending on the Observability Distribution Topic.

You should be using this profile or a profile inheriting from it when you configure the distribution of telemetry data using the <distribution_settings> tag under <monitoring> for the <participant_factory_qos>.

Note: This profile does not enable the use of the RTI Monitoring Library 2.0. To do that you can use the Snippet **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_FEATURE_MONITORING2_ENABLE** (p. 204).

String-version: "Generic.Monitoring2"

See also

com.rti.ndds.utility.MONITORING_PERIODIC_TOPIC_NAME

com.rti.ndds.utility.MONITORING_EVENT_TOPIC_NAME

com.rti.ndds.utility.MONITORING_LOGGING_TOPIC_NAME

6.80.2.42 PROFILE_PATTERN_PERIODIC_DATA

```
final String PROFILE_PATTERN_PERIODIC_DATA [static]
```

Used for applications that expect periodic data.

This QoS Profile is intended to be used for applications that expect periodic data such as sensor data. The deadline that is set in this profile can be used to detect when DataWriters are not publishing data with the expected periodicity.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.PeriodicData"

6.80.2.43 PROFILE_PATTERN_STREAMING

```
final String PROFILE_PATTERN_STREAMING [static]
```

Used for applications that stream data.

The data sent in streaming applications is commonly periodic. Therefore this profile simply inherits from the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_PATTERN_PERIODIC_DATA** (p. 190) profile. Note: With this QoS Profile, the application may lose data, which may be acceptable in use cases such as video conferencing.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.Streaming"

6.80.2.44 PROFILE_PATTERN_RELIABLE_STREAMING

```
final String PROFILE_PATTERN_RELIABLE_STREAMING  [static]
```

Used for applications that stream data *and* require reliable communication.

Sometimes streaming applications require reliable communication while still tolerating some data loss. In this case, we inherit from the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_KEEP_LAST_RELIABLE** (p. 183) QoS Profile and the following QoS Snippets:

- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE** (p. 198)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1** (p. 199)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_↩ KEEP_LAST** (p. 194)

This QoS Snippet also increases the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) to reduce the probability of losing samples.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.ReliableStreaming"

6.80.2.45 PROFILE_PATTERN_EVENT

```
final String PROFILE_PATTERN_EVENT  [static]
```

Used for applications that handle events.

This QoS Profile can be used by applications in which samples represent events such as button pushes or alerts. When events are triggered, the system should almost always do something, meaning that you don't want the system to lose the event. This means that the system requires strictly reliable communication. To enable it, use the following QoS Snippets:

- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE** (p. 198)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL** (p. 199)
- **com.rti.dds.infrastructure.BuiltinQosProfiles.SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_↩ KEEP_ALL** (p. 193)

Since events and alerts are critical and non-periodic data, it is important to detect situations in which communication between a **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) is broken. This is why this profile sets the **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243). If the **com.rti.dds.↩ publication.DataWriter** (p. 553) does not assert its liveliness in a timely manner, the **com.rti.dds.subscription.Data↩ Reader** (p. 450) will report 'loss of liveliness' to the application.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.Event"

6.80.2.46 PROFILE_PATTERN_ALARM_EVENT

```
final String PROFILE_PATTERN_ALARM_EVENT [static]
```

Used for applications that handle alarm events.

An alarm is a type of event; therefore this profile simply inherits from **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_PATTERN_EVENT** (p. 191).

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.AlarmEvent"

6.80.2.47 PROFILE_PATTERN_STATUS

```
final String PROFILE_PATTERN_STATUS [static]
```

Used for applications whose samples represent statuses.

This QoS Profile can be used by applications in which samples represent state variables whose values remain valid as long as they don't explicitly change. State variables typically do not change periodically. State variables and their values should also be available to applications that appear after the value originally changed because it is unreasonable to have to wait until the next change of state, which may be indeterminate.

Whether to use this QoS Profile or **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_PATTERN_PERIODIC↩ _DATA** (p. 190) can often be an application choice. For example, if a DataWriter is publishing temperature sensor data, it could use the **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_PATTERN_PERIODIC_DATA** (p. 190) QoS Profile and publish the data at a fixed rate or it could use the **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_PATTERN_STATUS** (p. 192) QoS Profile and only publish the temperature when it changes more than 1 degree.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.Status"

6.80.2.48 PROFILE_PATTERN_ALARM_STATUS

```
final String PROFILE_PATTERN_ALARM_STATUS [static]
```

Used for applications in which samples represent alarm statuses.

An alarm status is a type of status; therefore this QoS Profile simply inherits from **com.rti.dds.infrastructure.Builtin↩ QosProfiles.PROFILE_PATTERN_STATUS** (p. 192).

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.AlarmStatus"

6.80.2.49 PROFILE_PATTERN_LAST_VALUE_CACHE

```
final String PROFILE_PATTERN_LAST_VALUE_CACHE [static]
```

Used for applications that only need the last published value.

With this QoS Profile, a **com.rti.dds.publication.DataWriter** (p. 553) will keep in its queue the last value that was published for each sample instance. Late-joining DataReaders will get that value when they join the system. This QoS Profile inherits from **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_KEEP_LAST_RELIABLE_↩ TRANSIENT_LOCAL** (p. 187) because the use case requires delivery to late-joiners.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB** (p. 177) and **com.rti.dds.↩ infrastructure.BuiltinQosProfiles.BUILTIN_QOS_LIB_EXP** (p. 182)

String-version: "Pattern.LastValueCache"

6.80.2.50 BUILTIN_QOS_SNIPPET_LIB

```
final String BUILTIN_QOS_SNIPPET_LIB [static]
```

A library of QoS Snippets.

String-version: "BuiltinQosSnippetLib"

6.80.2.51 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON [static]
```

QoS Snippet that configures the reliability protocol with a common configuration.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet defines parameters common to a set of "alternative" QoS Snippets that configure the reliability protocol.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

This QoS Snippet configures the reliability protocol parameters for more aggressive (faster) heartbeats so that sample loss is detected and repaired faster.

This QoS Snippet also sets the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_heartbeat_retries** (p. 1611), which works in combination with the heartbeat rate to determine when the **com.rti.dds.publication.Data↩ Writer** (p. 553) considers a **com.rti.dds.subscription.DataReader** (p. 450) non-responsive. This QoS Snippet configures the **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600) parameters so that the **com.rti.dds.publication.DataWriter** (p. 553) considers the **com.rti.dds.subscription.DataReader** (p. 450) to be "inactive" after 500 unresponded heartbeats.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.Common"

6.80.2.52 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL [static]
```

QoS Snippet that configures the reliability protocol for KEEP_ALL.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **com.rti.dds.publication.DataWriter** (p. 553) reliable protocol parameters for a reliable **com.rti.dds.publication.DataWriter** (p. 553) with **com.rti.dds.infrastructure.HistoryQosPolicyKind** (p. 1147) set to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

Note that this QoS Snippet does not configure the **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) or **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.KeepAll"

6.80.2.53 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST [static]
```

QoS Snippet that configures the reliability protocol for KEEP_LAST.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **com.rti.dds.publication.DataWriter** (p. 553) reliable protocol parameters for a reliable **com.rti.dds.publication.DataWriter** (p. 553) with **com.rti.dds.infrastructure.HistoryQosPolicyKind** (p. 1147) set to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

Note that this QoS Snippet does not configure the **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) or **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) QoS policies. It is intended to be used in combination with other QoS Snippets that configure those policies.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.KeepLast"

6.80.2.54 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE [static]
```

QoS Snippet that configures the reliability protocol for sending data at a high rate.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet optimizes the **com.rti.dds.publication.DataWriter** (p. 553) reliable protocol parameters for a reliable **com.rti.dds.publication.DataWriter** (p. 553) that is writing messages at high rates, especially in situations where throughput is favored over latency.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

This QoS Snippet sets a fast rate of heartbeats so that errors are detected and repaired more swiftly.

Note that to get the highest thoughout you may need to apply additional changes to the final QoS Profile. See the QoS Profile **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_RELIABLE_HIGH_↩ THROUGHPUT** (p. 183) for further information.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.HighRate"

6.80.2.55 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY [static]
```

QoS Snippet that configures the reliability protocol for sending data at low latency.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

This QoS Snippet modifies the Reliable Protocol parameters to accomplish low latency.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

Note that to get the lowest latency you may need to apply additional changes to the final QoS Profile. See the QoS Profile **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY** (p. 184) for further information.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.LowLatency"

6.80.2.56 SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA

```
final String SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA [static]
```

QoS Snippet that configures the reliability protocol for large data.

The QoS Snippets starting with "Optimization.ReliabilityProtocol." provide alternative configurations of the same QoS Policies. Choose just one QoS Snippet of the alternatives when defining a new QoS Profile or QoS Snippet.

Modifies the Reliable Protocol parameters and Resource Limits to work with Large Data.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504)

Note that to send large data you need to apply additional changes that configure the data caches, asynchronous writing, transport buffers, etc. See, for example, **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_↩ KEEP_LAST_RELIABLE_LARGE_DATA** (p. 185), and derivatives for fully functional Large Data QoS Profiles.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.ReliabilityProtocol.LargeData"

6.80.2.57 SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC

```
final String SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC [static]
```

Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.data_writer.history.memory_manager.∗"
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590)
- **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.data_reader.history.memory_manager.∗"

This configuration is needed to handle data that contains unbounded sequences or strings. This QoS Snippet is also recommended if samples can have very different sizes and the bigger samples can be very large.

If dynamic memory allocation is not used for the larger samples, then all samples are allocated to their maximum size which can consume a lot of resources.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.DataCache.LargeData.DynamicMemAlloc"

6.80.2.58 SNIPPET_OPTIMIZATION_DISCOVERY_COMMON

```
final String SNIPPET_OPTIMIZATION_DISCOVERY_COMMON [static]
```

QoS Snippet that optimizes discovery with a common configuration.

Optimizes the **com.rti.dds.domain.DomainParticipantQos** (p. 795) to detect faster discovery changes. This QoS Snippet increases the speed moderately so that it fits the normal scenarios.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_lease_duration** (p. 649)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_assert_period** (p. 649)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_liveliness_loss_detection_period** (p. 650)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.initial_participant_announcements** (p. 650)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.Discovery.Common"

6.80.2.59 SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT

```
final String SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT [static]
```

QoS Snippet that optimizes the Participant QoS to send less discovery information.

Modified QoS Parameters:

- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.participant.inter_participant.*"
- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.sys_info.*"

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.Discovery.Participant.Compact"

6.80.2.60 SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST

```
final String SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST [static]
```

QoS Snippet that optimizes the Endpoint Discovery to be faster.

This is useful when using security, to prevent a noticeable delay.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652)
- **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.Discovery.Endpoint.Fast"

6.80.2.61 SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS

```
final String SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS [static]
```

QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.

This is useful when using Large Data

Modified QoS Parameters:

- **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523)
- **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Optimization.Transport.LargeBuffers"

6.80.2.62 SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE

```
final String SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE [static]
```

QoS Snippet that sets RELIABILITY QoS to RELIABLE.

This also configures a blocking time in case the **com.rti.dds.publication.DataWriter** (p. 553) writes faster than the DataReaders can accommodate.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526)

Note that by itself enabling reliability does not ensure that every sample written is delivered to the DataReaders. This is because the **com.rti.dds.publication.DataWriter** (p. 553) and/or **com.rti.dds.subscription.DataReader** (p. 450) can be configured to override samples in its cache based on the configuration of the **com.rti.dds.infrastructure.History↩ QosPolicy** (p. 1144) QoS policy.

To ensure delivery of every sample (at the expense of potentially blocking the **com.rti.dds.publication.Data↩ Writer** (p. 553)), use the QoS Profile **com.rti.dds.infrastructure.BuiltinQosProfiles.PROFILE_GENERIC_STRICT↩ _RELIABLE** (p. 182) or one of the derived QoS Profiles.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Reliability.Reliable"

6.80.2.63 SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT

```
final String SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT  [static]
```

QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526)

With best-effort, there are no resources spent to confirm delivery of samples nor repairs of any samples that may be lost.

Best-effort communication reduces jitter; therefore, the delay between sending data and receiving it is more deterministic for the samples that are actually received. Best-effort is good for periodic data where it may be better to get the next value than to wait for the previous one to be repaired.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Reliability.BestEffort"

6.80.2.64 SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1

```
final String SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1  [static]
```

QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.History.KeepLast_1"

6.80.2.65 SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL

```
final String SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL [static]
```

QoS Snippet that sets HISTORY **QosPolicy** (p. 1501) to KEEP_ALL kind.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.History.KeepAll"

6.80.2.66 SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS

```
final String SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS [static]
```

QoS Snippet that sets PUBLISH_MODE **QosPolicy** (p. 1501) to ASYNCHRONOUS kind.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)

Asynchronous Publish mode decouples the application thread that calls the **com.rti.dds.publication.DataWriter** (p. 553) "write" operation from the thread used to send the data on the network. See `https://community.rti.com/glossary/asynchronous-writer`

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.PublishMode.Asynchronous"

6.80.2.67 SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL

```
final String SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL [static]
```

QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to TRANSIENT_LOCAL kind.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)

DataWriters will store and send previously published DDS samples for delivery to newly discovered DataReaders as long as the **com.rti.dds.publication.DataWriter** (p. 553) still exists. For this setting to be effective, you must also set the **com.rti.dds.infrastructure.ReliabilityQosPolicyKind** (p. 1531) to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) (not Best Effort). Which particular DDS samples are kept depends on other QoS settings such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Durability.TransientLocal"

6.80.2.68 SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT

```
final String SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT [static]
```

QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to TRANSIENT kind.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)

RTI Connext will store previously published DDS samples in memory using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy** (p. 1590) of the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **com.rti.dds.infrastructure.DurabilityQosPolicyKind** (p. 835) of the DataWriters configured with com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT↩ _DURABILITY_QOS.

You need a Persistence Service instance running to use this behavior.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Durability.Transient"

6.80.2.69 SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT

```
final String SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT [static]
```

QoS Snippet that sets DURABILITY **QosPolicy** (p. 1501) to PERSISTENT kind.

Modified QoS Parameters:

- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)
- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)

RTI Connext will store previously published DDS samples in permanent storage, like a disk, using Persistence Service, which will send the stored data to newly discovered DataReaders. Which particular DDS samples are kept and sent by Persistence Service depends on the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy** (p. 1590) in the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **com.rti.dds.infrastructure.DurabilityQos↩ PolicyKind** (p. 835) of the DataWriters configured with com.rti.dds.infrastructure.DurabilityQosPolicyKind.Durability↩ QosPolicyKind.PERSISTENT_DURABILITY_QOS.

You need a Persistence Service instance running to use this behavior.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Durability.Persistent"

6.80.2.70 SNIPPET_QOS_POLICY_BATCHING_ENABLE

```
final String SNIPPET_QOS_POLICY_BATCHING_ENABLE  [static]
```

QoS Snippet that sets BATCH **QosPolicy** (p. 1501) to true.

Modified QoS Parameters:

- **com.rti.dds.infrastructure.BatchQosPolicy** (p. 355)

This QoS Snippet specifies and configures the mechanism that allows RTI Connext to collect multiple user data DDS samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "QosPolicy.Batching.Enable"

6.80.2.71 SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS

```
final String SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS  [static]
```

QoS Snippet that configures and set a FlowController of 838 Mbps.

Defines a **com.rti.dds.publication.FlowController** (p. 1055) and configures the **com.rti.dds.publication.Data↩ WriterQos** (p. 612) with it.

This is a **com.rti.dds.publication.FlowController** (p. 1055) of 838 Mbps ($\sim$ 100 MB/sec)

Modified QoS Parameters:

- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.flow_controller.token_bucket.$*$"
- **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.FlowController.838Mbps"

6.80.2.72 SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS

```
final String SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS [static]
```

QoS Snippet that configures and sets a FlowController of 209 Mbps.

Defines a **com.rti.dds.publication.FlowController** (p. 1055) and configures the **com.rti.dds.publication.Data↩ WriterQos** (p. 612) with it.

This is a **com.rti.dds.publication.FlowController** (p. 1055) of 209 Mbps ($\sim$ 25 MB/sec)

Modified QoS Parameters:

- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.flow_controller.token_bucket.$*$"
- **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.FlowController.209Mbps"

6.80.2.73 SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS

```
final String SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS [static]
```

QoS Snippet that configures and sets a FlowController of 52 Mbps.

Defines a **com.rti.dds.publication.FlowController** (p. 1055) and configures the **com.rti.dds.publication.Data↩ WriterQos** (p. 612) with it.

This is a **com.rti.dds.publication.FlowController** (p. 1055) of 52 Mbps ($\sim$ 6.25 MB/sec)

Modified QoS Parameters:

- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.flow_controller.token_bucket.$*$"
- **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.FlowController.52Mbps"

6.80.2.74 SNIPPET_FEATURE_AUTO_TUNING_ENABLE

```
final String SNIPPET_FEATURE_AUTO_TUNING_ENABLE  [static]
```

QoS Snippet that enables auto_throttle and turbo_mode to true.

Sets the **com.rti.dds.domain.DomainParticipantQos** (p. 795) properties to enable auto_throttle and turbo_mode to true.

Modified QoS Parameters:

- **com.rti.dds.domain.DomainParticipantQos** (p. 795) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.domain_participant.auto_throttle"
- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.data_writer.auto_throttle.enable"
- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) named "dds.data_writer.enable_turbo_mode"

The domain_participant.auto_throttle configures the **com.rti.dds.domain.DomainParticipant** (p. 670) to gather internal measurements (during **com.rti.dds.domain.DomainParticipant** (p. 670) creation) that are required for the Auto Throttle feature. This allows DataWriters belonging to this **com.rti.dds.domain.DomainParticipant** (p. 670) to use the Auto Throttle feature.

The turbo_mode adjusts the batch max_data_bytes based on how frequently the **com.rti.dds.publication.DataWriter** (p. 553) writes data.

Data_writer.auto_throttle enables automatic throttling in the **com.rti.dds.publication.DataWriter** (p. 553) so it can automatically adjust the writing rate and the send window size; this minimizes the need for repairing DDS samples and improves latency.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.AutoTuning.Enable"

6.80.2.75 SNIPPET_FEATURE_MONITORING_ENABLE

```
final String SNIPPET_FEATURE_MONITORING_ENABLE  [static]
```

QoS Snippet that enables the use of the RTI Monitoring Library.

To enable the use of RTI Monitoring Library apply this QoS Snippet to the QoS Profile used to create your **com.rti.↩ dds.domain.DomainParticipant** (p. 670). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Monitoring.Enable</element>

    </base_name>

</qos_profile>
```

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.Monitoring.Enable"

6.80.2.76 SNIPPET_FEATURE_MONITORING2_ENABLE

```
final String SNIPPET_FEATURE_MONITORING2_ENABLE  [static]
```

QoS Snippet that enables the use of the RTI Monitoring Library 2.0.

QoS Snippet to enable the use of the RTI Monitoring Library 2.0 with a dedicated DomainParticipant publishing telemetry data from your application in domain ID 2. All metrics are enabled for all resources in this profile.

To enable the use of RTI Monitoring Library 2.0, apply this QoS Snippet to the QoS Profile used to create your DomainParticipantFactory. For example:

```
<qos_profile name="MyProfile" is_default_participant_factory_profile="true">
  <base_name>
    <element>BuiltinQosSnippetLib::Feature.Monitoring2.Enable</element>
  </base_name>
</qos_profile>
```

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.Monitoring2.Enable"

6.80.2.77 SNIPPET_FEATURE_SECURITY_ENABLE

```
final String SNIPPET_FEATURE_SECURITY_ENABLE  [static]
```

QoS Snippet that enables security using the Builtin Security Plugins.

To enable the use of the Builtin DDS Security Library apply this QoS Snippet to the QoS Profile used to create your **com.rti.dds.domain.DomainParticipant** (p. 670). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.Security.Enable</element>

    </base_name>

</qos_profile>
```

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.Security.Enable"

6.80.2.78 SNIPPET_FEATURE_TOPIC_QUERY_ENABLE

```
final String SNIPPET_FEATURE_TOPIC_QUERY_ENABLE  [static]
```

QoS Snippet that enables Topic Query.

To enable the use of the RTI Connext **com.rti.dds.subscription.TopicQuery** (p. 1830) feature, apply this QoS Snippet to the QoS Profile used to create your **com.rti.dds.publication.DataWriter** (p. 553). For example:

```
 <qos_profile name="MyProfile">

    <base_name>

        <element>BuiltinQosLib::Feature.TopicQuery.Enable</element>

    </base_name>

</qos_profile>
```

For more information on Topic Query see the "Topic Queries" chapter in the `User's Manual`.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Feature.TopicQuery.Enable"

6.80.2.79 SNIPPET_TRANSPORT_TCP_LAN_CLIENT

```
final String SNIPPET_TRANSPORT_TCP_LAN_CLIENT = "Transport.TCP.LAN.Client"  [static]
```

QoS Snippet that configures a TCP LAN Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the initial_peers and the property dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Therefore, they must be modified:

- initial_peers: should point to the remote client IP and port.
- server_bind_port: is the port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **com.rti.dds.infrastructure.Entity** (p. 1029). These new values will overwrite the current invalid values.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.TCP.LAN.Client"

6.80.2.80 SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT

```
final String SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT  [static]
```

QoS Snippet that configures a symmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_port are incorrect (just sample strings). Also the initial_peers information is not correct. Therefore, the following must be modified:

- initial_peers: should point to the remote client IP and port.
- public_address: public IP address where this application can be reached.
- server_bind_port: port this application will be using.

This modification should be done in the QoS Profile that will be used to create the **com.rti.dds.infrastructure.Entity** (p. 1029). These new values will overwrite the current invalid values.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.TCP.WAN.Symmetric.Client"

6.80.2.81 SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER

```
final String SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER [static]
```

QoS Snippet that an asymmetric WAN TCP Server over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The values of the properties dds.transport.TCPv4.tcp1.public_address and dds.transport.TCPv4.tcp1.server_bind_↩ port are incorrect (just sample strings). Therefore, they must be modified to the corresponding public_address and port_number. This modification should be done in the QoS Profile that will be used to create the **com.rti.dds.↩ infrastructure.Entity** (p. 1029). These new values will overwrite the current invalid values.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.TCP.WAN.Asymmetric.Server"

6.80.2.82 SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT

```
final String SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT [static]
```

QoS Snippet that configures an asymmetric WAN TCP Client over DDS.

This QoS Snippet sets all the mandatory properties; however the final QoS Profile requires additional configuration.

The value of discovery.initial_peers and public_ip have to match the values set on the Server side (**com.rti.dds.↩ infrastructure.BuiltinQosProfiles.SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER** (p. 206)). This modification should be done in the QoS Profile that will be used to create the **com.rti.dds.infrastructure.Entity** (p. 1029). This new value will overwrite the current invalid value.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.TCP.WAN.Asymmetric.Client"

6.80.2.83 SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION

```
final String SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION [static]
```

QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.

For WAN communications and, in general, for communications in third party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

This snippet provides a way to avoid IP fragmentation in Connext applications using the built-in UDP transports. Instead, Connext will be responsible for fragmentation, which is done at the RTPS level.

Among other changes, this configuration changes the transport MTU (message_size_max) to be 1400 bytes. Notice that this change will affect other transports such as SHMEM since Connext chooses the minimum transport MTU across all enabled transports to determine the maximum size of outgoing RTPS messages.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.UDP.AvoidIPFragmentation"

6.80.2.84 SNIPPET_TRANSPORT_UDP_WAN

```
final String SNIPPET_TRANSPORT_UDP_WAN [static]
```

QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).

The snippet disables all the other built-in transports and avoids the use of IP fragmentation.

For WAN communications and, in general, for communications in third-party networks, it is not a good idea to rely on IP fragmentation. IP fragmentation causes significant issues in UDP, where there is no integrated support for a path MTU (maximum transmission unit) discovery protocol as there is in TCP.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Transport.UDP.WAN"

6.80.2.85 SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3

```
final String SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3 [static]
```

QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.

QoS Snippet that sets the **com.rti.dds.subscription.DataReaderQos** (p. 517) and **com.rti.dds.publication.Data↩ WriterQos** (p. 612) **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247) to com.rti.dds.infrastructure.↩ LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS It also disables the built-in shared memory transport.

Modified QoS Parameters:

- **com.rti.dds.subscription.DataReaderQos** (p. 517) => **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243)
- **com.rti.dds.publication.DataWriterQos** (p. 612) => **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243)
- **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843)

RTI Connext Micro versions 2.4.3 and earlier only supported com.rti.dds.infrastructure.LivelinessQosPolicyKind.↩ LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS **com.rti.dds.infrastructure.LivelinessQos↩ PolicyKind** (p. 1247). In order to be compatible with RTI Connext Micro 2.4.3, the **com.rti.dds.subscription.↩ DataReader** (p. 450) and **com.rti.dds.publication.DataWriter** (p. 553) must have their **com.rti.dds.infrastructure.↩ LivelinessQosPolicyKind** (p. 1247) changed to this value because the default kind in RTI Connext is com.rti.dds.↩ infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Compatibility.ConnextMicro.Version243"

6.80.2.86 SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE

```
final String SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE  [static]
```

QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Compatibility.OtherDDSVendor.Enable"

6.80.2.87 SNIPPET_5_1_0_TRANSPORT_ENABLE

```
final String SNIPPET_5_1_0_TRANSPORT_ENABLE  [static]
```

QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

In Library: **com.rti.dds.infrastructure.BuiltinQosProfiles.BUILTIN_QOS_SNIPPET_LIB** (p. 193)

String-version: "Compatibility.510Transport.Enable"

6.81 Conditions and WaitSets

com.rti.dds.infrastructure.Condition (p. 429) and **com.rti.dds.infrastructure.WaitSet** (p. 1973) and related items.

Classes

- interface **Condition**

  *<<**interface**>> (p. 156) Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1973).*
- class **ConditionSeq**

  *Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.Condition** (p. 429) >*
- class **GuardCondition**

  *<<**interface**>> (p. 156) A specific **com.rti.dds.infrastructure.Condition** (p. 429) whose `trigger_value` is completely under the control of the application.*
- interface **StatusCondition**

  *<<**interface**>> (p. 156) A specific **com.rti.dds.infrastructure.Condition** (p. 429) that is associated with each **com.rti.dds.infrastructure.Entity** (p. 1029).*
- class **WaitSet**

  *<<**interface**>> (p. 156) Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.Condition** (p. 429) objects has a `trigger_value` of com.rti.dds.infrastructure.true or else until the timeout expires.*
- class **WaitSetProperty_t**

  *<<**extension**>> (p. 155) Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1973) behavior for multiple trigger events.*

6.81.1 Detailed Description

com.rti.dds.infrastructure.Condition (p. 429) and **com.rti.dds.infrastructure.WaitSet** (p. 1973) and related items.

6.82 DATABASE

<<***extension***>> *(p. 155)* Various threads and resource limits settings used by RTI Connext to control its internal database.

Classes

- class **DatabaseQosPolicy**

  Various threads and resource limits settings used by RTI Connext to control its internal database.

Variables

- static final **QosPolicyId_t DATABASE_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.DatabaseQosPolicy*** *(p. 445)*

6.82.1 Detailed Description

<<***extension***>> *(p. 155)* Various threads and resource limits settings used by RTI Connext to control its internal database.

6.82.2 Variable Documentation

6.82.2.1 DATABASE_QOS_POLICY_ID

```
final  QosPolicyId_t DATABASE_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 445)

6.83 DATA_READER_PROTOCOL

<<***extension***>> *(p. 155)* Specifies the DataReader-specific protocol QoS.

Classes

- class **DataReaderProtocolQosPolicy**

  *Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.dds.infrastructure.DataWriter↩ ProtocolQosPolicy** (p. 596), this QoS policy configures the DDS on-the-network protocol (RTPS).*

Variables

- static final **QosPolicyId_t DATAREADERPROTOCOL_QOS_POLICY_ID**

  *<<**extension**>> (p. 155) Identifier for **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501)*

6.83.1 Detailed Description

*<<**extension**>> (p. 155)* Specifies the DataReader-specific protocol QoS.

6.83.2 Variable Documentation

6.83.2.1 DATAREADERPROTOCOL_QOS_POLICY_ID

```
final  QosPolicyId_t DATAREADERPROTOCOL_QOS_POLICY_ID  [static]
```

*<<**extension**>> (p. 155)* Identifier for **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501)

6.84 DATA_READER_RESOURCE_LIMITS

*<<**extension**>> (p. 155)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Classes

- class **DataReaderInstanceRemovalKind**

  *Sets the kinds of instances that can be replaced when instance resource limits (**com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.max_instances** (p. 1592)) are reached.*
- class **DataReaderResourceLimitsInstanceReplacementSettings**

  Instance replacement kind applied to each instance state.
- class **DataReaderResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 450) allocates and uses physical memory for internal resources.*

Variables

- static final int **AUTO_MAX_TOTAL_INSTANCES**

  <<***extension***>> *(p. 155) This value is used to make* ***com.rti.dds.infrastructure.DataReaderResourceLimitsQos↩ Policy.max_total_instances*** *(p. 536) equal to* ***com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances*** *(p. 1592).*
- static final **QosPolicyId_t DATAREADERRESOURCELIMITS_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy*** *(p. 529)*

6.84.1 Detailed Description

<<***extension***>> *(p. 155)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

6.84.2 Variable Documentation

6.84.2.1 AUTO_MAX_TOTAL_INSTANCES

```
final int AUTO_MAX_TOTAL_INSTANCES  [static]
```

<<***extension***>> *(p. 155)* This value is used to make **com.rti.dds.infrastructure.DataReaderResourceLimits↩ QosPolicy.max_total_instances** (p. 536) equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_↩ instances** (p. 1592).

6.84.2.2 DATAREADERRESOURCELIMITS_QOS_POLICY_ID

```
final  QosPolicyId_t DATAREADERRESOURCELIMITS_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529)

6.85 DATA_REPRESENTATION

A list of data representations and compression methods supported by a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450).

Classes

- class **DataRepresentationQosPolicy**

  This QoS policy contains a list of representation identifiers and compression settings used by ***com.rti.dds.publication.↩ DataWriter*** *(p. 553) and* ***com.rti.dds.subscription.DataReader*** *(p. 450) entities to negotiate which data representation and compression settings to use.*

Variables

- static final short **XCDR_DATA_REPRESENTATION** = 0

  Corresponds to the Extended Common Data Representation encoding version 1.
- static final short **XML_DATA_REPRESENTATION** = 1

  Corresponds to the XML Data Representation (unsupported).
- static final short **XCDR2_DATA_REPRESENTATION** = 2

  Corresponds to the Extended Common Data Representation encoding version 2.
- static final short **AUTO_DATA_REPRESENTATION** = -1

  Representation is automatically chosen based on the type.
- static final **QosPolicyId_t** **DATA_REPRESENTATION_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DataRepresentationQosPolicy** (p. 544).*

6.85.1 Detailed Description

A list of data representations and compression methods supported by a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450).

6.85.2 Variable Documentation

6.85.2.1 XCDR_DATA_REPRESENTATION

```
final short XCDR_DATA_REPRESENTATION = 0  [static]
```

Corresponds to the Extended Common Data Representation encoding version 1.

6.85.2.2 XML_DATA_REPRESENTATION

```
final short XML_DATA_REPRESENTATION = 1  [static]
```

Corresponds to the XML Data Representation (unsupported).

Note

This value is currently not supported.

6.85.2.3 XCDR2_DATA_REPRESENTATION

```
final short XCDR2_DATA_REPRESENTATION = 2  [static]
```

Corresponds to the Extended Common Data Representation encoding version 2.

6.85.2.4 AUTO_DATA_REPRESENTATION

```
final short AUTO_DATA_REPRESENTATION = -1  [static]
```

Representation is automatically chosen based on the type.

For plain language binding, if the allowed_data_representation annotation is not specified or if it contains the value XCDR, RTI Connext translates this field to **com.rti.dds.infrastructure.DataRepresentationQosPolicy.XCDR_DATA_REPRESENTATION** (p. 213). Otherwise, it translates this field to **com.rti.dds.infrastructure.DataRepresentationQosPolicy.XCDR2_DATA_REPRESENTATION** (p. 213).

For the **FlatData language binding** (p. 55), RTI Connext translates this field to **com.rti.dds.infrastructure.DataRepresentationQosPolicy.XCDR2_DATA_REPRESENTATION** (p. 213).

Referenced by **TypeCode.cdr_serialized_sample_key_max_size()**, **TypeCode.cdr_serialized_sample_max_size()**, **TypeCode.cdr_serialized_sample_min_size()**, **DynamicData.to_cdr_buffer()**, and **DynamicDataTypeSupport.to_cdr_buffer()**.

6.85.2.5 DATA_REPRESENTATION_QOS_POLICY_ID

```
final  QosPolicyId_t DATA_REPRESENTATION_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.DataRepresentationQosPolicy** (p. 544).

6.86 DATA_TAG

Stores (name, value) pairs that can be used to determine access permissions.

Classes

- class **DataTagQosPolicy**

  Stores (name, value) pairs that can be used to determine access permissions.
- class **DataTagQosPolicyHelper**

  Policy helpers that facilitate management of the data tags in the input policy.
- class **Tag**

  Tags are name/value pair objects.
- class **TagSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.Tag*** *(p. 1783) >*

Variables

- static final **QosPolicyId_t DATATAG_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DataTagQosPolicy** (p. 547).*

6.86.1 Detailed Description

Stores (name, value) pairs that can be used to determine access permissions.

The **com.rti.dds.infrastructure.DataTagQosPolicy** (p. 547) can be used to associate a set of tags in the form of (name, value) pairs with a **com.rti.dds.subscription.DataReader** (p. 450) or **com.rti.dds.publication.DataWriter** (p. 553). This is similar to the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438), except you cannot select whether or not a particular pair should be propagated (included in the built-in topic). Data tags are always propagated. The Access Control plugin may use the tags to determine publish and subscribe permissions.

6.86.2 Variable Documentation

6.86.2.1 DATATAG_QOS_POLICY_ID

```
final QosPolicyId_t DATATAG_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.DataTagQosPolicy** (p. 547).

6.87 DATA_WRITER_PROTOCOL

<<***extension***>> *(p. 155)* Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.↩ dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501), this QoS policy configures the DDS on-the-network protocol (RTPS).

Classes

- class **DataWriterProtocolQosPolicy**

  *Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 553) instances.*

Variables

- static final **QosPolicyId_t DATAWRITERPROTOCOL_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596)*

6.87.1 Detailed Description

<<***extension***>> *(p. 155)* Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.↩ dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501), this QoS policy configures the DDS on-the-network protocol (RTPS).

6.87.2 Variable Documentation

6.87.2.1 DATAWRITERPROTOCOL_QOS_POLICY_ID

```
final  QosPolicyId_t DATAWRITERPROTOCOL_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596)

6.88 DATA_WRITER_RESOURCE_LIMITS

<<***extension***>> *(p. 155)* Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.

Classes

- class **DataWriterResourceLimitsInstanceReplacementKind**

  Sets the kinds of instances that can be replaced when instance resource limits are reached.
- class **DataWriterResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.*

Variables

- static final **QosPolicyId_t DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy** (p. 626)*

6.88.1 Detailed Description

<<***extension***>> *(p. 155)* Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.

6.88.2 Variable Documentation

6.88.2.1 DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID

```
final QosPolicyId_t DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy** (p. 626)

6.89 DEADLINE

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Classes

- class **DeadlineQosPolicy**
  Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Variables

- static final **QosPolicyId_t DEADLINE_QOS_POLICY_ID**
  *Identifier for **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632).*

6.89.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

6.89.2 Variable Documentation

6.89.2.1 DEADLINE_QOS_POLICY_ID

```
final QosPolicyId_t DEADLINE_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632).

6.90 DESTINATION_ORDER

Controls the criteria used to determine the logical order among changes made by **com.rti.dds.publication.Publisher** (p. 1466) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1807) and key).

Classes

- class **DestinationOrderQosPolicy**

  *Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1807) and key).*
- class **DestinationOrderQosPolicyKind**

  Kinds of destination order.
- class **DestinationOrderQosPolicyScopeKind**

  Scope of source destination order.

Variables

- static final **QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DestinationOrderQosPolicy** (p. 635).*

6.90.1 Detailed Description

Controls the criteria used to determine the logical order among changes made by **com.rti.dds.publication.Publisher** (p. 1466) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1807) and key).

6.90.2 Variable Documentation

6.90.2.1 DESTINATIONORDER_QOS_POLICY_ID

```
final  QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.DestinationOrderQosPolicy** (p. 635).

6.91 DISCOVERY_CONFIG

<<***extension***>> *(p. 155)* Specifies the discovery configuration QoS.

Classes

- class **BuiltinTopicReaderResourceLimits_t**

  Built-in topic reader's resource limits.
- class **DiscoveryConfigBuiltinChannelKind**

  Built-in channels that can be enabled.
- class **DiscoveryConfigBuiltinPluginKind**

  Built-in discovery plugins that can be used.
- class **DiscoveryConfigQosPolicy**

  Settings for discovery configuration.
- class **RemoteParticipantPurgeKind**

  Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

Variables

- static final int **MASK_NONE**

  No bits are set, indicating that all channels that can be disabled, are disabled. Not all builtin channels can be disabled by the user. Only builtin channels represented by ***com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKind*** *(p. 643) are able to be enabled or disabled by the user.*
- static final int **MASK_ALL**

  All bits are set, indicating that all channels are enabled.
- static final int **MASK_DEFAULT** = **SERVICE_REQUEST_CHANNEL**

  The default value of ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.enabled_builtin_channels*** *(p. 655).*
- static final int **MASK_NONE**

  No bits are set.
- static final int **MASK_DEFAULT**

  The default value of ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.builtin_discovery_plugins*** *(p. 655).*
- static final **QosPolicyId_t** **DISCOVERYCONFIG_QOS_POLICY_ID**

  *<<**extension**>> (p. 155) Identifier for* ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy*** *(p. 646)*

6.91.1 Detailed Description

*<<**extension**>> (p. 155)* Specifies the discovery configuration QoS.

6.91.2 Variable Documentation

6.91.2.1 MASK_NONE [1/2]

```
final int MASK_NONE  [static]
```

No bits are set, indicating that all channels that can be disabled, are disabled. Not all builtin channels can be disabled by the user. Only builtin channels represented by **com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKind** (p. 643) are able to be enabled or disabled by the user.

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKindMask

6.91.2.2 MASK_ALL

```
final int MASK_ALL  [static]
```

All bits are set, indicating that all channels are enabled.

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKindMask

6.91.2.3 MASK_DEFAULT [1/2]

```
final int MASK_DEFAULT =  SERVICE_REQUEST_CHANNEL  [static]
```

The default value of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.enabled_builtin_channels** (p. 655).

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKindMask

6.91.2.4 MASK_NONE [2/2]

```
final int MASK_NONE  [static]
```

No bits are set.

6.91.2.5 MASK_DEFAULT [2/2]

```
final int MASK_DEFAULT  [static]
```

The default value of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.builtin_discovery_plugins** (p. 655).

6.91.2.6 DISCOVERYCONFIG_QOS_POLICY_ID

```
final  QosPolicyId_t DISCOVERYCONFIG_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy** (p. 646)

6.92 DISCOVERY

<<***extension***>> *(p. 155)* Specifies the attributes required to discover participants in the domain.

Modules

- **NDDS_DISCOVERY_PEERS**

  *Environment variable or a file that specifies the default values of **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) contained in the **com.rti.dds.domain.DomainParticipantQos.discovery** (p. 800) qos policy.*

Classes

- class **DiscoveryQosPolicy**

  <<***extension***>> *(p. 155) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.*

Variables

- static final **QosPolicyId_t DISCOVERY_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 665)*

6.92.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies the attributes required to discover participants in the domain.

6.92.2 Variable Documentation

6.92.2.1 DISCOVERY_QOS_POLICY_ID

```
final QosPolicyId_t DISCOVERY_QOS_POLICY_ID [static]
```

<<***extension***>> *(*p. *155)* Identifier for **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 665)

6.93 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) contained in the **com.rti.dds.domain.DomainParticipantQos.discovery** (p. 800) qos policy.

Environment variable or a file that specifies the default values of **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) contained in the **com.rti.dds.domain.DomainParticipantQos.discovery** (p. 800) qos policy.

The default value of the **com.rti.dds.domain.DomainParticipantQos** (p. 795) is obtained by calling **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos()** (p. 767).

NDDS_DISCOVERY_PEERS specifies the default value of the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) fields, when the default participant QoS policies have not been explictly set by the user (i.e., **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos()** (p. 768) has never been called or was called using **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42)).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667), starting at element 0. They will be stored in the order in which they appear in NDDS_DISCOVERY_PEERS.

Note: IPv4 multicast addresses must have a prefix. Therefore, when using the UDPv6 transport: if there are any IPv4 multicast addresses in the peers list, make sure they have "udpv4://" in front of them (such as udpv4://239.255.0.1).

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667).

NDDS_DISCOVERY_PEERS provides a mechanism to dynamically switch the discovery configuration of an RTI Connext application without recompilation. The application programmer is free to not use the default values; instead use values supplied by other means.

NDDS_DISCOVERY_PEERS can be specified either in an environment variable as comma (',') separated "peer descriptors" (see **Peer Descriptor Format** (p. 223)) or in a file. These formats are described below.

6.93.1 Peer Descriptor Format

A **peer descriptor** string specifies a range of participants at a given `locator`. Peer descriptor strings are used in the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) field and the **com.rti.dds.domain.DomainParticipant.add_peer()** (p. 729) operation.

The anatomy of a `peer` descriptor is illustrated below using a UDPv4 transport and a custom "StarFabric" transport example.

A peer descriptor consists of:

optional **Participant ID Limit**. If a simple integer is specified, it indicates the maximum participant ID to be contacted by the RTI Connext discovery mechanism at the given locator. If that integer is enclosed in square brackets (e.g.: [2]) *only* that Participant ID will be used. You can also specify a range in the form of [a-b]: in this case only the Participant IDs in that specific range are contacted. If omitted, a default value of 4 is implied: participant IDs 0,1,2,3, and 4 will be contacted.

- **Locator**. See **Locator Format** (p. 223).

These are separated by the '@' character. The separator may be omitted if a participant ID limit is not explictly specified.

Note that the "participant ID limit" only applies to unicast locators; it is ignored for multicast locators (and therefore should be omitted for multicast peer descriptors).

6.93.1.1 Locator Format

A **locator** string specifies a transport and an address in string format. Locators are used to form peer descriptors. A locator is equivalent to a peer descriptor with the default maximum participant ID.

A locator consists of:

optional **Transport name** (**alias** or class). This identifies the set of transport plugins (**Transport Aliases** (p. 95)) that may be used to parse the `address` portion of the locator. Note that a transport class name is an implicit alias that is used to refer to all the transport plugin instances of that class.

optional **Address**. See **Address Format** (p. 224).

These are separated by the "//:" string. The separator is specified if and only if a transport name is specified.

If a transport name is specified, the address may be omitted; in that case, all the unicast addresses (across all transport plugin instances) associated with the transport class are implied. Thus, a locator string may specify several addresses.

If an address is specified, the transport name and the separator string may be omitted; in that case all the available transport plugins (for the **com.rti.dds.infrastructure.Entity** (p. 1029)) may be used to parse the address string.

6.93.1.2 Address Format

An **address** string specifies a transport-independent network address that qualifies a **transport-dependent** address string. Addresses are used to form locators. Addresses are also used in **com.rti.dds.infrastructure.Discovery↩ QosPolicy.multicast_receive_addresses** (p. 667), and **com.rti.dds.infrastructure.TransportMulticastSettings_t.↩ receive_address** (p. 1857) fields. An address is equivalent to a locator in which the transport name and separator are omitted.

An address consists of:

optional **Network Address**. An address in IPv4 or IPv6 string notation. If omitted, the network address of the transport is implied (**Transport Network Address** (p. 98)).

optional **Transport Address**. A string that is passed to the transport for processing. The transport maps this string into **com.rti.ndds.transport.Transport.Property_t.address_bit_count** (p. 1403) bits. If omitted the network address is used as the fully qualified address.

These are separated by the '#' character. If a separator is specified, it must be followed by a non-empty string which is passed to the transport plugin. If the separator is omitted, it is treated as a transport address with an implicit network address (of the transport plugin). The implicit network address is the address used when registering the transport: e.g, the UDPv4 implicit network address is 0.0.0.0.0.0.0.0.0.0.

The bits resulting from the transport address string are combined with the network address. The least or most significant **com.rti.ndds.transport.Transport.Property_t.address_bit_count** (p. 1403) bits of the network address are ignored and replaced with the transport address. The sign of the **com.rti.ndds.transport.Transport.Property_t.address_↩ bit_count** (p. 1403) value determines whether the most significant bits (negative sign) or least significant bits (positive sign) are ignored (**Transport Network Address** (p. 98)).

6.93.2 NDDS_DISCOVERY_PEERS Environment Variable Format

NDDS_DISCOVERY_PEERS can be specified via an environment variable of the same name, consisting of a sequence of peer descriptors separated by the comma (',') character.

Examples

Multicast (maximum participant ID is irrelevant)

- 239.255.0.1

Default maximum participant ID on localhost

- localhost

Default maximum participant ID on host 192.168.1.1 (IPv4)

- 192.168.1.1

Default maximum participant ID on host FAA0::0 (IPv6)

- FAA0::1

Default maximum participant ID on host himalaya accessed using the "udpv4" transport plugin(s) (IPv4)

- udpv4://himalaya

Default maximum participant ID on localhost using the "udpv4" transport plugin(s) registered at network address FAA0::0

- udpv4://FAA0::0#localhost

Default maximum participant ID on host 0/0/R (StarFabric)

- 0/0/R
- #0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s)

- starfabric://0/0/R
- starfabric://#0/0/R

Default maximum participant ID on host 0/0/R (StarFabric) using the "starfabric" (StarFabric) transport plugin(s) registered at network address FAA0::0

- starfabric://FBB0::0#0/0/R

Default maximum participant ID on all unicast addresses accessed via the "starfabric" (StarFabric) transport plugin(s)

- starfabric://

Default maximum participant ID on all unicast addresses accessed via the "shmem" (shared memory) transport plugin(s) registered at network address FCC0::0

- shmem://FCC0::0

Default maximum participant ID on hosts himalaya and gangotri

- himalaya,gangotri

Maximum participant ID of 1 on hosts himalaya and gangotri

- 1@himalaya,1@gangotri

Combinations of above

- 239.255.0.1,localhost,192.168.1.1,0/0/R
- FAA0::1,FAA0::0#localhost,FBB0::0#0/0/R
- udpv4://himalaya,udpv4://FAA0::0#localhost,#0/0/R
- starfabric://0/0/R,starfabric://FBB0::0#0/0/R,shmem://
- starfabric://,shmem://FCC0::0,1@himalaya,1@gangotri

6.93.3 NDDS_DISCOVERY_PEERS File Format

NDDS_DISCOVERY_PEERS can be specified via a file of the same name in the program's current working directory. A NDDS_DISCOVERY_PEERS file would contain a sequence of peer descriptors separated by whitespace or the comma (',') character. The file may also contain comments starting with a semicolon (';') character till the end of the line.

Example:

```
;; NDDS_DISCOVERY_PEERS - Discovery Configuration File
;;
;;
;; NOTE:
;;   1. This file must be in the current working directory, i.e.
;;      in the folder from which the application is launched.
;;
;;   2. This file takes precedence over the environment variable NDDS_DISCOVERY_PEERS
;;
;; Multicast
239.255.0.1                 ; The default dds discovery multicast address
;; Unicast
localhost,192.168.1.1       ; A comma can be used a separator
FAA0::1  FAA0::0#localhost  ; Whitespace can be used as a separator
1@himalaya                  ; Maximum participant ID of 1 on 'himalaya'
1@gangotri
;; UDPv4
udpv4://himalaya            ; 'himalaya' via 'udpv4' transport plugin(s)
udpv4://FAA0::0#localhost   ; 'localhost' via 'updv4' transport
                            ;  plugin registered at network address FAA0::0
;; Shared Memory
shmem://                    ; All 'shmem' transport plugin(s)
builtin.shmem://            ; The builtin 'shmem' transport plugin
shmem://FCC0::0             ; Shared memory transport plugin registered
                            ;                at network address FCC0::0
;; StarFabric
0/0/R                       ; StarFabric node 0/0/R
starfabric://0/0/R          ; 0/0/R accessed via 'starfabric'
                            ;             transport plugin(s)
```

```
starfabric://FBB0::0#0/0/R   ; StarFabric transport plugin registered
                             ;             at network address FBB0::0
starfabric://                ; All 'starfabric' transport plugin(s)
```

6.93.4 NDDS_DISCOVERY_PEERS Precedence

If the current working directory from which the RTI Connext application is launched contains a file called NDDS_↩ DISCOVERY_PEERS, and an environment variable named NDDS_DISCOVERY_PEERS is also defined, the file takes precedence; the environment variable is ignored.

6.93.5 NDDS_DISCOVERY_PEERS Default Value

If NDDS_DISCOVERY_PEERS is not specified (either as a file in the current working directory, or as an environment variable), it implicitly defaults to the following.

```
;; Multicast (only on platforms which allow UDPv4 multicast out of the box)
;;
;; This allows any dds applications anywhere on the local network to
;; discover each other over UDPv4.
builtin.udpv4://239.255.0.1  ; dds's default discovery multicast address
                             ; This is also the default multicast receive address
;; Unicast - UDPv4 (on all platforms)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over UDP/IPv4.
builtin.udpv4://127.0.0.1
;; Unicast - Shared Memory (only on platforms that support shared memory)
;;
;; This allows two dds applications using participant IDs up to the maximum
;; default participant ID on the local host and domain to discover each
;; other over shared memory.
builtin.shmem://
```

6.93.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - **com.rti.ndds.transport.ShmemTransport** (p. 1681)
- `udpv4` - **com.rti.ndds.transport.UDPv4Transport** (p. 1943)
- `udpv6` - **com.rti.ndds.transport.UDPv6Transport** (p. 1952)

These may be used as the transport names in the **Locator Format** (p. 223).

6.93.7 NDDS_DISCOVERY_PEERS and Local Host Communication

Suppose you want to communicate with other RTI Connext applications on the same host and you are setting NDDS↩ _DISCOVERY_PEERS explicitly (generally in order to use unicast discovery with applications on other hosts).

If the local host platform does not support the shared memory transport, then you can include the name of the local host in the NDDS_DISCOVERY_PEERS list.

If the local host platform supports the shared memory transport, then you can do one of the following:

- Include "shmem://" in the NDDS_DISCOVERY_PEERS list. This will cause shared memory to be used for discovery and data traffic for applications on the same host.

or:

- Include the name of the local host in the NDDS_DISCOVERY_PEERS list and disable the shared memory transport in the **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843) of the **com.rti.dds.domain.↩ DomainParticipant** (p. 670). This will cause UDP loopback to be used for discovery and data traffic for applications on the same host.

(To check if your platform supports shared memory, see the `Platform Notes`.)

See also

com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses (p. 667)

com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 668)

com.rti.dds.domain.DomainParticipant.add_peer() (p. 729)

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos() (p. 767)

Transport Aliases (p. 95)

Transport Network Address (p. 98)

6.94 DOMAIN_PARTICIPANT_RESOURCE_LIMITS

<<***extension***>> *(p. 155)* Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Classes

- class **AllocationSettings_t**

  Resource allocation settings.
- class **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**

  Available replacement policies for the ignored entities.
- class **DomainParticipantResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

Variables

- static final **DomainParticipantResourceLimitsIgnoredEntityReplacementKind NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT** = new **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**("NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT", 0)

  Available replacement policies for the ignored entities.
- static final **DomainParticipantResourceLimitsIgnoredEntityReplacementKind NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT** = new **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**("NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT", 1)

  Available replacement policies for the ignored entities.
- static final **QosPolicyId_t DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 803)*

6.94.1 Detailed Description

<<***extension***>> *(p. 155)* Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

6.94.2 Variable Documentation

6.94.2.1 NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT

```
final DomainParticipantResourceLimitsIgnoredEntityReplacementKind NO_REPLACEMENT_IGNORED_ENTITY↩
_REPLACEMENT = new DomainParticipantResourceLimitsIgnoredEntityReplacementKind("NO_REPLACEMENT_↩
IGNORED_ENTITY_REPLACEMENT", 0) [static]
```

Available replacement policies for the ignored entities.

QoS:

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.NO_REPLACEMENT_IGNORED_↩ ENTITY_REPLACEMENT

6.94.2.2 NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT

```
final DomainParticipantResourceLimitsIgnoredEntityReplacementKind NOT_ALIVE_FIRST_IGNORED_↩
ENTITY_REPLACEMENT = new DomainParticipantResourceLimitsIgnoredEntityReplacementKind("NOT_ALIVE↩
_FIRST_IGNORED_ENTITY_REPLACEMENT", 1) [static]
```

Available replacement policies for the ignored entities.

QoS:

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.NOT_ALIVE_FIRST_IGNORED_ENTITY↩ _REPLACEMENT

6.94.2.3 DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID

```
final QosPolicyId_t DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 803)

6.95 DURABILITY

This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.

Classes

- class **DurabilityQosPolicy**

  *This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.*
- class **DurabilityQosPolicyKind**

  Kinds of durability.
- class **PersistentJournalKind**

  Sets the journal mode of the persistent storage.
- class **PersistentStorageSettings**

  Configures durable writer history and durable reader state.

Variables

- static final int **AUTO_WRITER_DEPTH**

  *A special value used as the default value for **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834).*
- static final **QosPolicyId_t DURABILITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830).*

6.95.1 Detailed Description

This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.

6.95.2 Variable Documentation

6.95.2.1 AUTO_WRITER_DEPTH

```
final int AUTO_WRITER_DEPTH  [static]
```

A special value used as the default value for **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834).

This values resolves to the following:

- **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) if the history kind is com.rti.dds.infrastructure.↩ HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS.
- **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) in the **com.↩ rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) if the history kind is com.rti.dds.infrastructure.↩ HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS.

6.95.2.2 DURABILITY_QOS_POLICY_ID

```
final QosPolicyId_t DURABILITY_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830).

6.96 DURABILITY_SERVICE

Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.

Classes

- class **DurabilityServiceQosPolicy**

  *Various settings to configure the external RTI Persistence Service used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicyKind.↩ DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.*

Variables

- static final **QosPolicyId_t DURABILITY_SERVICE_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DurabilityServiceQosPolicy** (p. 837).*

6.96.1 Detailed Description

Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.

6.96.2 Variable Documentation

6.96.2.1 DURABILITY_SERVICE_QOS_POLICY_ID

```
final QosPolicyId_t DURABILITY_SERVICE_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.DurabilityServiceQosPolicy** (p. 837).

6.97 Time Support

Time and duration types and defines.

Classes

- class **Duration_t**

  Type for duration representation.
- class **Time_t**

  Type for time representation.

6.97.1 Detailed Description

Time and duration types and defines.

6.98 Entity Support

com.rti.dds.infrastructure.Entity (p. 1029), **com.rti.dds.infrastructure.Listener** (p. 1236) and related items.

Classes

- interface **DomainEntity**

  <<***interface***>> *(*p. *156) Abstract base class for all DDS entities except for the* ***com.rti.dds.domain.DomainParticipant*** *(*p. *670).*
- interface **Entity**

  <<***interface***>> *(*p. *156) Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.*
- interface **Listener**

  <<***interface***>> *(*p. *156) Abstract base class for all* ***Listener*** *(*p. *1236) interfaces.*

6.98.1 Detailed Description

com.rti.dds.infrastructure.Entity (p. 1029), **com.rti.dds.infrastructure.Listener** (p. 1236) and related items.

com.rti.dds.infrastructure.Entity (p. 1029) subtypes are created and destroyed by factory objects. With the exception of **com.rti.dds.domain.DomainParticipant** (p. 670), whose factory is **com.rti.dds.domain.DomainParticipant↩ Factory** (p. 761), all **com.rti.dds.infrastructure.Entity** (p. 1029) factory objects are themselves **com.rti.dds.↩ infrastructure.Entity** (p. 1029) subtypes as well.

Important: all **com.rti.dds.infrastructure.Entity** (p. 1029) delete operations are inherently thread-unsafe. The user must take extreme care that a given **com.rti.dds.infrastructure.Entity** (p. 1029) is not destroyed in one thread while being used concurrently (including being deleted concurrently) in another thread. An operation's effect in the presence of the concurrent deletion of the operation's target **com.rti.dds.infrastructure.Entity** (p. 1029) is undefined.

6.99 ENTITY_FACTORY

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.

Classes

- class **EntityFactoryQosPolicy**

  *A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.*

Variables

- static final **QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 1035).*

6.99.1 Detailed Description

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.

6.99.2 Variable Documentation

6.99.2.1 ENTITYFACTORY_QOS_POLICY_ID

```
final QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 1035).

6.100 ENTITY_NAME

<<***extension***>> *(p. 155)* Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

Classes

- class **EntityNameQosPolicy**

  *Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

6.100.1 Detailed Description

<<***extension***>> *(p. 155)* Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

6.101 EVENT

<<***extension***>> *(p. 155)* Configures the internal thread in a DomainParticipant that handles timed events.

Classes

- class **EventQosPolicy**

  Settings for event.

Variables

- static final **QosPolicyId_t EVENT_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.EventQosPolicy** (p. 1044)*

6.101.1 Detailed Description

<<***extension***>> *(p. 155)* Configures the internal thread in a DomainParticipant that handles timed events.

6.101.2 Variable Documentation

6.101.2.1 EVENT_QOS_POLICY_ID

```
final  QosPolicyId_t EVENT_QOS_POLICY_ID  [static]
```

<<***extension***>> *(*p. *155)* Identifier for **com.rti.dds.infrastructure.EventQosPolicy** (p. 1044)

6.102 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Classes

- class **GroupDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (*p. *51) during discovery.*

Variables

- static final **QosPolicyId_t GROUPDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.GroupDataQosPolicy** (*p. *1127).*

6.102.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

6.102.2 Variable Documentation

6.102.2.1 GROUPDATA_QOS_POLICY_ID

```
final  QosPolicyId_t GROUPDATA_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1127).

6.103 GUID Support

<<***extension***>> *(*p. *155)* GUID type and defines.

Classes

- class **GUID_t**

  Type for GUID (Global Unique Identifier) representation.

6.103.1 Detailed Description

<<***extension***>> *(*p. *155)* GUID type and defines.

6.104 HISTORY

Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Classes

- class **HistoryQosPolicy**

  Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.
- class **HistoryQosPolicyKind**

  Kinds of history.

Variables

- static final **QosPolicyId_t HISTORY_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.HistoryQosPolicy*** *(*p. *1144).*

6.104.1 Detailed Description

Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

6.104.2 Variable Documentation

6.104.2.1 HISTORY_QOS_POLICY_ID

```
final  QosPolicyId_t HISTORY_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144).

6.105 LATENCY_BUDGET

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Classes

- class **LatencyBudgetQosPolicy**

  Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Variables

- static final **QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1231).*

6.105.1 Detailed Description

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

6.105.2 Variable Documentation

6.105.2.1 LATENCYBUDGET_QOS_POLICY_ID

```
final  QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1231).

6.106 LIFESPAN

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.

Classes

- class **LifespanQosPolicy**

  *Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.*

Variables

- static final **QosPolicyId_t LIFESPAN_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234).*

6.106.1 Detailed Description

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.

6.106.2 Variable Documentation

6.106.2.1 LIFESPAN_QOS_POLICY_ID

```
final QosPolicyId_t LIFESPAN_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234).

6.107 LIVELINESS

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".

Classes

- class **LivelinessQosPolicy**

  *Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".*
- class **LivelinessQosPolicyKind**

  Kinds of liveliness.

Variables

- static final **QosPolicyId_t LIVELINESS_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243).*

6.107.1 Detailed Description

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".

6.107.2 Variable Documentation

6.107.2.1 LIVELINESS_QOS_POLICY_ID

```
final QosPolicyId_t LIVELINESS_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243).

6.108 LOCATORFILTER

<<***extension***>> *(p. 155)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).

Classes

- class **LocatorFilter_t**

  Specifies the configuration of an individual channel within a MultiChannel DataWriter.
- class **LocatorFilterQosPolicy**

  *The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).*
- class **LocatorFilterSeq**

  Declares IDL `sequence<` ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1258)* `>`.

Variables

- static final **QosPolicyId_t LOCATORFILTER_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.LocatorFilterQosPolicy** (p. 1260)*

6.108.1 Detailed Description

<<***extension***>> *(p. 155)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).

6.108.2 Variable Documentation

6.108.2.1 LOCATORFILTER_QOS_POLICY_ID

```
final  QosPolicyId_t LOCATORFILTER_QOS_POLICY_ID  [static]
```

<<***extension***>> *(*p. *155)* Identifier for **com.rti.dds.infrastructure.LocatorFilterQosPolicy** (p. 1260)

6.109 LOGGING

<<***extension***>> *(*p. *155)* Configures the RTI Connext logging facility.

Classes

- class **LoggingQosPolicy**
  Configures the RTI Connext logging facility.

Variables

- static final **QosPolicyId_t LOGGING_QOS_POLICY_ID**
  <<***extension***>> *(*p. *155) Identifier for **com.rti.dds.infrastructure.LoggingQosPolicy** (*p. *1275)*

6.109.1 Detailed Description

<<***extension***>> *(*p. *155)* Configures the RTI Connext logging facility.

6.109.2 Variable Documentation

6.109.2.1 LOGGING_QOS_POLICY_ID

```
final  QosPolicyId_t LOGGING_QOS_POLICY_ID  [static]
```

<<***extension***>> *(*p. *155)* Identifier for **com.rti.dds.infrastructure.LoggingQosPolicy** (p. 1275)

6.110 MONITORING

Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

Classes

- class **MonitoringDedicatedParticipantSettings**

  *Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.*
- class **MonitoringDistributionSettings**

  Configures the distribution of telemetry data.
- class **MonitoringEventDistributionSettings**

  Configures the distribution of event metrics.
- class **MonitoringLoggingDistributionSettings**

  Configures the distribution of log messages.
- class **MonitoringLoggingForwardingSettings**

  *Configures the forwarding levels of log messages for the different **com.rti.ndds.config.LogFacility** (p. 1265).*
- class **MonitoringMetricSelection**

  This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.
- class **MonitoringMetricSelectionSeq**

  *Declares IDL `sequence` < **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) >*
- class **MonitoringPeriodicDistributionSettings**

  Configures the distribution of periodic metrics.
- class **MonitoringQosPolicy**

  Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.
- class **MonitoringTelemetryData**

  Configures the telemetry data that will be distributed.

Variables

- static final **QosPolicyId_t MONITORING_QOS_POLICY_ID**

  <<***extension***>> (p. 155) *Identifier for **com.rti.dds.infrastructure.MonitoringQosPolicy** (p. 1314)*

6.110.1 Detailed Description

Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

6.110.2 Variable Documentation

6.110.2.1 MONITORING_QOS_POLICY_ID

```
final  QosPolicyId_t MONITORING_QOS_POLICY_ID  [static]
```

<<***extension***>> (p. 155) Identifier for **com.rti.dds.infrastructure.MonitoringQosPolicy** (p. 1314)

6.111 MULTICHANNEL

<<***extension***>> *(p. 155)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Classes

- class **ChannelSettings_t**

  Type used to configure the properties of a channel.
- class **ChannelSettingsSeq**

  Declares IDL `sequence<` ***com.rti.dds.infrastructure.ChannelSettings_t*** *(p. 411)* `>`
- class **MultiChannelQosPolicy**

  Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Variables

- static final **QosPolicyId_t MULTICHANNEL_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.MultiChannelQosPolicy*** *(p. 1318)*

6.111.1 Detailed Description

<<***extension***>> *(p. 155)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

6.111.2 Variable Documentation

6.111.2.1 MULTICHANNEL_QOS_POLICY_ID

```
final  QosPolicyId_t MULTICHANNEL_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318)

6.112 Object Support

<<***extension***>> *(p. 155)* Object related items.

Classes

- interface **Copyable**

  <<***extension***>> (p. *155)* <<***interface***>> (p. *156) Interface for all the user-defined data type classes that support copy.*
- class **ObjectHolder**

  <<***extension***>> (p. *155) Holder of object instance*

6.112.1 Detailed Description

<<***extension***>> (p. *155)* Object related items.

A user-defined type class implements this interface to indicate that the class can be copied. This is typically used in **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) or **com.rti.ndds.example.FooDataReader.↩ read_next_sample** (p. 1078).

6.113 OWNERSHIP

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Classes

- class **OwnershipQosPolicy**

  *Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*
- class **OwnershipQosPolicyKind**

  Kinds of ownership.

Variables

- static final **QosPolicyId_t OWNERSHIP_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.OwnershipQosPolicy** (p. 1342).*

6.113.1 Detailed Description

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

6.113.2 Variable Documentation

6.113.2.1 OWNERSHIP_QOS_POLICY_ID

```
final  QosPolicyId_t OWNERSHIP_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.OwnershipQosPolicy** (p. 1342).

6.114 OWNERSHIP_STRENGTH

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).

Classes

- class **OwnershipStrengthQosPolicy**

  *Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).*

Variables

- static final **QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1348).*

6.114.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).

6.114.2 Variable Documentation

6.114.2.1 OWNERSHIPSTRENGTH_QOS_POLICY_ID

```
final  QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1348).

6.115 Extended Qos Support

<<***extension***>> *(p. 155)* Types and defines used in extended QoS policies.

Modules

- **Thread Settings**

  The properties of a thread of execution. Consult `Platform Notes` *for additional platform specific details.*

Classes

- class **RtpsReliableReaderProtocol_t**

  Qos *(p. 1500) related to reliable reader protocol defined in RTPS.*
- class **RtpsReliableWriterProtocol_t**

  QoS related to the reliable writer protocol defined in RTPS.

6.115.1 Detailed Description

<<***extension***>> *(p. 155)* Types and defines used in extended QoS policies.

6.116 PARTITION

Set of strings that introduces logical partitions in **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.↩ publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entities.

Classes

- class **PartitionQosPolicy**

  Set of strings that introduces logical partitions in ***com.rti.dds.domain.DomainParticipant*** *(p. 670),* ***com.rti.dds.↩ publication.Publisher*** *(p. 1466), or* ***com.rti.dds.subscription.Subscriber*** *(p. 1730) entities.*

Variables

- static final **QosPolicyId_t PARTITION_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.PartitionQosPolicy*** *(p. 1365).*

6.116.1 Detailed Description

Set of strings that introduces logical partitions in **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.↩ publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entities.

6.116.2 Variable Documentation

6.116.2.1 PARTITION_QOS_POLICY_ID

```
final QosPolicyId_t PARTITION_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365).

6.117 PRESENTATION

Specifies how the samples representing changes to data instances are presented to a subscribing application.

Classes

- class **PresentationQosPolicy**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.
- class **PresentationQosPolicyAccessScopeKind**

  Kinds of presentation "access scope".

Variables

- static final **QosPolicyId_t PRESENTATION_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1379).*

6.117.1 Detailed Description

Specifies how the samples representing changes to data instances are presented to a subscribing application.

6.117.2 Variable Documentation

6.117.2.1 PRESENTATION_QOS_POLICY_ID

```
final QosPolicyId_t PRESENTATION_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1379).

6.118 PROFILE

<<***extension***>> *(p. 155)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Classes

- class **ProfileQosPolicy**

  Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Variables

- static final **QosPolicyId_t PROFILE_QOS_POLICY_ID**

  *<<**extension**>> (p. 155) Identifier for **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1393)*

6.118.1 Detailed Description

*<<**extension**>> (p. 155)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

6.118.2 Variable Documentation

6.118.2.1 PROFILE_QOS_POLICY_ID

```
final  QosPolicyId_t PROFILE_QOS_POLICY_ID  [static]
```

*<<**extension**>> (p. 155)* Identifier for **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1393)

6.119 PROPERTY

*<<**extension**>> (p. 155)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Classes

- class **Property_t**

  Properties are name/value pairs objects.
- class **PropertyQosPolicy**

  Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.
- class **PropertyQosPolicyHelper**

  Policy helpers that facilitate management of the properties in the input policy.
- class **PropertyQosPolicyMutability**

  Determines if, and when, the value of a property can change.
- class **PropertySeq**

  *Declares IDL `sequence` < **com.rti.dds.infrastructure.Property_t** (p. 1395) >*

6.119.1 Detailed Description

<<***extension***>> *(p. 155)* Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

RTI Connext will automatically set some system properties in the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with a **com.rti.dds.domain.DomainParticipantQos** (p. 795). See **System Properties** (p. 119) for additional details.

6.119.2 This code contains trade secrets of Real-Time Innovations, Inc.

RTI Connext will automatically set some system properties in the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with a **com.rti.dds.domain.DomainParticipantQos** (p. 795). See **System Properties** (p. 119) for additional details.

6.120 PUBLISH_MODE

<<***extension***>> *(p. 155)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Classes

- interface **PublicationPriority**
- class **PublishModeQosPolicy**

  Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.
- class **PublishModeQosPolicyKind**

  Kinds of publishing mode.

Variables

- static final int **UNDEFINED**

  *Initializer value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).*
- static final int **AUTOMATIC**

  *Constant value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).*
- static final **QosPolicyId_t** **PUBLISHMODE_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)*

6.120.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

6.120.2 Variable Documentation

6.120.2.1 UNDEFINED

```
final int UNDEFINED [static]
```

Initializer value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the lowest possible value. For muilti-channel data writers, if either the data writer or channel priority is NOT set to this value, then the publication priority of the entity will be set to the defined value.

6.120.2.2 AUTOMATIC

```
final int AUTOMATIC [static]
```

Constant value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).

When assigned this value, the publication priority of the data writer, or channel of a multi-channel data writer, will be set to the largest priority value of any sample currently queued for publication by the data writer or data writer channel.

6.120.2.3 PUBLISHMODE_QOS_POLICY_ID

```
final QosPolicyId_t PUBLISHMODE_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496)

6.121 QoS Policies

Quality of Service (QoS) policies.

Modules

- **ASYNCHRONOUS_PUBLISHER**

  *<<**extension**>> (p. 155) Specifies the asynchronous publishing settings of the **com.rti.dds.publication.Publisher** (p. 1466) instances.*
- **AVAILABILITY**

  *<<**extension**>> (p. 155) Configures the availability of data.*
- **BATCH**

  *<<**extension**>> (p. 155) Batch QoS policy used to enable batching in **com.rti.dds.publication.DataWriter** (p. 553) instances.*
- **DATABASE**

  *<<**extension**>> (p. 155) Various threads and resource limits settings used by RTI Connext to control its internal database.*
- **DATA_READER_PROTOCOL**

  *<<**extension**>> (p. 155) Specifies the DataReader-specific protocol QoS.*
- **DATA_READER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 155) Various settings that configure how DataReaders allocate and use physical memory for internal resources.*
- **DATA_REPRESENTATION**

  *A list of data representations and compression methods supported by a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450).*
- **DATA_TAG**

  Stores (name, value) pairs that can be used to determine access permissions.
- **DATA_WRITER_PROTOCOL**

  *<<**extension**>> (p. 155) Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501), this QoS policy configures the DDS on-the-network protocol (RTPS).*
- **DATA_WRITER_RESOURCE_LIMITS**

  *<<**extension**>> (p. 155) Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.*
- **DEADLINE**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.
- **DESTINATION_ORDER**

  *Controls the criteria used to determine the logical order among changes made by **com.rti.dds.publication.Publisher** (p. 1466) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1807) and key).*
- **DISCOVERY_CONFIG**

  *<<**extension**>> (p. 155) Specifies the discovery configuration QoS.*
- **DISCOVERY**

  *<<**extension**>> (p. 155) Specifies the attributes required to discover participants in the domain.*
- **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**

  *<<**extension**>> (p. 155) Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*
- **DURABILITY**

  *This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.*
- **DURABILITY_SERVICE**

  *Various settings to configure the external RTI Persistence Service used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.*

- **ENTITY_FACTORY**

  *A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.*

- **ENTITY_NAME**

  *<<**extension**>> (p. 155) Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.*

- **EVENT**

  *<<**extension**>> (p. 155) Configures the internal thread in a DomainParticipant that handles timed events.*

- **GROUP_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*

- **HISTORY**

  Specifies the behavior of RTI Connext in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

- **LATENCY_BUDGET**

  Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

- **LIFESPAN**

  *Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.*

- **LIVELINESS**

  *Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".*

- **LOCATORFILTER**

  *<<**extension**>> (p. 155) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).*

- **LOGGING**

  *<<**extension**>> (p. 155) Configures the RTI Connext logging facility.*

- **MONITORING**

  Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

- **MULTICHANNEL**

  *<<**extension**>> (p. 155) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.*

- **OWNERSHIP**

  *Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*

- **OWNERSHIP_STRENGTH**

  *Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).*

- **Extended Qos Support**

  *<<**extension**>> (p. 155) Types and defines used in extended QoS policies.*

- **PARTITION**

  *Set of strings that introduces logical partitions in **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entities.*

- **PRESENTATION**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.

- **PROFILE**

  *<<**extension**>> (p. 155) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.*

- **PROPERTY**

*<<**extension**>> (p. 155) Stores (name, value) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.*

- **PUBLISH_MODE**

  *<<**extension**>> (p. 155) Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.*

- **READER_DATA_LIFECYCLE**

  Controls how a DataReader manages the lifecycle of the data that it has received.

- **RECEIVER_POOL**

  *<<**extension**>> (p. 155) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).*

- **RELIABILITY**

  Indicates the level of reliability offered/requested by RTI Connext.

- **RESOURCE_LIMITS**

  Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

- **SERVICE**

  *<<**extension**>> (p. 155) Service QoS policy is used to indicate what kind of service is associated with the DDS entity.*

- **SYSTEM_RESOURCE_LIMITS**

  *<<**extension**>> (p. 155) Configures DomainParticipant-independent resources used by RTI Connext.*

- **TIME_BASED_FILTER**

  *Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.*

- **TOPIC_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*

- **TOPIC_QUERY_DISPATCH**

  *Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish historical samples.*

- **TRANSPORT_BUILTIN**

  *<<**extension**>> (p. 155) Specifies which built-in transports are used.*

- **TRANSPORT_MULTICAST_MAPPING**

  *<<**extension**>> (p. 155) Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.*

- **TRANSPORT_MULTICAST**

  *<<**extension**>> (p. 155) Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.*

- **TRANSPORT_PRIORITY**

  This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

- **TRANSPORT_SELECTION**

  *<<**extension**>> (p. 155) Specifies the physical transports that a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) may use to send or receive data.*

- **TRANSPORT_UNICAST**

  *<<**extension**>> (p. 155) Specifies a subset of transports and a port number that can be used by an Entity to receive data.*

- **TYPE_CONSISTENCY_ENFORCEMENT**

  Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

- **TYPESUPPORT**

*<<**extension**>> (p. 155) Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*

- **USER_DATA**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*

- **WIRE_PROTOCOL**

  *<<**extension**>> (p. 155) Specifies the wire protocol related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 670).*

- **WRITER_DATA_LIFECYCLE**

  Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

Classes

- class **Qos**

  An abstract base class for all QoS types.

- class **QosPolicy**

  The base class for all QoS policies.

- class **QosPolicyCount**

  *Type to hold a counter for a **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504).*

- class **QosPolicyCountSeq**

  *Declares IDL `sequence` < **com.rti.dds.infrastructure.QosPolicyCount** (p. 1502) >*

- class **QosPolicyId_t**

  Type to identify `QosPolicies`.

- class **QosPrintFormat**

  A collection of attributes used to configure how a QoS appears when printed.

6.121.1 Detailed Description

Quality of Service (QoS) policies.

Data Distribution Service (DDS) relies on the use of QoS. A QoS is a set of characteristics that controls some aspect of the behavior of DDS. A QoS is comprised of individual QoS policies (objects conceptually deriving from an *abstract* `QosPolicy` class).

"Supported QoS policies"

The QosPolicy provides the basic mechanism for an application to specify quality of service parameters. It has an attribute name that is used to uniquely identify each `QosPolicy`.

QosPolicy implementation is comprised of a name, an ID, and a type. The type of a `QosPolicy` value may be atomic, such as an integer or float, or compound (a structure). Compound types are used whenever multiple parameters must be set coherently to define a consistent value for a QosPolicy.

QoS (i.e., a list of `QosPolicy` objects) may be associated with all **com.rti.dds.infrastructure.Entity** (p. 1029) objects in the system such as **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.dds.domain.DomainParticipant** (p. 670).

6.121.2 Specifying QoS on entities

QosPolicies can be set programmatically when an **com.rti.dds.infrastructure.Entity** (p. 1029) is created, or modified with the **com.rti.dds.infrastructure.Entity** (p. 1029)'s **set_qos (abstract)** (p. 1030) method.

QosPolicies can also be configured from XML resources (files, strings). With this approach, you can change the QoS without recompiling the application. For more information, see **Configuring QoS Profiles with XML** (p. 120).

To customize a **com.rti.dds.infrastructure.Entity** (p. 1029)'s QoS before creating the entity, the correct pattern is:

- First, initialize a QoS object with the appropriate INITIALIZER constructor.
- Call the relevant get_<entity>_default_qos() method.
- Modify the QoS values as desired.
- Finally, create the entity.

Each `QosPolicy` is treated independently from the others. This approach has the advantage of being very extensible. However, there may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified via the **set_qos (abstract)** (p. 1030) operation, or when the **com.rti.dds.infrastructure.Entity** (p. 1029) is created.

When a policy is changed after being set to a given value, it is not required that the new value be applied instantaneously; RTI Connext is allowed to apply it after a transition phase. In addition, some `QosPolicy` have immutable semantics, meaning that they can only be specified either at **com.rti.dds.infrastructure.Entity** (p. 1029) creation time or else prior to calling the **com.rti.dds.infrastructure.Entity.enable** (p. 1032) operation on the entity.

Each **com.rti.dds.infrastructure.Entity** (p. 1029) can be configured with a list of `QosPolicy` objects. However, not all QosPolicies are supported by each **com.rti.dds.infrastructure.Entity** (p. 1029). For instance, a **com.rti.dds.↩ domain.DomainParticipant** (p. 670) supports a different set of QosPolicies than a **com.rti.dds.topic.Topic** (p. 1807) or a **com.rti.dds.publication.Publisher** (p. 1466).

Additional properties that are not exposed through the formal QoS policies can also be set for an **com.rti.dds.↩ infrastructure.Entity** (p. 1029) via the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438). These properties are described in the `Property Reference Guide`.

6.121.3 QoS compatibility

In several cases, for communications to occur properly (or efficiently), a QosPolicy on the publisher side must be compatible with a corresponding policy on the subscriber side. For example, if a **com.rti.dds.subscription.Subscriber** (p. 1730) requests to receive data reliably while the corresponding **com.rti.dds.publication.Publisher** (p. 1466) defines a best-effort policy, communication will not happen as requested.

To address this issue and maintain the desirable decoupling of publication and subscription as much as possible, the QosPolicy specification follows the **subscriber-requested, publisher-offered pattern**.

In this pattern, the subscriber side can specify a "requested" value for a particular `QosPolicy`. The publisher side specifes an "offered" value for that `QosPolicy`. RTI Connext will then determine whether the value requested by the subscriber side is compatible with what is offered by the publisher side. If the two policies are compatible, then communication will be established. If the two policies are not compatible, RTI Connext will not establish communications between the two **com.rti.dds.infrastructure.Entity** (p. 1029) objects and will record this fact by means of the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS on the publisher end and com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS on the subscriber end. The application can detect this fact by means of a **com.rti.dds.infrastructure.Listener** (p. 1236) or a **com.rti.↩ dds.infrastructure.Condition** (p. 429).

The following **properties** are defined on a QosPolicy.

- **RxO (**p. **256) property**
  The QosPolicy objects that need to be set in a compatible manner between the **publisher** and **subscriber** end are indicated by the setting of the **RxO** (p. 256) property:
  - **RxO** (p. 256) = **YES**
    indicates that the policy can be set both at the publishing and subscribing ends and the values must be set in a compatible manner. In this case the compatible values are explicitly defined.
  - **RxO** (p. 256) = **NO**
    indicates that the policy can be set both at the publishing and subscribing ends but the two settings are independent. That is, all combinations of values are compatible.
  - **RxO** (p. 256) = **N/A**
    indicates that the policy can only be specified at either the publishing or the subscribing end, but not at both ends. So compatibility does not apply.
- **Changeable (**p. **256)property**
  Determines whether a QosPolicy can be changed.
  - **NO (**p. **256)** -- policy can only be specified at **com.rti.dds.infrastructure.Entity** (p. 1029) creation time.
  - **UNTIL ENABLE (**p. **256)** -- policy can only be changed before the **com.rti.dds.infrastructure.Entity** (p. 1029) is enabled.
  - **YES (**p. **256)** -- policy can be changed at any time.

6.122 READER_DATA_LIFECYCLE

Controls how a DataReader manages the lifecycle of the data that it has received.

Classes

- class **ReaderDataLifecycleQosPolicy**
  Controls how a DataReader manages the lifecycle of the data that it has received.

Variables

- static final **QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID**
  *Identifier for **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy** (*p. *1520).*

6.122.1 Detailed Description

Controls how a DataReader manages the lifecycle of the data that it has received.

6.122.2 Variable Documentation

6.122.2.1 READERDATALIFECYCLE_QOS_POLICY_ID

```
final QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy** (p. 1520).

6.123 RECEIVER_POOL

<<***extension***>> *(p. 155)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Classes

- class **ReceiverPoolQosPolicy**

  Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Variables

- static final **QosPolicyId_t RECEIVERPOOL_QOS_POLICY_ID**

  <<***extension***>> *(p. 155)* Identifier for ***com.rti.dds.infrastructure.ReceiverPoolQosPolicy*** *(p. 1523)*
- static final int **LENGTH_AUTO**

  A special value indicating that the actual value will be automatically resolved.

6.123.1 Detailed Description

<<***extension***>> *(p. 155)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

6.123.2 Variable Documentation

6.123.2.1 RECEIVERPOOL_QOS_POLICY_ID

```
final QosPolicyId_t RECEIVERPOOL_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523)

6.123.2.2 LENGTH_AUTO

```
final int LENGTH_AUTO  [static]
```

A special value indicating that the actual value will be automatically resolved.

6.124 RELIABILITY

Indicates the level of reliability offered/requested by RTI Connext.

Classes

- class **InstanceStateConsistencyKind**

  <<***extension***>> (p. *155) Whether instance state consistency is enabled.*
- class **PersistentSynchronizationKind**

  <<***extension***>> (p. *155) Whether instance state consistency is enabled.*
- class **ReliabilityQosPolicy**

  Indicates the level of reliability offered/requested by RTI Connext.
- class **ReliabilityQosPolicyAcknowledgmentModeKind**

  <<***extension***>> (p. *155) Kinds of acknowledgment.*
- class **ReliabilityQosPolicyKind**

  Kinds of reliability.

Variables

- static final **QosPolicyId_t RELIABILITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526).*

6.124.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Connext.

6.124.2 Variable Documentation

6.124.2.1 RELIABILITY_QOS_POLICY_ID

```
final  QosPolicyId_t RELIABILITY_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526).

6.125 RESOURCE_LIMITS

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Classes

- class **ResourceLimitsQosPolicy**

  Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Variables

- static final **QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).*
- static final int **LENGTH_UNLIMITED**

  A special value indicating an unlimited quantity.

6.125.1 Detailed Description

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

6.125.2 Variable Documentation

6.125.2.1 RESOURCELIMITS_QOS_POLICY_ID

```
final QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

6.125.2.2 LENGTH_UNLIMITED

```
final int LENGTH_UNLIMITED [static]
```

A special value indicating an unlimited quantity.

Referenced by **Requester< TReq, TRep >.readReplies()**, **Replier< TReq, TRep >.readRequests()**, **Requester< TReq, TRep >.receiveReplies()**, **Replier< TReq, TRep >.receiveRequests()**, **Requester< TReq, TRep >.takeReplies()**, and **Replier< TReq, TRep >.takeRequests()**.

6.126 Return Codes

Types of return codes.

Classes

- class **RETCODE_ALREADY_DELETED**

  The object target of this operation has already been deleted.
- class **RETCODE_BAD_PARAMETER**

  Illegal parameter value.
- class **RETCODE_ERROR**

  Generic, unspecified error.
- class **RETCODE_ILLEGAL_OPERATION**

  The operation was called under improper circumstances.
- class **RETCODE_IMMUTABLE_POLICY**

  Application attempted to modify an immutable QoS policy.
- class **RETCODE_INCONSISTENT_POLICY**

  Application specified a set of QoS policies that are not consistent with each other.
- class **RETCODE_NO_DATA**

  Indicates a transient situation where the operation did not return any data but there is no inherent error.
- class **RETCODE_NOT_ALLOWED_BY_SECURITY**

  An operation on the DDS API that fails because the security plugins do not allow it.
- class **RETCODE_NOT_ENABLED**

  *Operation invoked on a **com.rti.dds.infrastructure.Entity*** (p. *1029) that is not yet enabled.*
- class **RETCODE_OUT_OF_RESOURCES**

  RTI Connext ran out of the resources needed to complete the operation.
- class **RETCODE_PRECONDITION_NOT_MET**

  A pre-condition for the operation was not met.
- class **RETCODE_TIMEOUT**

  The operation timed out.
- class **RETCODE_UNSUPPORTED**

  Unsupported operation. Only returned by operations that are unsupported.

6.126.1 Detailed Description

Types of return codes.

6.126.2 Standard Return Codes

Any void operation that documents that it may throw an exception of type may throw exactly **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595) or **com.rti.dds.infrastructure.RETCODE_ILLEGAL_OPERATION** (p. 1595). Any such operation that takes one or more input parameters may additionally throw the subclass **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). Any operation on an object created from any of the factories may additionally throw the subclass **com.rti.dds.infrastructure.RETCODE_ALREADY_DELETED** (p. 1594). Any operation that is stated as optional may additionally throw the subclass **com.rti.dds.infrastructure.RETCODE_UNSUPPORTED** (p. 1599). com.rti.dds.infrastructure.RETCODE_OK **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595) or **com.rti.dds.infrastructure.RETCODE_ILLEGAL_OPERATION** (p. 1595). Any operation that takes one or more input parameters may additionally return **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). Any operation on an object created from any of the factories may additionally return **com.rti.dds.infrastructure.RETCODE_ALREADY_DELETED** (p. 1594). Any operation that is stated as optional may additionally return **com.rti.dds.infrastructure.RETCODE_UNSUPPORTED** (p. 1599).

Thus, the standard return codes are:

- com.rti.dds.infrastructure.RETCODE_OK
- **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595)
- **com.rti.dds.infrastructure.RETCODE_ILLEGAL_OPERATION** (p. 1595)
- **com.rti.dds.infrastructure.RETCODE_ALREADY_DELETED** (p. 1594)
- **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594)
- **com.rti.dds.infrastructure.RETCODE_UNSUPPORTED** (p. 1599)

Operations that may throw any other exception type will state so explicitly.

6.127 Sequence Number Support

<<***extension***>> *(p. 155)* Sequence number type and defines.

Classes

- class **SequenceNumber_t**
  Type for sequence number representation.

6.127.1 Detailed Description

<<***extension***>> *(p. 155)* Sequence number type and defines.

6.128 SERVICE

<<***extension***>> *(p. 155)* Service QoS policy is used to indicate what kind of service is associated with the DDS entity.

Classes

- class **ServiceQosPolicy**
  Service associated with a DDS entity.
- class **ServiceQosPolicyKind**
  Kinds of service.

Variables

- static final **QosPolicyId_t SERVICE_QOS_POLICY_ID**
  *<<**extension**>> (p. 155) Identifier for **com.rti.dds.infrastructure.ServiceQosPolicy** (p. 1672)*

6.128.1 Detailed Description

*<<**extension**>> (p. 155)* Service QoS policy is used to indicate what kind of service is associated with the DDS entity.

6.128.2 Variable Documentation

6.128.2.1 SERVICE_QOS_POLICY_ID

```
final  QosPolicyId_t SERVICE_QOS_POLICY_ID  [static]
```

*<<**extension**>> (p. 155)* Identifier for **com.rti.dds.infrastructure.ServiceQosPolicy** (p. 1672)

6.129 Status Kinds

Kinds of communication status.

Classes

- class **StatusKind**
  Type for status kinds.

Variables

- static final int **STATUS_MASK_NONE**
  No bits are set.
- static final int **STATUS_MASK_ALL**
  All bits are set.

6.129.1 Detailed Description

Kinds of communication status.

Entity:

com.rti.dds.infrastructure.Entity (p. 1029)

QoS:

QoS Policies (p. 250)

Listener:

com.rti.dds.infrastructure.Listener (p. 1236)

Each concrete **com.rti.dds.infrastructure.Entity** (p. 1029) is associated with a set of Status objects whose value represents the communication status of that entity. Each status value can be accessed with a corresponding method on the **com.rti.dds.infrastructure.Entity** (p. 1029).

When these status values change, the corresponding **com.rti.dds.infrastructure.StatusCondition** (p. 1699) objects are activated and the proper **com.rti.dds.infrastructure.Listener** (p. 1236) objects are invoked to asynchronously inform the application.

An application is notified of communication status by means of the **com.rti.dds.infrastructure.Listener** (p. 1236) or the **com.rti.dds.infrastructure.WaitSet** (p. 1973) / **com.rti.dds.infrastructure.Condition** (p. 429) mechanism. The two mechanisms may be combined in the application (e.g., using **com.rti.dds.infrastructure.WaitSet** (p. 1973) (s) / **com.↩ rti.dds.infrastructure.Condition** (p. 429) (s) to access the data and **com.rti.dds.infrastructure.Listener** (p. 1236) (s) to be warned asynchronously of erroneous communication statuses).

It is likely that the application will choose one or the other mechanism for each particular communication status (not both). However, if both mechanisms are enabled, then the **com.rti.dds.infrastructure.Listener** (p. 1236) mechanism is used first and then the **com.rti.dds.infrastructure.WaitSet** (p. 1973) objects are signalled.

The statuses may be classified into:

- *read communication statuses*: i.e., those that are related to arrival of data, namely com.rti.dds.infrastructure.↩ StatusKind.StatusKind.DATA_ON_READERS_STATUS and com.rti.dds.infrastructure.StatusKind.StatusKind.↩ DATA_AVAILABLE_STATUS.

- *plain communication statuses*: i.e., all the others.

Read communication statuses are treated slightly differently than the others because they don't change independently. In other words, at least two changes will appear at the same time (com.rti.dds.infrastructure.StatusKind.StatusKind.↩ DATA_ON_READERS_STATUS and com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS) and even several of the last kind may be part of the set. This 'grouping' has to be communicated to the application.

For each plain communication status, there is a corresponding structure to hold the status value. These values contain the information related to the change of status, as well as information related to the statuses themselves (e.g., contains cumulative counts).

"Status Values"

6.129.2 Changes in Status

Associated with each one of an **com.rti.dds.infrastructure.Entity** (p. 1029)'s communication status is a logical `StatusChangedFlag`. This flag indicates whether that particular communication status has changed since the last time the status was read by the application. The way the status changes is slightly different for the Plain Communication Status and the Read Communication status.

"\p StatusChangedFlag indicates if status has changed"

6.129.2.1 Changes in plain communication status

For the plain communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever the plain communication status changes and it is reset to com.rti.dds.infrastructure.false each time the application accesses the plain communication status via the proper `get_<plain communication status>()` operation on the **com.rti.dds.infrastructure.Entity** (p. 1029).

The communication status is also reset to FALSE whenever the associated listener operation is called as the listener implicitly accesses the status which is passed as a parameter to the operation. The fact that the status is reset prior to calling the listener means that if the application calls the get_<plain communication status> from inside the listener it will see the status already reset.

An exception to this rule is when the associated listener is the 'nil' listener. The 'nil' listener is treated as a NOOP and the act of calling the 'nil' listener does not reset the communication status.

For example, the value of the `StatusChangedFlag` associated with the com.rti.dds.infrastructure.StatusKind.↩ StatusKind.REQUESTED_DEADLINE_MISSED_STATUS will become TRUE each time new deadline occurs (which increases **com.rti.dds.subscription.RequestedDeadlineMissedStatus.total_count** (p. 1560)). The value changes to FALSE when the application accesses the status via the corresponding **com.rti.dds.subscription.DataReader.get_↩ requested_deadline_missed_status** (p. 461) method on the proper Entity

"Changes in \p StatusChangedFlag for plain communication status"

6.129.2.2 Changes in read communication status

For the read communication status, the `StatusChangedFlag` flag is initially set to FALSE. The `Status↩ ChangedFlag` becomes TRUE when either a data-sample arrives or else the **com.rti.dds.subscription.ViewState↩ Kind** (p. 1970), **com.rti.dds.subscription.SampleStateKind** (p. 1663), or **com.rti.dds.subscription.InstanceState↩ Kind** (p. 1160) of any existing sample changes for any reason other than a call to **com.rti.ndds.example.FooData↩ Reader.read** (p. 1069), **com.rti.ndds.example.FooDataReader.take** (p. 1071) or their variants. Specifically any of the following events will cause the `StatusChangedFlag` to become TRUE:

- The arrival of new data.
- A change in the **com.rti.dds.subscription.InstanceStateKind** (p. 1160) of a contained instance. This can be caused by either:
  - The arrival of the notification that an instance has been disposed by:
    - the **com.rti.dds.publication.DataWriter** (p. 553) that owns it if **OWNERSHIP** (p. 244) QoS kind= **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1348)

    * or by any **com.rti.dds.publication.DataWriter** (p. 553) if **OWNERSHIP** (p. 244) QoS kind= **com.rti.dds.infrastructure.OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS** (p. 1347)
  - The loss of liveliness of the **com.rti.dds.publication.DataWriter** (p. 553) of an instance for which there is no other **com.rti.dds.publication.DataWriter** (p. 553).
  - The arrival of the notification that an instance has been unregistered by the only **com.rti.dds.publication.DataWriter** (p. 553) that is known to be writing the instance.

Depending on the kind of `StatusChangedFlag`, the flag transitions to FALSE again as follows:

- The com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS `StatusChangedFlag` becomes FALSE when either the corresponding listener operation (on_data_available) is called or the read or take operation (or their variants) is called on the associated **com.rti.dds.subscription.DataReader** (p. 450).
- The com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS `StatusChangedFlag` becomes FALSE when any of the following events occurs:
  - The corresponding listener operation (on_data_on_readers) is called.
  - The on_data_available listener operation is called on any **com.rti.dds.subscription.DataReader** (p. 450) belonging to the **com.rti.dds.subscription.Subscriber** (p. 1730).
  - The read or take operation (or their variants) is called on any **com.rti.dds.subscription.DataReader** (p. 450) belonging to the **com.rti.dds.subscription.Subscriber** (p. 1730).

"Changes in \p StatusChangedFlag for read communication status"

See also

com.rti.dds.infrastructure.Listener (p. 1236)

com.rti.dds.infrastructure.WaitSet (p. 1973), **com.rti.dds.infrastructure.Condition** (p. 429)

6.129.3 Variable Documentation

6.129.3.1 STATUS_MASK_NONE

```
final int STATUS_MASK_NONE  [static]
```

No bits are set.

6.129.3.2 STATUS_MASK_ALL

```
final int STATUS_MASK_ALL  [static]
```

All bits are set.

6.130 SYSTEM_RESOURCE_LIMITS

<<***extension***>> *(p. 155)* Configures DomainParticipant-independent resources used by RTI Connext.

Classes

- class **SystemResourceLimitsQosPolicy**

  <<***extension***>> *(p. 155) Configures **com.rti.dds.domain.DomainParticipant** (p. 670)-independent resources used by RTI Connext. Mainly used to change the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 670) entities that can be created within a single process (address space).*

Variables

- static final **QosPolicyId_t SYSTEMRESOURCELIMITS_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy** (p. 1782)*

6.130.1 Detailed Description

<<***extension***>> *(p. 155)* Configures DomainParticipant-independent resources used by RTI Connext.

6.130.2 Variable Documentation

6.130.2.1 SYSTEMRESOURCELIMITS_QOS_POLICY_ID

```
final  QosPolicyId_t SYSTEMRESOURCELIMITS_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy** (p. 1782)

6.131 Thread Settings

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

Classes

- class **ThreadSettings_t**

  The properties of a thread of execution.
- class **ThreadSettingsCpuRotationKind**

  *Determines how **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794) affects processor affinity for thread-related QoS policies that apply to multiple threads.*
- class **ThreadSettingsKind**

  A collection of flags used to configure threads of execution.

Variables

- static final int **THREAD_SETTINGS_KIND_MASK_DEFAULT**

  The mask of default thread options.

6.131.1 Detailed Description

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

6.131.2 Variable Documentation

6.131.2.1 THREAD_SETTINGS_KIND_MASK_DEFAULT

```
final int THREAD_SETTINGS_KIND_MASK_DEFAULT  [static]
```

The mask of default thread options.

6.132 TIME_BASED_FILTER

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.

Classes

- class **TimeBasedFilterQosPolicy**

  *Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.*

Variables

- static final **QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy** (p. 1804).*

6.132.1 Detailed Description

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.

6.132.2 Variable Documentation

6.132.2.1 TIMEBASEDFILTER_QOS_POLICY_ID

```
final QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy** (p. 1804).

6.133 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Classes

- class **TopicDataQosPolicy**
  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*

Variables

- static final **QosPolicyId_t TOPICDATA_QOS_POLICY_ID**
  *Identifier for **com.rti.dds.infrastructure.TopicDataQosPolicy** (p. 1819).*

6.133.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

6.133.2 Variable Documentation

6.133.2.1 TOPICDATA_QOS_POLICY_ID

```
final QosPolicyId_t TOPICDATA_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.TopicDataQosPolicy** (p. 1819).

6.134 TOPIC_QUERY_DISPATCH

Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish historical samples.

Classes

- class **TopicQueryDispatchQosPolicy**

  *Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish samples in response to a **com.rti.dds.subscription.TopicQuery** (p. 1830).*

Variables

- static final **QosPolicyId_t TOPICQUERYDISPATCH_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy** (p. 1833)*

6.134.1 Detailed Description

Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish historical samples.

6.134.2 Variable Documentation

6.134.2.1 TOPICQUERYDISPATCH_QOS_POLICY_ID

```
final  QosPolicyId_t TOPICQUERYDISPATCH_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy** (p. 1833)

6.135 TRANSPORT_BUILTIN

<<***extension***>> *(p. 155)* Specifies which built-in transports are used.

Classes

- class **TransportBuiltinKind**

  Built-in transport kind.
- class **TransportBuiltinQosPolicy**

  Specifies which built-in transports are used.

Variables

- static final **QosPolicyId_t TRANSPORTBUILTIN_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy*** *(p. 1843)*
- static final String **UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport: "builtin.udpv4".
- static final String **SHMEM_ALIAS**

  Alias name for the shared memory built-in transport: "builtin.shmem".
- static final String **UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport: "builtin.udpv6".
- static final String **UDPv4_WAN_ALIAS**

  Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".
- static final int **MASK_NONE**

  None of the built-in transports will be registered automatically when the ***com.rti.dds.domain.DomainParticipant*** *(p. 670) is enabled.*
- static final int **MASK_DEFAULT**

  The default value of ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask*** *(p. 1844).*
- static final int **MASK_ALL**

  All the available built-in transports are registered automatically when the ***com.rti.dds.domain.DomainParticipant*** *(p. 670) is enabled.*

6.135.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies which built-in transports are used.

See also

Changing the automatically registered built-in transports (p. 138)

6.135.2 Variable Documentation

6.135.2.1 TRANSPORTBUILTIN_QOS_POLICY_ID

```
final  QosPolicyId_t TRANSPORTBUILTIN_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843)

6.135.2.2 UDPv4_ALIAS

```
final String UDPv4_ALIAS  [static]
```

Alias name for the UDPv4 built-in transport: "builtin.udpv4".

6.135.2.3 SHMEM_ALIAS

```
final String SHMEM_ALIAS  [static]
```

Alias name for the shared memory built-in transport: "builtin.shmem".

6.135.2.4 UDPv6_ALIAS

```
final String UDPv6_ALIAS  [static]
```

Alias name for the UDPv6 built-in transport: "builtin.udpv6".

6.135.2.5 UDPv4_WAN_ALIAS

```
final String UDPv4_WAN_ALIAS  [static]
```

Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".

6.135.2.6 MASK_NONE

```
final int MASK_NONE  [static]
```

None of the built-in transports will be registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.

The user must explictly register transports using com.rti.ndds.transport.TransportSupport.register_transport.

See also

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.135.2.7 MASK_DEFAULT

```
final int MASK_DEFAULT  [static]
```

The default value of **com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask** (p. 1844).

The set of builtin transport plugins that will be automatically registered with the participant by default. The user can register additional transports using com.rti.ndds.transport.TransportSupport.register_transport.

[default] [UDPv4|Shmem]

See also

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.135.2.8 MASK_ALL

```
final int MASK_ALL  [static]
```

All the available built-in transports are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.

See also

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.136 TRANSPORT_MULTICAST_MAPPING

<<***extension***>> *(p. 155)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

<<***extension***>> *(p. 155)* Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

6.137 TRANSPORT_MULTICAST

<<***extension***>> *(p. 155)* Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **com.rti.dds.↩ domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

Classes

- class **TransportMulticastMappingQosPolicy**

  *Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.*
- class **TransportMulticastQosPolicy**

  *Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.*
- class **TransportMulticastQosPolicyKind**

  Transport Multicast Policy Kind.

Variables

- static final **QosPolicyId_t TRANSPORTMULTICAST_QOS_POLICY_ID**

  *<<**extension**>> (p. 155) Identifier for **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853)*
- static final **TransportMulticastQosPolicyKind AUTOMATIC_TRANSPORT_MULTICAST_QOS**

  Transport Multicast Policy Kind.
- static final **TransportMulticastQosPolicyKind UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** = new **TransportMulticastQosPolicyKind**("UNICAST_ONLY_TRANSPORT_MULTICAST_QOS", 1)

  Transport Multicast Policy Kind.

6.137.1 Detailed Description

*<<**extension**>> (p. 155)* Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

6.137.2 Variable Documentation

6.137.2.1 TRANSPORTMULTICAST_QOS_POLICY_ID

```
final QosPolicyId_t TRANSPORTMULTICAST_QOS_POLICY_ID [static]
```

*<<**extension**>> (p. 155)* Identifier for **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853)

6.137.2.2 AUTOMATIC_TRANSPORT_MULTICAST_QOS

```
final TransportMulticastQosPolicyKind AUTOMATIC_TRANSPORT_MULTICAST_QOS [static]
```

Initial value:

```
=
                    new TransportMulticastQosPolicyKind(
                                            "AUTOMATIC_TRANSPORT_MULTICAST_QOS", 0)
```

Transport Multicast Policy Kind.

See also

com.rti.dds.infrastructure.TransportMulticastQosPolicy.AUTOMATIC_TRANSPORT_MULTICAST_QOS

6.137.2.3 UNICAST_ONLY_TRANSPORT_MULTICAST_QOS

```
final TransportMulticastQosPolicyKind UNICAST_ONLY_TRANSPORT_MULTICAST_QOS = new Transport↩
MulticastQosPolicyKind("UNICAST_ONLY_TRANSPORT_MULTICAST_QOS", 1) [static]
```

Transport Multicast Policy Kind.

See also

com.rti.dds.infrastructure.TransportMulticastQosPolicy.UNICAST_ONLY_TRANSPORT_MULTICAST_QOS

6.138 TRANSPORT_PRIORITY

This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

Classes

- class **TransportPriorityQosPolicy**

  This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Variables

- static final **QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TransportPriorityQosPolicy** (p. 1859).*

6.138.1 Detailed Description

This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

6.138.2 Variable Documentation

6.138.2.1 TRANSPORTPRIORITY_QOS_POLICY_ID

```
final QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.TransportPriorityQosPolicy** (p. 1859).

6.139 TRANSPORT_SELECTION

<<***extension***>> *(p. 155)* Specifies the physical transports that a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) may use to send or receive data.

Classes

- class **TransportSelectionQosPolicy**

  *Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.Data↩Reader** (p. 450) may use to send or receive data.*

Variables

- static final **QosPolicyId_t TRANSPORTSELECTION_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1860)*

6.139.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies the physical transports that a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) may use to send or receive data.

6.139.2 Variable Documentation

6.139.2.1 TRANSPORTSELECTION_QOS_POLICY_ID

```
final QosPolicyId_t TRANSPORTSELECTION_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1860)

6.140 TRANSPORT_UNICAST

<<***extension***>> *(p. 155)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Classes

- class **TransportUnicastQosPolicy**

  *Specifies a subset of transports and a port number that can be used by an **Entity** (p. 1029) to receive data.*

Variables

- static final **QosPolicyId_t TRANSPORTUNICAST_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1867)*

6.140.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

6.140.2 Variable Documentation

6.140.2.1 TRANSPORTUNICAST_QOS_POLICY_ID

```
final QosPolicyId_t TRANSPORTUNICAST_QOS_POLICY_ID [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1867)

6.141 TYPE_CONSISTENCY_ENFORCEMENT

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Classes

- class **TypeConsistencyEnforcementQosPolicy**

  Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.
- class **TypeConsistencyKind**

  Kinds of type consistency.

Variables

- static final **QosPolicyId_t TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy** (p. 1935).*

6.141.1 Detailed Description

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

6.141.2 Variable Documentation

6.141.2.1 TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID

```
final QosPolicyId_t TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy** (p. 1935).

6.142 TYPESUPPORT

<<***extension***>> *(p. 155)* Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

Classes

- class **CdrPaddingKind**

  The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.
- class **TypeSupportQosPolicy**

  *Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*

Variables

- static final **QosPolicyId_t TYPESUPPORT_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** *(p. 1941)*
- static final **QosPolicyId_t ENTITYNAME_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** *(p. 1941)*

6.142.1 Detailed Description

<<***extension***>> *(p. 155)* Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

6.142.2 Variable Documentation

6.142.2.1 TYPESUPPORT_QOS_POLICY_ID

```
final  QosPolicyId_t TYPESUPPORT_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1941)

6.142.2.2 ENTITYNAME_QOS_POLICY_ID

```
final  QosPolicyId_t ENTITYNAME_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1941)

6.143 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Classes

- class **UserDataQosPolicy**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 51) during discovery.*

Variables

- static final **QosPolicyId_t USERDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1959).*

6.143.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

6.143.2 Variable Documentation

6.143.2.1 USERDATA_QOS_POLICY_ID

```
final  QosPolicyId_t USERDATA_QOS_POLICY_ID  [static]
```

Identifier for **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1959).

6.144 Exception Codes

<<***extension***>> *(p. 155)* Exception codes.

Classes

- class **BAD_PARAM**

  Exception thrown when a parameter passed to a call is considered illegal.
- class **BAD_TYPECODE**

  *The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.*
- class **BadKind**

  *The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.*
- class **BadMemberId**

  *The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member ID is invalid.*
- class **BadMemberName**

  *The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member name is invalid.*
- class **Bounds**

  A user exception thrown when a parameter is not within the legal bounds.
- class **NO_MEMORY**

  Exception thrown when there is not enough memory for a dynamic memory allocation.
- class **SystemException**

  System exception.
- class **UserException**

  User exception.

6.144.1 Detailed Description

<<***extension***>> *(*p. *155)* Exception codes.

These exceptions are used for error handling by the **Type Code Support** (p. 57) API.

6.145 WIRE_PROTOCOL

<<***extension***>> *(*p. *155)* Specifies the wire protocol related attributes for the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Classes

- class **RtpsReservedPortKind**

  RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.
- class **RtpsWellKnownPorts_t**

  RTPS well-known port mapping configuration.
- class **WireProtocolQosPolicy**

  *Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (*p. *670).*
- class **WireProtocolQosPolicyAutoKind**

  Mechanism to automatically calculate the GUID prefix.

Variables

- static final **QosPolicyId_t** **WIREPROTOCOL_QOS_POLICY_ID**

  <<***extension***>> *(*p. *155) Identifier for **com.rti.dds.infrastructure.WireProtocolQosPolicy** (*p. *1986)*
- static final int **MASK_DEFAULT** = **BUILTIN_UNICAST** | **BUILTIN_MULTICAST** | **USER_UNICAST**

  *The default value of **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask** (*p. *1992).*
- static final int **MASK_NONE**

  No bits are set.
- static final int **MASK_ALL**

  All bits are set.
- static final **RtpsWellKnownPorts_t** **RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS**

  Assign to use well-known port mappings which are compatible with previous versions of the RTI Connext middleware.
- static final **RtpsWellKnownPorts_t** **INTEROPERABLE_RTPS_WELL_KNOWN_PORTS**

  Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.
- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_IP** = new **WireProtocolQosPolicy↩ AutoKind**("RTPS_AUTO_ID_FROM_IP", 0)

  Mechanism to automatically calculate the GUID prefix.
- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_MAC** = new **WireProtocolQos↩ PolicyAutoKind**("RTPS_AUTO_ID_FROM_MAC", 1)

  Mechanism to automatically calculate the GUID prefix.
- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_UUID** = new **WireProtocolQos↩ PolicyAutoKind**("RTPS_AUTO_ID_FROM_UUID", 2)

  Mechanism to automatically calculate the GUID prefix.

6.145.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies the wire protocol related attributes for the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

6.145.2 Variable Documentation

6.145.2.1 WIREPROTOCOL_QOS_POLICY_ID

```
final  QosPolicyId_t WIREPROTOCOL_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986)

6.145.2.2 MASK_DEFAULT

```
final int MASK_DEFAULT =  BUILTIN_UNICAST | BUILTIN_MULTICAST | USER_UNICAST  [static]
```

The default value of **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask** (p. 1992).

Most of the ports that may be needed by DDS will be reserved by the transport when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622) will be detected at this time and the enable operation will fail.

This setting will avoid reserving the **usertraffic** multicast port, which is not actually used unless there are DataReaders that enable multicast but fail to specify a port.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

com.rti.dds.infrastructure.RtpsReservedPortKindMask

6.145.2.3 MASK_NONE

```
final int MASK_NONE  [static]
```

No bits are set.

None of the ports that are needed by DDS will be allocated until they are specifically required. With this value set, automatic participant Id selection will be based on selecting a port for discovery (metatraffic) unicast traffic on a single transport.

See also

com.rti.dds.infrastructure.RtpsReservedPortKindMask

6.145.2.4 MASK_ALL

```
final int MASK_ALL  [static]
```

All bits are set.

All of the ports that may be needed by DDS will be reserved when the participant is enabled. With this value set, failure to allocate a port that is computed based on the **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622) will be detected at this time, and the enable operation will fail.

Note that this will also reserve the **usertraffic** multicast port which is not actually used unless there are DataReaders that enable multicast but fail to specify a port. To avoid unnecesary resource usage for these ports, use RTPS_↩ RESERVED_PORT_MASK_DEFAULT.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also

com.rti.dds.infrastructure.RtpsReservedPortKindMask

6.145.2.5 RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS

```
final RtpsWellKnownPorts_t RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS [static]
```

Assign to use well-known port mappings which are compatible with previous versions of the RTI Connext middleware.

Assign **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports** (p. 1991) to this value to remain compatible with previous versions of the RTI Connext middleware that used fixed port mappings.

The following are the `rtps_well_known_ports` values for **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 282):

`port_base` = 7400
`domain_id_gain` = 10
`participant_id_gain` = 1000
`builtin_multicast_port_offset` = 2
`builtin_unicast_port_offset` = 0
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 3

These settings are *not* compliant with OMG's DDS Interoperability Wire Protocol. To comply with the specification, please use **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 283).

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports (p. 1991)

com.rti.dds.infrastructure.RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 283)

6.145.2.6 INTEROPERABLE_RTPS_WELL_KNOWN_PORTS

```
final RtpsWellKnownPorts_t INTEROPERABLE_RTPS_WELL_KNOWN_PORTS [static]
```

Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

Assign **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports** (p. 1991) to this value to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

The following are the `rtps_well_known_ports` values for **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 283):

`port_base` = 7400
`domain_id_gain` = 250
`participant_id_gain` = 2
`builtin_multicast_port_offset` = 0
`builtin_unicast_port_offset` = 10
`user_multicast_port_offset` = 1
`user_unicast_port_offset` = 11

Assuming a maximum port number of 65535 (UDPv4), the above settings enable the use of about 230 domains with up to 120 Participants per node per domain.

These settings are *not* backwards compatible with previous versions of the RTI Connext middleware that used fixed port mappings. For backwards compability, please use **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 282).

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports (p. 1991)

com.rti.dds.infrastructure.RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_↩ KNOWN_PORTS (p. 282)

6.145.2.7 RTPS_AUTO_ID_FROM_IP

```
final WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_IP = new WireProtocolQosPolicyAutoKind("RTPS↩
_AUTO_ID_FROM_IP", 0) [static]
```

Mechanism to automatically calculate the GUID prefix.

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind (p. 1993) .**RTPS_AUTO_ID_FROM_↩ IP** (p. 284)

6.145.2.8 RTPS_AUTO_ID_FROM_MAC

```
final WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_MAC = new WireProtocolQosPolicyAuto↩
Kind("RTPS_AUTO_ID_FROM_MAC", 1) [static]
```

Mechanism to automatically calculate the GUID prefix.

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind (p. 1993) .**RTPS_AUTO_ID_FROM_↩ MAC** (p. 284)

6.145.2.9 RTPS_AUTO_ID_FROM_UUID

```
final WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_UUID = new WireProtocolQosPolicyAuto↩
Kind("RTPS_AUTO_ID_FROM_UUID", 2) [static]
```

Mechanism to automatically calculate the GUID prefix.

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind (p. 1993) .**RTPS_AUTO_ID_FROM_↩ UUID** (p. 284)

6.146 WRITER_DATA_LIFECYCLE

Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

Classes

- class **WriterDataLifecycleQosPolicy**

  *Controls how a **com.rti.dds.publication.DataWriter** (p. 553) handles the lifecycle of the instances (keys) that it is registered to manage.*

Variables

- static final **QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006).*

6.146.1 Detailed Description

Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

6.146.2 Variable Documentation

6.146.2.1 WRITERDATALIFECYCLE_QOS_POLICY_ID

```
final QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID [static]
```

Identifier for **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006).

6.147 String Built-in Type

Built-in type consisting of a single character string.

Classes

- class **StringSeq**

  Declares IDL `sequence` < com.rti.dds.infrastructure.String > .
- class **StringDataReader**

  *<<**interface**>> (p. 156) Instantiates `DataReader` < com.rti.dds.infrastructure.String >.*
- class **StringDataWriter**

  *<<**interface**>> (p. 156) Instantiates `DataWriter` < com.rti.dds.infrastructure.String >.*
- class **StringTypeSupport**

  *<<**interface**>> (p. 156) String type support.*

6.147.1 Detailed Description

Built-in type consisting of a single character string.

6.148 KeyedString Built-in Type

Built-in type consisting of a string payload and a second string that is the key.

Classes

- class **KeyedString**

  Keyed string built-in type.
- class **KeyedStringDataReader**

  <<***interface***>> (p. *156*) *Instantiates* `DataReader` < *com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString* >.
- class **KeyedStringDataWriter**

  <<***interface***>> (p. *156*) *Instantiates* `DataWriter` < *com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString* >.
- class **KeyedStringSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.type.builtin.*↩ *com.rti.dds.type.builtin.KeyedString* > .
- class **KeyedStringTypeSupport**

  <<***interface***>> (p. *156*) *Keyed string type support.*

6.148.1 Detailed Description

Built-in type consisting of a string payload and a second string that is the key.

6.149 Octets Built-in Type

Built-in type consisting of a variable-length array of opaque bytes.

Classes

- class **Bytes**

  Built-in type consisting of a variable-length array of opaque bytes.
- class **BytesDataReader**

  <<***interface***>> (p. *156*) *Instantiates* `DataReader` < *com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes* >.
- class **BytesDataWriter**

  <<***interface***>> (p. *156*) *Instantiates* `DataWriter` < *com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes* >.
- class **BytesSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *com.rti.dds.type.builtin.*↩ *com.rti.dds.type.builtin.Bytes* > .
- class **BytesTypeSupport**

  <<***interface***>> (p. *156*) *com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type support.*

6.149.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes.

6.150 KeyedOctets Built-in Type

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

Classes

- class **KeyedBytes**

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

  *<<**interface**>> (p. 156) Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.*
- class **KeyedBytesDataWriter**

  *<<**interface**>> (p. 156) Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.*
- class **KeyedBytesSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.type.builtin.↩ com.rti.dds.type.builtin.KeyedBytes >.
- class **KeyedBytesTypeSupport**

  *<<**interface**>> (p. 156) com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type support.*

6.150.1 Detailed Description

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

6.151 Sequence Support

The com.rti.dds.infrastructure.com.rti.dds.util.Sequence interface allows you to work with variable-length collections of homogeneous data.

Modules

- **Built-in Sequences**

  Defines sequences of primitive data type. .

Classes

- class **FooSeq**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. `1066`).*
- interface **Sequence**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. `1066`).*

6.151.1 Detailed Description

The com.rti.dds.infrastructure.com.rti.dds.util.Sequence interface allows you to work with variable-length collections of homogeneous data.

This interface is a minimal extension to the standard `java.util.List` interface, making it easier to use RTI Connext alongside other Java APIs.

The `java.util.List` interface does not provide direct support for lists of primitive types. The **Built-in Sequences** (p. 92) provide extension APIs for working with collections of primitive elements without the overhead of boxing the unboxing.

When you use the rtiddsgen code generation tool, it will automatically generate concrete sequence instantiations for each of your own custom types.

See also

`http://java.sun.com/javase/6/docs/api/java/util/List.html`

6.152 Activity Context

Add contextual information to log messages.

Classes

- class **ActivityContext**

  Activity Context APIs.
- class **ActivityContextAttributeKind**

  *The resources of the **Activity Context** (p. 288) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.*

Functions

- static void **set_attribute_mask** (int attribute_mask)

  Set the com.rti.ndds.config.ActivityContextAttributeKindMask of the Activity Context.

6.152.1 Detailed Description

Add contextual information to log messages.

The Activity Context is a group of resources and activities associated with an action such as the creation of an entity.

- A resource is a abstraction of an entity. It can contain attributes such as Topic or domain ID.
- An activity is a general task that the resource is doing, such as "Getting QoS".

Logging context is one of the formats RTI Connext logging infrastructure supports. It is used by default in **com.rti.ndds.config.LogPrintFormat.NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1283). It provides information about resources and activities. The activity context is used in two places:

- Logging: activity context is one of the **com.rti.ndds.config.LogPrintFormat** (p. 1283) DDS logging infrastructure supports. If that format is set every time RTI Connext logs a message, it will contain the contextual information.
- Heap monitoring: every time memory is allocated and heap monitoring is enabled, the string representation of the activity context will be associated with the allocation. This information will be available when taking the snapshot.

For example, in the creation of a DataWriter, the activity context will provide information about:

- Resource: the Publisher creating the DataWriter. Attributes of the publisher will be GUID, kind, name and domain ID.
- Activity: entity creation. It will have two parameters, the entity kind and the Topic. In the example below, these are "Writer" and "TestTopic".

The string representation of the above activity context would be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000088{Entity=Pu,Name=TestPublisher,Domain=1}|CREATE Writer WITH TOPIC
       TestTopic]
```

Another example could be when a DataWriter writes a sample. The activity context will provide information about:

- Resource: the DataWriter writing the sample. The attributes of the DataWriter will be GUID, name, kind, Topic, data type, and the domain ID.
- Activity will be "write a sample".

The string representation of the activity context will be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]
```

6.152.2 Function Documentation

6.152.2.1 set_attribute_mask()

```
static void set_attribute_mask (
             int attribute_mask )  [static]
```

Set the com.rti.ndds.config.ActivityContextAttributeKindMask of the Activity Context.

6.153 Logging

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

Modules

- **Activity Context**

  Add contextual information to log messages.

Classes

- class **LogCategory**

  Categories of logged messages.
- class **LogFacility**

  A number that identifies the source of a log message.
- class **Logger**

  *<<**interface**>> (*p. *156) The singleton type used to configure RTI Connext logging.*
- interface **LoggerDevice**

  *<<**interface**>> (*p. *156) Logging device interface. Use for user-defined logging devices.*
- class **LogLevel**

  Level category assigned to RTI Connext log messages returned to an output device.
- class **LogMessage**

  Log message.
- class **LogPrintFormat**

  The format used to output RTI Connext diagnostic information.
- class **LogVerbosity**

  The verbosities at which RTI Connext diagnostic information is logged.
- class **SyslogLevel**

  Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.
- class **SyslogVerbosity**

  *The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **com.rti.ndds.config.LogVerbosity** (*p. *1285). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.*

6.153.1 Detailed Description

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

6.154 Version

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

Classes

- class **LibraryVersion_t**

  The version of a single library shipped as part of an RTI Connext distribution.
- class **Version**

  *<<**interface**>> (*p. *156) The version of an RTI Connext distribution.*

6.154.1 Detailed Description

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

There are three ways to obtain version information: looking at the revision files, using Visual Studio or the command line, or programmatically at run time. This HTML documentation includes a reference for consulting the APIs that allow you to get version information programmatically. For more information see the RTI Connext DDS Core Libraries User's Manual.

The version information includes four fields:

- Major product version.
- Minor product version.
- Release letter for product version.
- Revision number of product.

6.155 Heap Monitoring

Monitor memory allocations done by the middleware on the native heap.

Classes

- class **HeapMonitoring**

  Heap Monitoring APIs.
- class **HeapMonitoringParams**

  Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.
- class **HeapMonitoringSnapshotContentFormat**

  Bitmap used to decide which information of the snapshot will be displayed.
- class **HeapMonitoringSnapshotOutputFormat**

  Specify the format of the output of the snapshot. RTI Connext.

6.155.1 Detailed Description

Monitor memory allocations done by the middleware on the native heap.

RTI Connext allows you to monitor the memory allocations done by the middleware on the native heap. This feature can be used to analyze and debug unexpected memory growth.

After **com.rti.ndds.utility.HeapMonitoring.enable** (p. 1135) is called, you may invoke **com.rti.ndds.utility.Heap↩ Monitoring.take_heap_snapshot** (p. 1137) to save the current heap memory usage to a file. By comparing two snapshots, you can tell if new memory has been allocated and, in many cases, where.

6.156 Network Capture

Save network traffic into a capture file for further analysis.

Classes

- class **NetworkCapture**

  Network Capture APIs.
- class **NetworkCaptureContentKind**

  Bitmap used to specify a content type, i.e., a part of the RTPS frame.
- class **NetworkCaptureParams**

  Input parameters for starting network capture.
- class **NetworkCaptureTrafficKind**

  Bitmap used to specify whether we want to capture inbound or outbound traffic.

6.156.1 Detailed Description

Save network traffic into a capture file for further analysis.

RTI Connext allows you to capture the network traffic that one or more DomainParticipants send or receive. This feature can be used to analyze and debug communication problems between your DDS applications. When network capture is enabled, each DomainParticipant will generate a pcap-based file that can then be opened by a packet analyzer like Wireshark, provided the right dissectors are installed.

To some extent, network capture can be used as an alternative to existing pcap-based network capture software (such as Wireshark). This will be the case when you are only interested in analyzing the traffic a DomainParticipant sends/receives. In this scenario, network capture will actually have some advantages over using more general pcap-based network capture applications: RTI's network capture includes additional information such as security-related data; it also removes information that is not needed, such as user data, when you want to reduce the capture size. That said, RTI's network capture is not a replacement for other pcap-based network capture applications: it only captures the traffic exchanged by the DomainParticipants, but it does not capture any other traffic exchanged through the system network interfaces.

To capture network traffic **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324) must be invoked before creating any DomainParticipant. Similarly, **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324) must be called after deleting all participants. In between these calls, you may start, stop, pause or resume capturing traffic for one or all participants.

6.156.2 Capturing

Shared Memory Traffic

Every RTPS frame in network capture has a source and a destination associated with it. In the case of shared memory traffic, a process identifier and a port determine the source and destination endpoints.

Access to the process identifier (PID) of the source for inbound traffic requires changes in the shared memory segments. These changes would break shared memory compatibility with previous versions of RTI Connext. For this reason, by default, network capture will not populate the value of the source PID for inbound shared memory traffic.

If interoperability with previous versions of RTI Connext is not necessary, you can generate capture files containing the source PID for inbound traffic. To do so, configure the value of the *'dds.transport.minimum_compatibility_version'* property to 6.1.0. (See **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438)).

```
<domain_participant_qos>

    <property>

        <value>

            <element>

                <name>dds.transport.minimum_compatibility_version</name>

                <value>6.1.0</value>

                <propagate>false</propagate>

            </element>

        </value>

    </property>

</domain_participant_qos>
```

This property is never propagated, so it must be consistently configured throughout the whole system.

Note: Changing the value of this property affects the type of shared memory segments that RTI Connext uses. For that reason, you may see the following warning, resulting from leftover shared memory segments:

```
[0xC733A001,0xB248F671,0xAEC4A0C1:0x000001C1{Domain=200}|CREATE DP|ENABLE]
       NDDS_Transport_Shmem_is_segment_compatible:incompatible shared memory protocol detected.

Current version 4.0 not compatible with 2.0.
```

The leftover shared memory segments can be removed using the `ipcrm` command. See `https://community.rti.com/kb/what-are-possible-solutions-common-shared-memory-issues` for more information.

6.157 Observability Library

RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0 is one component of the RTI Connext Observability Framework which allows collecting and distributing telemetry data (metrics and logs) associated with the observable resources created by an RTI Connext application.

In this release, the only Observable resources are the following entities: **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807) and Application (a process running RTI Connext).

The library also accepts remote commands to change the set of distributed telemetry data at run-time.

The data distributed by RTI Monitoring Library 2.0 is sent to an RTI Observability Collector Service instance, which forwards the data to other RTI Observability Collector Service instances or stores the data in third-party observability backends such as Prometheus or Grafana Loki.

RTI Monitoring Library 2.0 is a separate library (rtimonitoring2), and applications can use it in three different modes:

- **Dynamically loaded:** This is the default mode, and it requires that the rtimonitoring2 shared library is in the library search path.
- **Dynamic linking:** The application is linked with the rtimonitoring2 shared library.
- **Static linking:** The application is linked with the rtimonitoring2 static library.

Dynamic and static linking are only supported in C and C++ applications.

To enable use of RTI Monitoring Library 2.0 and configure its behavior, use the **com.rti.dds.infrastructure.MonitoringQosPolicy** (p. 1314) QoS policy on the **com.rti.dds.domain.DomainParticipantFactory** (p. 761). This QoS policy can be configured programmatically or via XML.

6.158 Other Utilities

Other Utilities, such as com.rti.ndds.utility.Utility.spin.

Classes

- class **Utility**

  Other Utilities APIs.

6.158.1 Detailed Description

Other Utilities, such as com.rti.ndds.utility.Utility.spin.

Chapter 7

Namespace Documentation

7.1 Package com.rti

RTI.

Packages

- package **connext**

  RTI Connext Messaging communication patterns.
- package **dds**

  DDS.

7.1.1 Detailed Description

RTI.

7.2 Package com.rti.connext

RTI Connext Messaging communication patterns.

Packages

- package **requestreply**

  Support for the request-reply communication pattern.

7.2.1 Detailed Description

RTI Connext Messaging communication patterns.

7.3 Package com.rti.connext.requestreply

Support for the request-reply communication pattern.

Classes

- class **Replier**

  Allows receiving requests and sending replies.
- interface **ReplierListener**

  Called when a com.rti.connext.requestreply.Replier<TReq,TRep> has new available requests.
- class **ReplierParams**

  Contains the parameters for creating a com.rti.connext.requestreply.Replier<TReq,TRep>.
- class **Requester**

  Allows sending requests and receiving replies.
- class **RequesterParams**

  Contains the parameters for creating a com.rti.connext.requestreply.Requester<TReq,TRep>
- class **SimpleReplier**

  A callback-based replier.
- interface **SimpleReplierListener**

  *The listener called by a **SimpleReplier** (p. 1692).*
- class **SimpleReplierParams**

  Contains the parameters for creating a com.rti.connext.requestreply.SimpleReplier<TReq,TRep>

7.3.1 Detailed Description

Support for the request-reply communication pattern.

There are two basic entities that enable this pattern:

- com.rti.connext.requestreply.Requester<TReq,TRep>
- com.rti.connext.requestreply.Replier<TReq,TRep> (and a simpler version com.rti.connext.requestreply.Simple↩ Replier<TReq,TRep>)

This functionality is built on top of RTI Connext.

A **Requester** (p. 1563) publishes a request topic and subscribes to a reply topic. A **Replier** (p. 1542) subscribes to the request topic and publishes the reply topic.

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

See also

Request-Reply Examples (p. 146).

7.4 Package com.rti.dds

DDS.

Packages

- package **domain**

  *Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.*
- package **dynamicdata**

  *<<**extension**>> (p. 155) The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.*
- package **infrastructure**

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

  *Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.*
- package **subscription**
- package **topic**

  *Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.*
- package **typecode**
- package **util**

  Utility types that support the DDS API.

7.4.1 Detailed Description

DDS.

7.5 Package com.rti.dds.domain

Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.

Classes

- interface **DomainParticipant**

  *<<**interface**>> (p. 156) Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 669) objects.*
- class **DomainParticipantAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- class **DomainParticipantFactory**

  *<<**singleton**>> (p. 156) <<**interface**>> (p. 156) Allows creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 670) objects.*
- class **DomainParticipantFactoryQos**

  *QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).*
- interface **DomainParticipantListener**

  *<<**interface**>> (p. 156) Listener for participant status.*
- class **DomainParticipantProtocolStatus**

  *<<**extension**>> (p. 155) The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.*
- class **DomainParticipantQos**

  *QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 670) entity.*

7.5.1 Detailed Description

Contains the **com.rti.dds.domain.DomainParticipant** (p. 670) class that acts as an entrypoint of RTI Connext and acts as a factory for many of the classes. The **com.rti.dds.domain.DomainParticipant** (p. 670) also acts as a container for the other objects that make up RTI Connext.

7.6 Package com.rti.dds.dynamicdata

*<<**extension**>> (p. 155)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

Classes

- class **DynamicData**

  A sample of any complex data type, which can be inspected and manipulated reflectively.
- class **DynamicDataInfo**

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*
- class **DynamicDataMemberInfo**

  A descriptor for a single member (i.e. field) of dynamically defined data type.
- class **DynamicDataProperty_t**

  *A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataReader**

  *Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).*
- class **DynamicDataSeq**

  *An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 847) elements.*

- class **DynamicDataTypeProperty_t**

  *A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataTypeSerializationProperty_t**

  Properties that govern how data of a certain type will be serialized on the network.
- class **DynamicDataTypeSupport**

  *A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*
- class **DynamicDataWriter**

  *Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).*

7.6.1 Detailed Description

<<***extension***>> *(p. 155)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

This API allows you to define new data types, modify existing data types, and interact reflectively with samples. To use it, you will take the following steps:

1. Obtain a com.rti.dds.typecode.TypeCode (p. 1873) (see Type Code Support (p. 57)) that defines the type definition you want to use.
A **com.rti.dds.typecode.TypeCode** (p. 1873) includes a type's *kind* (**com.rti.dds.typecode.TCKind** (p. 1786)), *name*, and *members* (that is, fields). You can create your own **com.rti.dds.typecode.TypeCode** (p. 1873) using the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) class – see, for example, the **com.rti.dds.typecode.TypeCodeFactory.create_struct_tc** (p. 1923) method. Alternatively, you can use a remote **com.rti.dds.typecode.TypeCode** (p. 1873) that you discovered on the network (see **Built-in Topics** (p. 51)) or one generated by rtiddsgen (see the `Code Generator User's Manual`).

2. Wrap the com.rti.dds.typecode.TypeCode (p. 1873) in a com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 995) object.
See the constructor com.rti.dds.dynamicdata.DynamicDataTypeSupport.DynamicDataTypeSupport.DynamicDataTypeSupport. This object lets you connect the type definition to a **com.rti.dds.domain.DomainParticipant** (p. 670) and manage data samples (of type **com.rti.dds.dynamicdata.DynamicData** (p. 847)).

3. Register the com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 995) with one or more domain participants.
See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 997). This action associates the data type with a logical name that you can use to create topics. (Starting with this step, working with a dynamically defined data type is almost exactly the same as working with a generated one.)

4. Create a com.rti.dds.topic.Topic (p. 1807) from the com.rti.dds.domain.DomainParticipant (p. 670).
Use the name under which you registered your data type – see **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706). This **com.rti.dds.topic.Topic** (p. 1807) is what you will use to produce and consume data.

5. Create a com.rti.dds.dynamicdata.DynamicDataWriter (p. 1003) and/or com.rti.dds.dynamicdata.DynamicDataReader (p. 959).
These objects will produce and/or consume data (of type **com.rti.dds.dynamicdata.DynamicData** (p. 847)) on the **com.rti.dds.topic.Topic** (p. 1807). You can create these objects directly from the **com.rti.dds.domain.DomainParticipant** (p. 670) – see **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 700) and **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 703) – or by first creating intermediate **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730) objects – see **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693) and **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697).

6. Write and/or read the data of interest.

7. Tear down the objects described above.
You should delete them in the reverse order in which you created them. Note that unregistering your data type with the **com.rti.dds.domain.DomainParticipant** (p. 670) is optional; all types are automatically unregistered when the **com.rti.dds.domain.DomainParticipant** (p. 670) itself is deleted.

7.7 Package com.rti.dds.infrastructure

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

Classes

- class **AckResponseData_t**

  Data payload of an application-level acknowledgment.
- class **AllocationSettings_t**

  Resource allocation settings.
- class **AsynchronousPublisherQosPolicy**

  Configures the mechanism that sends user data in an external middleware thread.
- class **AvailabilityQosPolicy**

  Configures the availability of data.
- class **BAD_PARAM**

  Exception thrown when a parameter passed to a call is considered illegal.
- class **BAD_TYPECODE**

  *The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.*
- class **BadKind**

  *The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.*
- class **BadMemberId**

  *The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member ID is invalid.*
- class **BadMemberName**

  *The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member name is invalid.*
- class **BatchQosPolicy**

  Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.
- class **BooleanSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *boolean* >
- class **Bounds**

  A user exception thrown when a parameter is not within the legal bounds.
- class **BuiltinQosProfiles**

  The available built-in QoS libraries, profiles, and snippets.
- class **BuiltinTopicReaderResourceLimits_t**

  Built-in topic reader's resource limits.
- class **ByteSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *byte* >

- class **CdrPaddingKind**

  The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.
- class **ChannelSettings_t**

  Type used to configure the properties of a channel.
- class **ChannelSettingsSeq**

  Declares IDL `sequence<` ***com.rti.dds.infrastructure.ChannelSettings_t*** *(p. 411) >*
- class **CharSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *< char >*
- class **CoherentSetInfo_t**

  *<<**extension**>> (p. 155) Type definition for a coherent set info.*
- class **CompressionSettings_t**

  *<<**extension**>> (p. 155) Settings related to compressing user data.*
- interface **Condition**

  *<<**interface**>> (p. 156) Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1973).*
- class **ConditionSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *< **com.rti.dds.infrastructure.Condition** (p. 429) >*
- class **ContentFilterProperty_t**

  *<<**extension**>> (p. 155) Type used to provide all the required information to enable content filtering.*
- class **Cookie_t**

  *<<**extension**>> (p. 155) Sequence of bytes.*
- class **CookieSeq**

  Declares IDL `sequence` *< **com.rti.dds.infrastructure.Cookie_t** (p. 442) > .*
- interface **Copyable**

  *<<**extension**>> (p. 155) <<**interface**>> (p. 156) Interface for all the user-defined data type classes that support copy.*
- class **DatabaseQosPolicy**

  Various threads and resource limits settings used by RTI Connext to control its internal database.
- class **DataReaderInstanceRemovalKind**

  *Sets the kinds of instances that can be replaced when instance resource limits (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)) are reached.*
- class **DataReaderProtocolQosPolicy**

  *Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596), this QoS policy configures the DDS on-the-network protocol (RTPS).*
- class **DataReaderResourceLimitsInstanceReplacementSettings**

  Instance replacement kind applied to each instance state.
- class **DataReaderResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 450) allocates and uses physical memory for internal resources.*
- class **DataRepresentationQosPolicy**

  *This QoS policy contains a list of representation identifiers and compression settings used by **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities to negotiate which data representation and compression settings to use.*
- class **DataTagQosPolicy**

  Stores (name, value) pairs that can be used to determine access permissions.
- class **DataTagQosPolicyHelper**

  Policy helpers that facilitate management of the data tags in the input policy.
- class **DataWriterProtocolQosPolicy**

*Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 553) instances.*

- class **DataWriterResourceLimitsInstanceReplacementKind**

  Sets the kinds of instances that can be replaced when instance resource limits are reached.
- class **DataWriterResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.*
- class **DeadlineQosPolicy**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.
- class **DestinationOrderQosPolicy**

  *Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1807) and key).*
- class **DestinationOrderQosPolicyKind**

  Kinds of destination order.
- class **DestinationOrderQosPolicyScopeKind**

  Scope of source destination order.
- class **DiscoveryBuiltinReaderFragmentationResourceLimits_t**
- class **DiscoveryConfigBuiltinChannelKind**

  Built-in channels that can be enabled.
- class **DiscoveryConfigBuiltinPluginKind**

  Built-in discovery plugins that can be used.
- class **DiscoveryConfigQosPolicy**

  Settings for discovery configuration.
- class **DiscoveryQosPolicy**

  *<<**extension**>> (p. 155) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.*
- interface **DomainEntity**

  *<<**interface**>> (p. 156) Abstract base class for all DDS entities except for the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- class **DomainParticipantConfigParams_t**

  *<<**extension**>> (p. 155) Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.*
- class **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**

  Available replacement policies for the ignored entities.
- class **DomainParticipantResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*
- class **DoubleSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < *double* >
- class **DurabilityQosPolicy**

  *This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.*
- class **DurabilityQosPolicyKind**

  Kinds of durability.
- class **DurabilityServiceQosPolicy**

  *Various settings to configure the external RTI Persistence Service used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicyKind.↩ DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.*
- class **Duration_t**

Type for duration representation.

- class **EndpointGroup_t**

  Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

- class **EndpointGroupSeq**

  *A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022).*

- class **EndpointTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered endpoint.

- class **EndpointTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered endpoint.

- class **EndpointTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered endpoint.

- interface **Entity**

  *<<**interface**>> (p. 156) Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.*

- class **EntityFactoryQosPolicy**

  *A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.*

- class **EntityNameQosPolicy**

  *Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

- class **EventQosPolicy**

  Settings for event.

- class **FloatSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < float >

- class **GroupDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*

- class **GuardCondition**

  *<<**interface**>> (p. 156) A specific **com.rti.dds.infrastructure.Condition** (p. 429) whose `trigger_value` is completely under the control of the application.*

- class **GUID_t**

  Type for GUID (Global Unique Identifier) representation.

- class **HistoryQosPolicy**

  Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

- class **HistoryQosPolicyKind**

  Kinds of history.

- class **IMMUTABLE_TYPECODE**

  *An attempt was made to modify a **com.rti.dds.typecode.TypeCode** (p. 1873) that was received from a remote object.*

- class **InetAddressSeq**

  Declares IDL `sequence`< com.rti.dds.infrastructure.java.net.InetAddress >

- class **InstanceHandle_t**

  Type definition for an instance handle.

- class **InstanceHandleSeq**

  *Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) > .*

- class **InstanceStateConsistencyKind**

*<<**extension**>> (p. 155) Whether instance state consistency is enabled.*

- class **IntSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *< com.rti.dds.infrastructure.↩ int >*

- class **LatencyBudgetQosPolicy**

  Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

- class **LifespanQosPolicy**

  *Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.*

- interface **Listener**

  *<<**interface**>> (p. 156) Abstract base class for all **Listener** (p. 1236) interfaces.*

- class **LivelinessQosPolicy**

  *Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".*

- class **LivelinessQosPolicyKind**

  Kinds of liveliness.

- class **Locator_t**

  *<<**extension**>> (p. 155) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*

- class **LocatorFilter_t**

  Specifies the configuration of an individual channel within a MultiChannel DataWriter.

- class **LocatorFilterQosPolicy**

  *The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.↩ builtin.PublicationBuiltinTopicData** (p. 1452).*

- class **LocatorFilterSeq**

  Declares IDL `sequence<` ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1258) >.*

- class **LocatorSeq**

  Declares IDL `sequence` *< **com.rti.dds.infrastructure.Locator_t** (p. 1253) >*

- class **LoggingQosPolicy**

  Configures the RTI Connext logging facility.

- class **LongDoubleSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *< com.rti.dds.infrastructure.↩ LongDouble >*

- class **LongSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *< long >*

- class **MonitoringDedicatedParticipantSettings**

  *Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.*

- class **MonitoringDistributionSettings**

  Configures the distribution of telemetry data.

- class **MonitoringEventDistributionSettings**

  Configures the distribution of event metrics.

- class **MonitoringLoggingDistributionSettings**

  Configures the distribution of log messages.

- class **MonitoringLoggingForwardingSettings**

  *Configures the forwarding levels of log messages for the different **com.rti.ndds.config.LogFacility** (p. 1265).*

- class **MonitoringMetricSelection**

  This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

- class **MonitoringMetricSelectionSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.MonitoringMetricSelection*** *(p. 1308) >*
- class **MonitoringPeriodicDistributionSettings**

  Configures the distribution of periodic metrics.
- class **MonitoringQosPolicy**

  Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.
- class **MonitoringTelemetryData**

  Configures the telemetry data that will be distributed.
- class **MultiChannelQosPolicy**

  Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.
- class **NO_MEMORY**

  Exception thrown when there is not enough memory for a dynamic memory allocation.
- class **ObjectHolder**

  *<<****extension****>> (p. 155) Holder of object instance*
- class **OwnershipQosPolicy**

  Specifies whether it is allowed for multiple ***com.rti.dds.publication.DataWriter*** *(p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*
- class **OwnershipQosPolicyKind**

  Kinds of ownership.
- class **OwnershipStrengthQosPolicy**

  Specifies the value of the strength used to arbitrate among multiple ***com.rti.dds.publication.DataWriter*** *(p. 553) objects that attempt to modify the same instance of a data type (identified by* ***com.rti.dds.topic.Topic*** *(p. 1807) + key).*
- class **ParticipantTrustAlgorithmInfo**

  Trust Plugins algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustInterceptorAlgorithmInfo**

  Trust Plugins interception algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustKeyEstablishmentAlgorithmInfo**

  Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.
- class **ParticipantTrustProtectionInfo**

  Trust Plugins Protection information associated with the discovered DomainParticipant.
- class **ParticipantTrustSignatureAlgorithmInfo**

  Trust Plugins signature algorithm information associated with the discovered DomainParticipant.
- class **PartitionQosPolicy**

  Set of strings that introduces logical partitions in ***com.rti.dds.domain.DomainParticipant*** *(p. 670),* ***com.rti.dds.publication.Publisher*** *(p. 1466), or* ***com.rti.dds.subscription.Subscriber*** *(p. 1730) entities.*
- class **PersistentJournalKind**

  Sets the journal mode of the persistent storage.
- class **PersistentStorageSettings**

  Configures durable writer history and durable reader state.
- class **PersistentSynchronizationKind**

  *<<****extension****>> (p. 155) Whether instance state consistency is enabled.*
- class **PresentationQosPolicy**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.
- class **PresentationQosPolicyAccessScopeKind**

  Kinds of presentation "access scope".
- class **ProductVersion_t**

  *<<****extension****>> (p. 155) Type used to represent the current version of RTI Connext.*

- class **ProfileQosPolicy**

  Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.
- class **Property_t**

  Properties are name/value pairs objects.
- class **PropertyQosPolicy**

  Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.
- class **PropertyQosPolicyHelper**

  Policy helpers that facilitate management of the properties in the input policy.
- class **PropertyQosPolicyMutability**

  Determines if, and when, the value of a property can change.
- class **PropertySeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.Property_t*** *(*p. *1395) >*
- class **ProtocolVersion_t**

  *<<****extension****>> (*p. *155) Type used to represent the version of the RTPS protocol.*
- interface **PublicationPriority**
- class **PublishModeQosPolicy**

  Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its own thread to send data, instead of the user thread.
- class **PublishModeQosPolicyKind**

  Kinds of publishing mode.
- class **Qos**

  An abstract base class for all QoS types.
- class **QosPolicy**

  The base class for all QoS policies.
- class **QosPolicyCount**

  Type to hold a counter for a ***com.rti.dds.infrastructure.QosPolicyId_t*** *(*p. *1504).*
- class **QosPolicyCountSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.QosPolicyCount*** *(*p. *1502) >*
- class **QosPolicyId_t**

  Type to identify `QosPolicies`*.*
- class **QosPrintFormat**

  A collection of attributes used to configure how a QoS appears when printed.
- class **ReaderDataLifecycleQosPolicy**

  Controls how a DataReader manages the lifecycle of the data that it has received.
- class **ReceiverPoolQosPolicy**

  Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).
- class **ReliabilityQosPolicy**

  Indicates the level of reliability offered/requested by RTI Connext.
- class **ReliabilityQosPolicyAcknowledgmentModeKind**

  *<<****extension****>> (*p. *155) Kinds of acknowledgment.*
- class **ReliabilityQosPolicyKind**

  Kinds of reliability.
- class **RemoteParticipantPurgeKind**

  Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.
- class **ResourceLimitsQosPolicy**

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

- class **RETCODE_ALREADY_DELETED**

  The object target of this operation has already been deleted.
- class **RETCODE_BAD_PARAMETER**

  Illegal parameter value.
- class **RETCODE_ERROR**

  Generic, unspecified error.
- class **RETCODE_ILLEGAL_OPERATION**

  The operation was called under improper circumstances.
- class **RETCODE_IMMUTABLE_POLICY**

  Application attempted to modify an immutable QoS policy.
- class **RETCODE_INCONSISTENT_POLICY**

  Application specified a set of QoS policies that are not consistent with each other.
- class **RETCODE_NO_DATA**

  Indicates a transient situation where the operation did not return any data but there is no inherent error.
- class **RETCODE_NOT_ALLOWED_BY_SECURITY**

  An operation on the DDS API that fails because the security plugins do not allow it.
- class **RETCODE_NOT_ENABLED**

  *Operation invoked on a **com.rti.dds.infrastructure.Entity** (p. 1029) that is not yet enabled.*
- class **RETCODE_OUT_OF_RESOURCES**

  RTI Connext ran out of the resources needed to complete the operation.
- class **RETCODE_PRECONDITION_NOT_MET**

  A pre-condition for the operation was not met.
- class **RETCODE_TIMEOUT**

  The operation timed out.
- class **RETCODE_UNSUPPORTED**

  Unsupported operation. Only returned by operations that are unsupported.
- class **RtpsReliableReaderProtocol_t**

  ***Qos** (p. 1500) related to reliable reader protocol defined in RTPS.*
- class **RtpsReliableWriterProtocol_t**

  QoS related to the reliable writer protocol defined in RTPS.
- class **RtpsReservedPortKind**

  RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.
- class **RtpsWellKnownPorts_t**

  RTPS well-known port mapping configuration.
- class **SampleFlagBits**

  Type to identify the sample flags reserved by RTI.
- class **SampleIdentity_t**

  Type definition for a Sample Identity.
- class **SequenceNumber_t**

  Type for sequence number representation.
- class **ServiceQosPolicy**

  Service associated with a DDS entity.
- class **ServiceQosPolicyKind**

  Kinds of service.
- class **ShortSeq**

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < ***short*** >

- interface **StatusCondition**

  <<***interface***>> (p. *156*) *A specific* ***com.rti.dds.infrastructure.Condition*** (p. *429*) *that is associated with each* ***com.rti.dds.infrastructure.Entity*** (p. *1029*).

- class **StatusKind**

  Type for status kinds.

- class **StringSeq**

  Declares IDL `sequence` < *com.rti.dds.infrastructure.String* > .

- class **SystemException**

  System exception.

- class **SystemResourceLimitsQosPolicy**

  <<***extension***>> (p. *155*) *Configures* ***com.rti.dds.domain.DomainParticipant*** (p. *670*)*-independent resources used by RTI Connext. Mainly used to change the maximum number of* ***com.rti.dds.domain.DomainParticipant*** (p. *670*) *entities that can be created within a single process (address space).*

- class **Tag**

  Tags are name/value pair objects.

- class **TagSeq**

  Declares IDL `sequence` < ***com.rti.dds.infrastructure.Tag*** (p. *1783*) >

- class **ThreadSettings_t**

  The properties of a thread of execution.

- class **ThreadSettingsCpuRotationKind**

  Determines how ***com.rti.dds.infrastructure.ThreadSettings_t.cpu_list*** (p. *1794*) *affects processor affinity for thread-related QoS policies that apply to multiple threads.*

- class **ThreadSettingsKind**

  A collection of flags used to configure threads of execution.

- class **Time_t**

  Type for time representation.

- class **TimeBasedFilterQosPolicy**

  Filter that allows a ***com.rti.dds.subscription.DataReader*** (p. *450*) *to specify that it is interested only in (potentially) a subset of the values of the data.*

- class **TopicDataQosPolicy**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** (p. *51*) *during discovery.*

- class **TopicQueryDispatchQosPolicy**

  Configures the ability of a ***com.rti.dds.publication.DataWriter*** (p. *553*) *to publish samples in response to a* ***com.rti.dds.subscription.TopicQuery*** (p. *1830*).

- class **TransportBuiltinKind**

  Built-in transport kind.

- class **TransportBuiltinQosPolicy**

  Specifies which built-in transports are used.

- class **TransportInfo_t**

  Contains the class_id and message_size_max of an installed transport.

- class **TransportInfoSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < ***com.rti.dds.infrastructure.TransportInfo_t*** (p. *1845*) > .

- class **TransportMulticastMapping_t**

  Type representing a list of multicast mapping elements.

- class **TransportMulticastMappingFunction_t**

  Type representing an external mapping function.

- class **TransportMulticastMappingQosPolicy**

  *Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.*
- class **TransportMulticastMappingSeq**

  *Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) >*
- class **TransportMulticastQosPolicy**

  *Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.*
- class **TransportMulticastQosPolicyKind**

  Transport Multicast Policy Kind.
- class **TransportMulticastSettings_t**

  Type representing a list of multicast locators.
- class **TransportMulticastSettingsSeq**

  *Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) >*
- class **TransportPriorityQosPolicy**

  This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.
- class **TransportSelectionQosPolicy**

  *Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) may use to send or receive data.*
- class **TransportUdpWanCommPortsMappingInfo_t**

  Type for storing UDP WAN communication ports.
- class **TransportUdpWanCommPortsMappingInfoSeq**

  *List of < **com.rti.dds.infrastructure.TransportUdpWanCommPortsMappingInfo_t** (p. 1864) > .*
- class **TransportUnicastQosPolicy**

  *Specifies a subset of transports and a port number that can be used by an **Entity** (p. 1029) to receive data.*
- class **TransportUnicastSettings_t**

  Type representing a list of unicast locators.
- class **TransportUnicastSettingsSeq**

  *Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportUnicastSettings_t** (p. 1868) >*
- class **TrustAlgorithmRequirements**

  Type to describe Trust Plugins algorithm requirements for an entity.
- class **TypeConsistencyEnforcementQosPolicy**

  Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.
- class **TypeConsistencyKind**

  Kinds of type consistency.
- class **TypeSupportQosPolicy**

  *Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.*
- class **UserDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.*
- class **UserException**

  User exception.
- class **VendorId_t**

  *<<**extension**>> (p. 155) Type used to represent the vendor of the service implementing the RTPS protocol.*

- class **WaitSet**

  *<<**interface**>> (p. 156) Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.Condition** (p. 429) objects has a `trigger_value` of com.rti.dds.infrastructure.true or else until the timeout expires.*
- class **WaitSetProperty_t**

  *<<**extension**>> (p. 155) Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1973) behavior for multiple trigger events.*
- class **WcharSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.char >
- class **WireProtocolQosPolicy**

  *Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- class **WireProtocolQosPolicyAutoKind**

  Mechanism to automatically calculate the GUID prefix.
- class **WriteParams_t**

  *<<**extension**>> (p. 155) Input parameters for writing with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), **com.rti.ndds.example.FooDataWriter.dispose_w_params** (p. 1114), **com.rti.ndds.example.FooDataWriter.register_instance_w_params** (p. 1100), **com.rti.ndds.example.FooDataWriter.unregister_instance_w_params** (p. 1104)*
- class **WriterDataLifecycleQosPolicy**

  *Controls how a **com.rti.dds.publication.DataWriter** (p. 553) handles the lifecycle of the instances (keys) that it is registered to manage.*
- class **WstringSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.char >*

7.7.1 Detailed Description

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

"DCPS Infrastructure package"

7.8 Package com.rti.dds.publication

Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.

Classes

- class **AcknowledgmentInfo**

  Information about an application-level acknowledged sample.
- interface **DataWriter**

  *<<**interface**>> (p. 156) Allows an application to set the value of the data to be published under a given **com.rti.dds.topic.Topic** (p. 1807).*
- class **DataWriterAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods or functions.)*
- class **DataWriterCacheStatus**

  *<<**extension**>> (p. 155) The status of the **DataWriter** (p. 553)'s cache. Provides information on cache related metrics such as the number of samples and instances in the **DataWriter** (p. 553) queue.*
- interface **DataWriterListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for writer status.*
- class **DataWriterProtocolStatus**

  *<<**extension**>> (p. 155) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.*
- class **DataWriterQos**

  *QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 553) entity.*
- interface **FlowController**

  *<<**interface**>> (p. 156) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances are allowed to write data.*
- class **FlowControllerProperty_t**

  *Determines the flow control characteristics of the **com.rti.dds.publication.FlowController** (p. 1055).*
- class **FlowControllerSchedulingPolicy**

  Kinds of flow controller scheduling policy.
- class **FlowControllerTokenBucketProperty_t**

  ***com.rti.dds.publication.FlowController** (p. 1055) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.*
- class **LivelinessLostStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS
- class **OfferedDeadlineMissedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS
- class **OfferedIncompatibleQosStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS
- class **PublicationMatchedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS
- interface **Publisher**

  *<<**interface**>> (p. 156) A publisher is the object responsible for the actual dissemination of publications.*
- class **PublisherAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- interface **PublisherListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.publication.Publisher** (p. 1466) status.*
- class **PublisherQos**

  *QoS policies supported by a **com.rti.dds.publication.Publisher** (p. 1466) entity.*
- class **PublisherSeq**

Declares IDL `sequence` *<* ***com.rti.dds.publication.Publisher*** *(p. 1466) >.*

- class **ReliableReaderActivityChangedStatus**

  *<<**extension**>> (p. 155) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*
- class **ReliableWriterCacheChangedStatus**

  *<<**extension**>> (p. 155) A summary of the state of a data writer's cache of unacknowledged samples written.*
- class **ReliableWriterCacheEventCount**

  *<<**extension**>> (p. 155) The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.*
- class **ServiceRequestAcceptedStatus**

  com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS *(p. 1709)*

7.8.1 Detailed Description

Contains the **com.rti.dds.publication.FlowController** (p. 1055), **com.rti.dds.publication.Publisher** (p. 1466), and **com.rti.dds.publication.DataWriter** (p. 553) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1488) and **com.rti.dds.publication.DataWriterListener** (p. 589) interfaces, and more generally, all that is needed on the publication side.

"DCPS Publication package"

7.9 Package com.rti.dds.subscription

Classes

- interface **DataReader**

  *<<**interface**>> (p. 156) Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1730).*
- class **DataReaderAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- class **DataReaderCacheStatus**

  *<<**extension**>> (p. 155) The status of the reader's cache.*
- interface **DataReaderListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for reader status.*
- class **DataReaderProtocolStatus**

  *<<**extension**>> (p. 155) The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.*
- class **DataReaderQos**

  *QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 450) entity.*
- class **DataReaderSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.subscription.DataReader*** *(p. 450) >.*
- class **InstanceStateKind**

  *Indicates if the samples are from a live **com.rti.dds.publication.DataWriter** (p. 553) or not.*
- class **LivelinessChangedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS

- interface **QueryCondition**

  *<<**interface**>> (p. 156) These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1514) objects that allow the application to also specify a filter on the locally available data.*
- class **QueryConditionParams**

  *<<**extension**>> (p. 155) Input parameters for **com.rti.dds.subscription.DataReader.create_querycondition_w_↩params** (p. 456)*
- interface **ReadCondition**

  *<<**interface**>> (p. 156) Conditions specifically dedicated to read operations and attached to one **com.rti.dds.↩subscription.DataReader** (p. 450).*
- class **ReadConditionParams**

  *<<**extension**>> (p. 155) Input parameters for **com.rti.dds.subscription.DataReader.create_readcondition_w_↩params** (p. 455)*
- class **RequestedDeadlineMissedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS
- class **RequestedIncompatibleQosStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS
- class **SampleInfo**

  Information that accompanies each sample that is `read` or `taken`.
- class **SampleInfoSeq**

  *Declares IDL `sequence` < **com.rti.dds.subscription.SampleInfo** (p. 1634) > .*
- class **SampleLostStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS
- class **SampleLostStatusKind**

  *<<**extension**>> (p. 155) Kinds of reasons why a sample was lost.*
- class **SampleRejectedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS
- class **SampleRejectedStatusKind**

  Kinds of reasons for rejecting a sample.
- class **SampleStateKind**

  Indicates whether or not a sample has ever been read.
- class **StreamKind**

  *Indicates if the samples are **TopicQuery** (p. 1830) samples or not.*
- interface **Subscriber**

  *<<**interface**>> (p. 156) A subscriber is the object responsible for actually receiving data from a subscription.*
- class **SubscriberAdapter**

  A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)
- interface **SubscriberListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for status about a subscriber.*
- class **SubscriberQos**

  *QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1730) entity.*
- class **SubscriberSeq**

  *Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1730) > .*
- class **SubscriptionMatchedStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS
- interface **TopicQuery**

  *<<**extension**>> (p. 155) Allows a **com.rti.dds.subscription.DataReader** (p. 450) to query the sample cache of its matching **com.rti.dds.publication.DataWriter** (p. 553).*

- class **TopicQueryData**

  *<<**extension**>> (p. 155) Provides information about a **com.rti.dds.subscription.TopicQuery** (p. 1830)*
- class **TopicQueryHelper**

  *Helpers to provide utility operations related to **com.rti.dds.subscription.TopicQuery** (p. 1830).*
- class **TopicQuerySelection**

  *<<**extension**>> (p. 155) Specifies the data query that defines a **com.rti.dds.subscription.TopicQuery** (p. 1830)*
- class **TopicQuerySelectionKind**

  *Kinds of **TopicQuerySelection** (p. 1836).*
- class **ViewStateKind**

  Indicates whether or not an instance is new.

7.9.1 Detailed Description

See **Subscription Module** (p. 78)

7.10 Package com.rti.dds.topic

Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.

Classes

- class **BuiltinTopicKey_t**

  The key type of the built-in topic types.
- interface **ContentFilter**

  *<<**interface**>> (p. 156) Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436)*
- interface **ContentFilteredTopic**

  *<<**interface**>> (p. 156) Specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows for content-based subscriptions.*
- class **ExpressionProperty**

  *Provides additional information about the filter expression passed to **com.rti.dds.topic.WriterContentFilter.writer_compile** (p. 2003) .*
- class **FilterSampleInfo**

  Provides meta information associated with the sample.
- class **InconsistentTopicStatus**

  com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS
- interface **MultiTopic**

  ***[Not supported (optional)]** <<**interface**>> (p. 156) A specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows subscriptions that combine/filter/rearrange data coming from several topics.*
- class **PrintFormatKind**

  Format kinds available when converting data samples to string representations.
- class **PrintFormatProperty**

  A collection of attributes used to configure how data samples will be formatted when converted to a string.

- interface **Topic**

  *<<**interface**>> (p. 156) The most basic description of the data to be published and subscribed.*
- class **TopicAdapter**

  *<<**extension**>> (p. 155) A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*
- interface **TopicDescription**

  ***com.rti.dds.topic.Topic** (p. 1807) entity and associated elements*
- interface **TopicListener**

  *<<**interface**>> (p. 156) **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.topic.Topic** (p. 1807) entities.*
- class **TopicQos**

  *QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1807) entity.*
- interface **TypeSupport**

  *<<**interface**>> (p. 156) An abstract marker interface that has to be specialized for each concrete user data type that will be used by the application.*
- interface **WriterContentFilter**

  *<<**interface**>> (p. 156) Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).*

7.10.1 Detailed Description

Contains the **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) classes, the **com.rti.dds.topic.TopicListener** (p. 1822) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1807) objects and attach QoS policies to them.

7.11 Package com.rti.dds.type.builtin

Classes

- class **Bytes**

  Built-in type consisting of a variable-length array of opaque bytes.
- class **BytesDataReader**

  *<<**interface**>> (p. 156) Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.*
- class **BytesDataWriter**

  *<<**interface**>> (p. 156) Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.*
- class **BytesSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes > .
- class **BytesTypeSupport**

  *<<**interface**>> (p. 156) com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type support.*
- class **KeyedBytes**

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

  *<<**interface**>> (p. 156) Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.*
- class **KeyedBytesDataWriter**

  *<<**interface**>> (p. 156) Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.*
- class **KeyedBytesSeq**

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *<* *com.rti.dds.type.builtin.↩ com.rti.dds.type.builtin.KeyedBytes >.*

- class **KeyedBytesTypeSupport**

  *<<**interface**>>* (p. *156*) *com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type support.*
- class **KeyedString**

  Keyed string built-in type.
- class **KeyedStringDataReader**

  *<<**interface**>>* (p. *156*) *Instantiates* `DataReader` *< com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.*
- class **KeyedStringDataWriter**

  *<<**interface**>>* (p. *156*) *Instantiates* `DataWriter` *< com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.*
- class **KeyedStringSeq**

  Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` *<* *com.rti.dds.type.builtin.↩ com.rti.dds.type.builtin.KeyedString > .*
- class **KeyedStringTypeSupport**

  *<<**interface**>>* (p. *156*) *Keyed string type support.*
- class **StringDataReader**

  *<<**interface**>>* (p. *156*) *Instantiates* `DataReader` *< com.rti.dds.infrastructure.String >.*
- class **StringDataWriter**

  *<<**interface**>>* (p. *156*) *Instantiates* `DataWriter` *< com.rti.dds.infrastructure.String >.*
- class **StringTypeSupport**

  *<<**interface**>>* (p. *156*) *String type support.*

7.11.1 Detailed Description

See **Built-in Types** (p. 61)

7.12 Package com.rti.dds.typecode

Classes

- class **EnumMember**

  A description of a member of an enumeration.
- class **ExtensibilityKind**

  Type to indicate the extensibility of a type.
- class **PRIVATE_MEMBER**

  Constant used to indicate that a value type member is private.
- class **PUBLIC_MEMBER**

  Constant used to indicate that a value type member is public.
- class **StructMember**

  A description of a member of a struct.
- class **TCKind**

  Enumeration type for ***com.rti.dds.typecode.TypeCode*** (p. *1873*) *kinds.*
- class **TypeCode**

The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.

- class **TypeCodeFactory**

  A singleton factory for creating, copying, and deleting data type definitions dynamically.
- class **UnionMember**

  A description of a member of a union.
- class **ValueMember**

  A description of a member of a value type.
- class **VM_ABSTRACT**

  Constant used to indicate that a value type has the `abstract` modifier.
- class **VM_CUSTOM**

  Constant used to indicate that a value type has the `custom` modifier.
- class **VM_NONE**

  Constant used to indicate that a value type has no modifiers.
- class **VM_TRUNCATABLE**

  Constant used to indicate that a value type has the `truncatable` modifier.

7.12.1 Detailed Description

See **Type Code Support** (p. 57)

7.13 Package com.rti.dds.util

Utility types that support the DDS API.

Classes

- class **AbstractPrimitiveSequence**
- class **AbstractSequence**

  Abstract sequence.
- class **Base64**
- class **Enum**

  A superclass for all type-safe enumerated types.
- class **LoanableSequence**

  A sequence capable of storing its elements directly or taking out a loan on them from an internal middleware store.
- interface **Sequence**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. `1066`).*
- class **Union**

7.13.1 Detailed Description

Utility types that support the DDS API.

7.14 Package com.rti.ndds.config

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

Classes

- class **ActivityContext**

  Activity Context APIs.
- class **ActivityContextAttributeKind**

  *The resources of the **Activity Context** (p. 288) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.*
- class **LibraryVersion_t**

  The version of a single library shipped as part of an RTI Connext distribution.
- class **LogCategory**

  Categories of logged messages.
- class **LogFacility**

  A number that identifies the source of a log message.
- class **Logger**

  *<<**interface**>> (p. 156) The singleton type used to configure RTI Connext logging.*
- interface **LoggerDevice**

  *<<**interface**>> (p. 156) Logging device interface. Use for user-defined logging devices.*
- class **LogLevel**

  Level category assigned to RTI Connext log messages returned to an output device.
- class **LogMessage**

  Log message.
- class **LogPrintFormat**

  The format used to output RTI Connext diagnostic information.
- class **LogVerbosity**

  The verbosities at which RTI Connext diagnostic information is logged.
- class **SyslogLevel**

  Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.
- class **SyslogVerbosity**

  *The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **com.rti.ndds.config.LogVerbosity** (p. 1285). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.*
- class **Version**

  *<<**interface**>> (p. 156) The version of an RTI Connext distribution.*

7.14.1 Detailed Description

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

7.15 Package com.rti.ndds.example

Programming HowTos: Code templates for common use cases.

Classes

- class **Foo**

  A representative user-defined data type.
- class **FooDataReader**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type-specific data reader.*
- class **FooDataWriter**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type specific data writer.*
- class **FooSeq**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. 1066).*
- class **FooTypeSupport**

  *<<**interface**>> (p. 156) <<**generic**>> (p. 156) User data type specific interface.*

7.15.1 Detailed Description

Programming HowTos: Code templates for common use cases.

7.16 com::rti::ndds::transport Namespace Reference

Classes

- interface **ShmemTransport**

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

  RTI Connext's abstract pluggable transport interface.
- class **TransportSupport**

  *<<**interface**>> (p. 156) The utility class used to configure RTI Connext pluggable transports.*
- interface **UDPv4Transport**

  ***Transport** (p. 1841) plug-in using UDP/IPv4.*
- interface **UDPv4WanTransport**

  ***Transport** (p. 1841) plug-in using UDP/IPv4 for WAN communications..*
- interface **UDPv6Transport**

  ***Transport** (p. 1841) plug-in using UDP/IPv6.*

7.16.1 Detailed Description

See **APIs related to RTI Connext pluggable transports** (p. 95).

7.17 Package com.rti.ndds.utility

API of general utilities used in the RTI Connext distribution.

Classes

- class **HeapMonitoring**

  Heap Monitoring APIs.
- class **HeapMonitoringParams**

  Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.
- class **HeapMonitoringSnapshotContentFormat**

  Bitmap used to decide which information of the snapshot will be displayed.
- class **HeapMonitoringSnapshotOutputFormat**

  Specify the format of the output of the snapshot. RTI Connext.
- class **NetworkCapture**

  Network Capture APIs.
- class **NetworkCaptureContentKind**

  Bitmap used to specify a content type, i.e., a part of the RTPS frame.
- class **NetworkCaptureParams**

  Input parameters for starting network capture.
- class **NetworkCaptureTrafficKind**

  Bitmap used to specify whether we want to capture inbound or outbound traffic.

7.17.1 Detailed Description

API of general utilities used in the RTI Connext distribution.

Chapter 8

Class Documentation

8.1 AbstractBuiltinTopicDataTypeSupport Class Reference

Inheritance diagram for AbstractBuiltinTopicDataTypeSupport:

Protected Member Functions

- final void **initialize_delegatel** (DataReaderDelegate delegate)

8.1.1 Detailed Description

Abstract superclass for all ∗TypeSupport classes for built-in types.

Author

rwarren

8.1.2 Member Function Documentation

8.1.2.1 initialize_delegatel()

```
final void initialize_delegateI (
            DataReaderDelegate delegate )  [protected]
```

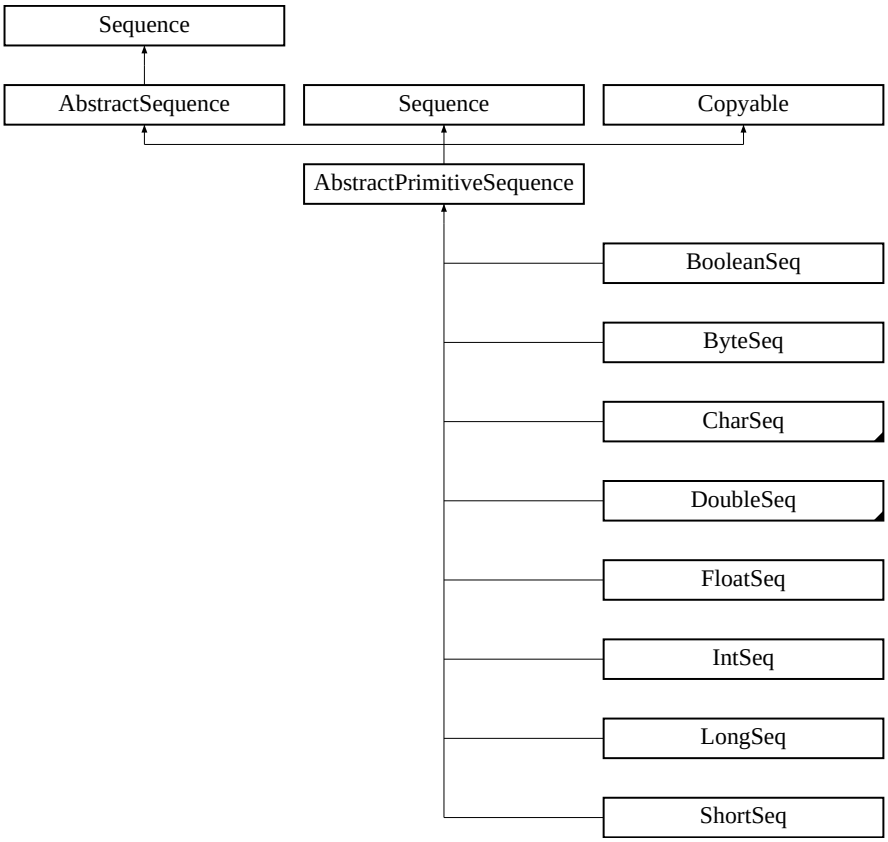
Subclasses should call this method immediately after chaining to the super constructor.

Exceptions

NullPointerException	if the delegate is null

8.2 AbstractPrimitiveSequence Class Reference

Inheritance diagram for AbstractPrimitiveSequence:



Public Member Functions

- final Class **getElementType** ()
- abstract void **add** (int index, Object element)

  Inserts the specified element at the specified position in this sequence.
- void **loan** (Object buffer, int new_length)

  Loan a contiguous buffer to this sequence.
- void **unloan** ()

  Return the loaned buffer in the sequence and set the maximum to 0.
- final boolean **hasOwnership** ()

  Return the value of the owned flag.
- final void **clear** ()
- final void **setSize** (int newSize)
- final int **size** ()
- final Object **copy_from** (Object src)

8.2.1 Detailed Description

A base class for sequences whose elements are of primitive types. Such sequences do not support null values.

8.2.2 Member Function Documentation

8.2.2.1 getElementType()

```
final Class getElementType ( )
```

Returns

the primitive type of this sequence, not the wrapper type.

See also

com.rti.dds.util.Sequence::getElementType() (p. 1667)

Reimplemented from **AbstractSequence** (p. 329).

8.2.2.2 add()

```
abstract void add (
            int index,
            Object element )  [abstract]
```

Inserts the specified element at the specified position in this sequence.

See also

java.util.List::add(int, java.lang.Object)

Reimplemented from **AbstractSequence** (p. 329).

Reimplemented in **BooleanSeq** (p. 365), **ByteSeq** (p. 401), **CharSeq** (p. 422), **DoubleSeq** (p. 830), **FloatSeq** (p. 1055), **IntSeq** (p. 1168), **LongSeq** (p. 1295), and **ShortSeq** (p. 1692).

8.2.2.3 loan()

```
void loan (
             Object buffer,
             int new_length )
```

Loan a contiguous buffer to this sequence.

This operation changes the `owned` flag of the sequence to com.rti.dds.infrastructure.false and also sets the underlying buffer used by the sequence. See the `User's Manual` for more information about sequences and memory ownership.

Use this method if you want to manage the memory used by the sequence yourself. You must provide an array of elements and integers indicating how many elements are allocated in that array (i.e. the maximum) and how many elements are valid (i.e. the length). The sequence will subsequently use the memory you provide and will not permit it to be freed by a call to com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

By default, a sequence you create owns its memory unless you explicitly loan memory of your own to it. In a very few cases, RTI Connext will return a sequence to you that has a loan; those cases are documented as such. For example, if you call **com.rti.ndds.example.FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) and pass in sequences with no loan and no memory allocated, RTI Connext will loan memory to your sequences which must be unloaned with **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094). See the documentation of those methods for more information.

Precondition

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.getMaximum == 0; i.e. the sequence has no memory allocated to it.

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.AbstractPrimitiveSequence.hasOwnership == com.rti.dds.↩ infrastructure.true; i.e. the sequence does not already have an outstanding loan

Postcondition

The sequence will store its elements in the buffer provided.

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.AbstractPrimitiveSequence.hasOwnership == com.rti.dds.↩ infrastructure.false

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size() == new_length

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.getMaximum == new_max

Parameters

buffer	The new buffer that the sequence will use. Must point to enough memory to hold new_max elements of type Foo. It may be NULL if new_max == 0.
new_length	The desired new length for the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.unloan , com.rti.dds.infrastructure.com.rti.dds.util.Sequence.↩ loan_discontiguous

References **Sequence.getMaximum()**.

8.2.2.4 unloan()

```
void unloan ( )
```

Return the loaned buffer in the sequence and set the maximum to 0.

This method affects only the state of this sequence; it does not change the contents of the buffer in any way.

Only the user who originally loaned a buffer should return that loan, as the user may have dependencies on that memory known only to them. Unloaning someone else's buffer may cause unspecified problems. For example, suppose a sequence is loaning memory from a custom memory pool. A user of the sequence likely has no way to release the memory back into the pool, so unloaning the sequence buffer would result in a resource leak. If the user were to then re-loan a different buffer, the original creator of the sequence would have no way to discover, when freeing the sequence, that the loan no longer referred to its own memory and would thus not free the user's memory properly, exacerbating the situation and leading to undefined behavior.

Precondition

owned == com.rti.dds.infrastructure.false

Postcondition

owned == com.rti.dds.infrastructure.true

maximum == 0

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.AbstractPrimitiveSequence.loan(Object, int), com.rti.↩ dds.infrastructure.com.rti.dds.util.Sequence.loan_discontiguous, com.rti.dds.infrastructure.com.rti.dds.util.↩ Sequence.Sequence.setMaximum

8.2.2.5 hasOwnership()

```
final boolean hasOwnership ( )
```

Return the value of the owned flag.

Returns

com.rti.dds.infrastructure.true if sequence owns the underlying buffer, or com.rti.dds.infrastructure.false if it has an outstanding loan.

8.2.2.6 clear()

```
final void clear ( )
```

Set the logical size of this sequence to zero. This method does not generate any garbage for collection.

See also

java.util.Collection::clear()

Referenced by **AbstractPrimitiveSequence.copy_from()**.

8.2.2.7 setSize()

```
final void setSize (
            int newSize )
```

Set the logical size of this sequence to the given value.

Parameters

newSize	the new logical size of this sequence; it must be less than or equal to the maximum allocated length of the underlying array.

Exceptions

IndexOutOfBoundsException	if the new size is less than zero or greater than the allocated length of the array.

See also

AbstractSequence::getMaximum() (p. 1665)

References **Sequence.getMaximum()**.

Referenced by **DynamicData.get_uint8_seq()**, **DynamicData.get_uint_seq()**, **DynamicData.get_ushort_seq()**, **DynamicData.set_uint8_seq()**, **DynamicData.set_uint_seq()**, **DynamicData.set_ulong_seq()**, **DynamicData.set_ushort_seq()**, and **DynamicData.to_cdr_buffer()**.

8.2.2.8 size()

```
final int size ( )
```

The logical size of this sequence.

Referenced by **AbstractPrimitiveSequence.copy_from()**, **DynamicData.get_boolean_seq()**, **DynamicData.get_byte_seq()**, **DynamicData.get_char_seq()**, **DynamicData.get_double_seq()**, **DynamicData.get_float_seq()**, **DynamicData.get_int_seq()**, **DynamicData.get_long_seq()**, **DynamicData.get_short_seq()**, **DynamicData.get_uint8_seq()**, **DynamicData.get_uint_seq()**, **DynamicData.get_ulong_seq()**, **DynamicData.get_ushort_seq()**, **DynamicData.set_boolean_seq()**, **DynamicData.set_byte_seq()**, **DynamicData.set_char_seq()**, **DynamicData.set_double_seq()**, **DynamicData.set_float_seq()**, **DynamicData.set_int_seq()**, **DynamicData.set_long_seq()**, **DynamicData.set_short_seq()**, **DynamicData.set_uint8_seq()**, **DynamicData.set_uint_seq()**, **DynamicData.set_ulong_seq()**, **DynamicData.set_ushort_seq()**, **BytesDataWriter.write()**, **KeyedBytesDataWriter.write()**, **BytesDataWriter.write_w_timestamp()**, and **KeyedBytesDataWriter.write_w_timestamp()**.

8.2.2.9 copy_from()

```
final Object copy_from (
             Object src )
```

Implementation of the `Copyable` interface.

Parameters

src	An **`AbstractPrimitiveSequence`** (p. 322) which contains the data to be copied.

Returns

`this`

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not a **`Sequence`** (p. 1664) OR if one of the objects contained in the **`Sequence`** (p. 1664) is not of the expected type.

Implements **Copyable** (p. 445).

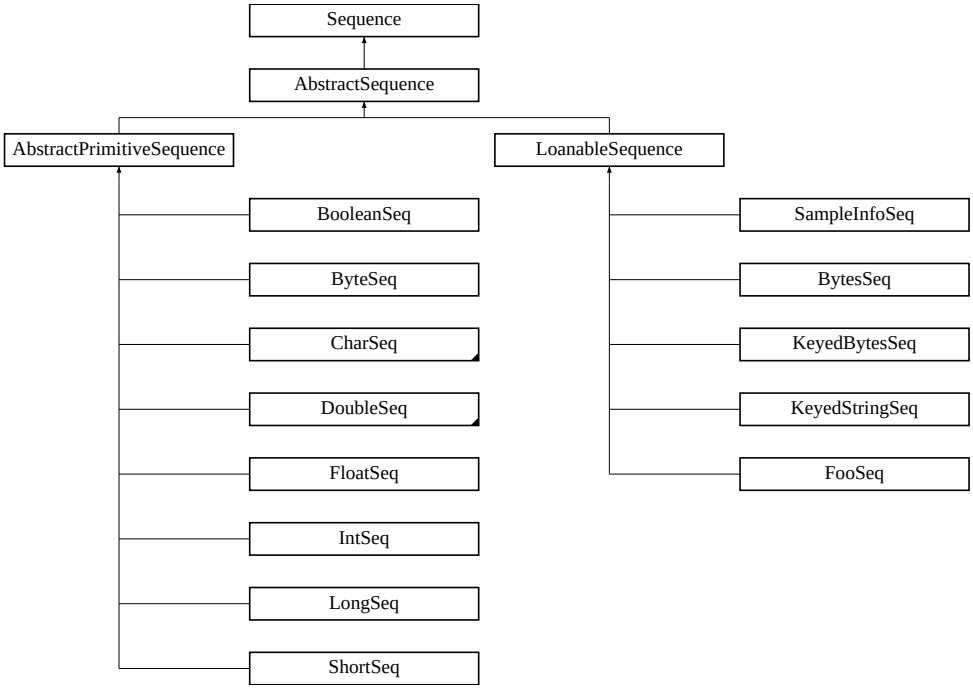
References **AbstractPrimitiveSequence.clear()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.3 AbstractSequence Class Reference

Abstract sequence.

Inheritance diagram for AbstractSequence:



Public Member Functions

- void **setMaximum** (int new_max)

  Resize this sequence to a new desired maximum.
- Class **getElementType** ()
- void **add** (int index, Object element)

  Inserts the specified element at the specified position in this sequence.
- boolean **add** (Object element)

  Appends the specified element to the end of this sequence.
- final Object **remove** (int index)

  Remove the element at the given index by shifting all subsequent elements "left" by one.

8.3.1 Detailed Description

Abstract sequence.

8.3.2 Member Function Documentation

8.3.2.1 setMaximum()

```
void setMaximum (
            int new_max )
```

Resize this sequence to a new desired maximum.

This operation does nothing if the new desired maximum matches the current maximum.

Note: If you add an element with **add()** (p. 329), the sequence's size is increased implicitly.

Postcondition

length == MINIMUM(original length, new_max)

Parameters

new_max	Must be >= 0.

Implements **Sequence** (p. 1666).

Reimplemented in **LoanableSequence** (p. 1251).

Referenced by **DynamicData.get_uint8_seq()**, **DynamicData.get_uint_seq()**, **DynamicData.get_ushort_seq()**, and **DynamicData.to_cdr_buffer()**.

8.3.2.2 getElementType()

```
Class getElementType ( )
```

Returns

a common supertype for all elements in this sequence.

Implements **Sequence** (p. 1667).

Reimplemented in **AbstractPrimitiveSequence** (p. 323).

8.3.2.3 add() [1/2]

```
void add (
            int index,
            Object element )
```

Inserts the specified element at the specified position in this sequence.

See also

java.util.List::add(int, java.lang.Object)

Reimplemented in **BooleanSeq** (p. 365), **ByteSeq** (p. 401), **CharSeq** (p. 422), **DoubleSeq** (p. 830), **FloatSeq** (p. 1055), **IntSeq** (p. 1168), **LongSeq** (p. 1295), **ShortSeq** (p. 1692), and **AbstractPrimitiveSequence** (p. 323).

Referenced by **BytesSeq.copy_from()**, **KeyedBytesSeq.copy_from()**, **KeyedStringSeq.copy_from()**, and **Foo↩ Seq.copy_from()**.

8.3.2.4 add() [2/2]

```
boolean add (
            Object element )
```

Appends the specified element to the end of this sequence.

See also

java.util.List::add(java.lang.Object)

8.3.2.5 remove()

```
final Object remove (
            int index )
```

Remove the element at the given index by shifting all subsequent elements "left" by one.

See also

java.util.List::remove(int)

8.4 AcknowledgmentInfo Class Reference

Information about an application-level acknowledged sample.

Inherits Status.

Public Attributes

- final **InstanceHandle_t subscription_handle**

  Subscription handle of the acknowledging ***com.rti.dds.subscription.DataReader*** *(p. 450).*
- final **SampleIdentity_t sample_identity**

  Identity of the sample being acknowledged.
- boolean **valid_response_data**

  Flag indicating validity of the user response data in the acknowledgment.
- final **AckResponseData_t response_data**

  User data payload of application-level acknowledgment message.

8.4.1 Detailed Description

Information about an application-level acknowledged sample.

When acknowledging a sample, the reader provides the writer with information about the sample being acknowledged. The **AcknowledgmentInfo** (p. 330) structure provides the identity and cookie of the sample being acknowledged, as well as user data payload provided by the reader.

8.4.2 Member Data Documentation

8.4.2.1 subscription_handle

```
final  InstanceHandle_t subscription_handle
```

Subscription handle of the acknowledging **com.rti.dds.subscription.DataReader** (p. 450).

8.4.2.2 sample_identity

```
final  SampleIdentity_t sample_identity
```

Identity of the sample being acknowledged.

See also

com.rti.dds.infrastructure.SampleIdentity_t (p. 1632)

8.4.2.3 valid_response_data

```
boolean valid_response_data
```

Flag indicating validity of the user response data in the acknowledgment.

This flag is true when the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.min_app_ack_response_↩ keep_duration** (p. 1604) has not yet elapsed for the acknowledgment's response data.

The flag is false when that duration has elapsed for the response data.

8.4.2.4 response_data

```
final AckResponseData_t response_data
```

User data payload of application-level acknowledgment message.

Response data set by **com.rti.dds.subscription.DataReader** (p. 450) when sample was acknowledged.

8.5 AckResponseData_t Class Reference

Data payload of an application-level acknowledgment.

Inherits Struct.

Public Attributes

- final **ByteSeq value**
  a sequence of octets

8.5.1 Detailed Description

Data payload of an application-level acknowledgment.

8.5.2 Member Data Documentation

8.5.2.1 value

```
final ByteSeq value
```

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0, **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_app↩ _ack_response_length** (p. 539)],

8.6 ActivityContext Class Reference

Activity Context APIs.

Static Public Member Functions

- static void **set_attribute_mask** (int attribute_mask)
  Set the com.rti.ndds.config.ActivityContextAttributeKindMask of the Activity Context.

8.6.1 Detailed Description

Activity Context APIs.

8.7 ActivityContextAttributeKind Class Reference

The resources of the **Activity Context** (p. 288) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.

Static Public Attributes

- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX** = 0x1
  Provide the entity GUID prefix to the resource of the Activity Context.
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC** = 0x2
  Provide the Topic to the resource of the Activity Context. The topic attribute is specified by "Topic".
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE** = 0x4
  Provide the data type to the resource of the Activity Context. The type attribute is specified by "Type".
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND** = 0x8
  Provide the entity kind to the resource of the Activity Context. The kind attribute is specified by "Entity".
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID** = 0x10
  Provide the domain ID to the resource of the Activity Context. The domain attribute is specified by "Domain".
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME** = 0x20
  Provide the entity name to the resource of the Activity Context. The name attribute is specified by "Name".
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT** = 0x3F
  Provide the default attributes of the resource of the Activity Context.
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE** = 0x0
  Not provide any attribute of the resource of the Activity Context.
- static final int **NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL** = 0x3F
  Provide all the possibles attributes of the resource of the Activity Context.

8.7.1 Detailed Description

The resources of the **Activity Context** (p. 288) can have multiple associated attributes. Those attributes provide extra information about the entity such as GUID prefix, Topic, data type, entity kind, entity name and domain ID. They are used to indicate what attributes of the resources are included in the activity context.

8.7.2 Member Data Documentation

8.7.2.1 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX = 0x1  [static]
```

Provide the entity GUID prefix to the resource of the Activity Context.

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The GUID prefix is 0X101A76B,0X79E5D71,0X50EE914. If the bit **com.rti.ndds.config.ActivityContext↩ AttributeKind.NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX** (p. 334) is not set, the string representation will not show the GUID prefix:
  `[:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`

8.7.2.2 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC = 0x2  [static]
```

Provide the Topic to the resource of the Activity Context. The topic attribute is specified by "Topic".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The Topic is "test." If the bit **com.rti.ndds.config.ActivityContextAttributeKind.NDDS_CONFIG_ACTIVITY_↩ CONTEXT_ATTRIBUTE_TOPIC** (p. 334) is not set, the string representation will not show the Topic:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Type=Foo,Domain=1}|Write]`

8.7.2.3 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE = 0x4  [static]
```

Provide the data type to the resource of the Activity Context. The type attribute is specified by "Type".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The data type is "Foo." If the bit **com.rti.ndds.config.ActivityContextAttributeKind.NDDS_CONFIG_↩ ACTIVITY_CONTEXT_ATTRIBUTE_TYPE** (p. 334) is not set, the string representation will not show the data type:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Domain=1}|Write]`

8.7.2.4 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND = 0x8  [static]
```

Provide the entity kind to the resource of the Activity Context. The kind attribute is specified by "Entity".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The entity kind is "Writer." If the bit **com.rti.ndds.config.ActivityContextAttributeKind.NDDS_CONFIG_↩ ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND** (p. 335) is not set, the string representation will not show the entity kind:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Topic=test,Type=Foo,Domain=1}|Write]`

8.7.2.5 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID = 0x10  [static]
```

Provide the domain ID to the resource of the Activity Context. The domain attribute is specified by "Domain".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The domain ID is "1." If the bit **com.rti.ndds.config.ActivityContextAttributeKind.NDDS_CONFIG_↩ ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID** (p. 335) is not set, the string representation will not show the domain ID:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo}|Write]`

8.7.2.6 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME = 0x20  [static]
```

Provide the entity name to the resource of the Activity Context. The name attribute is specified by "Name".

For example:

- For the following string representation of the context:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`
- The entity name is "testDataWriterName." If the bit **com.rti.ndds.config.ActivityContextAttributeKind.NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME** (p. 335) is not set, the string representation will not show the entity name:
  `[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]`

8.7.2.7 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT = 0x3F  [static]
```

Provide the default attributes of the resource of the Activity Context.

8.7.2.8 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE = 0x0  [static]
```

Not provide any attribute of the resource of the Activity Context.

8.7.2.9 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL

```
final int NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL = 0x3F  [static]
```

Provide all the possibles attributes of the resource of the Activity Context.

8.8 AllocationSettings_t Class Reference

Resource allocation settings.

Inherits Struct.

Public Member Functions

- **AllocationSettings_t** (int **initial_count**, int **max_count**, int **incremental_count**)
  Constructor with the given initial, maximum and incremental values.

Public Attributes

- int **initial_count**
  The initial count of resources.
- int **max_count**
  The maximum count of resources.
- int **incremental_count**
  The incremental count of resources.

8.8.1 Detailed Description

Resource allocation settings.

QoS:

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy (p. 803)

8.8.2 Constructor & Destructor Documentation

8.8.2.1 AllocationSettings_t()

```
AllocationSettings_t (
            int initial_count,
            int max_count,
            int incremental_count )
```

Constructor with the given initial, maximum and incremental values.

References **AllocationSettings_t.incremental_count**, **AllocationSettings_t.initial_count**, and **Allocation↩ Settings_t.max_count**.

8.8.3 Member Data Documentation

8.8.3.1 initial_count

```
int initial_count
```

The initial count of resources.

The initial resources to be allocated.

[default] It depends on the case.

[range] [0, 1 million], < max_count, (or = max_count only if increment_count == 0)

Referenced by **AllocationSettings_t.AllocationSettings_t()**.

8.8.3.2 max_count

```
int max_count
```

The maximum count of resources.

The maximum resources to be allocated.

[default] Depends on the case.

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), > initial_count (or = initial_count only if increment_count == 0)

Referenced by **AllocationSettings_t.AllocationSettings_t()**.

8.8.3.3 incremental_count

```
int incremental_count
```

The incremental count of resources.

The resource to be allocated when more resources are needed.

[default] Depends on the case.

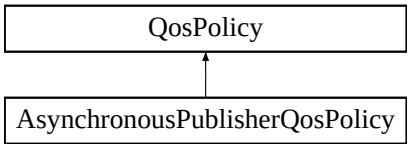
[range] -1 (Double the amount of extra memory allocated each time memory is needed) or [1,1 million] (or = 0 only if initial_count == max_count)

Referenced by **AllocationSettings_t.AllocationSettings_t()**.

8.9 AsynchronousPublisherQosPolicy Class Reference

Configures the mechanism that sends user data in an external middleware thread.

Inheritance diagram for AsynchronousPublisherQosPolicy:



Public Attributes

- boolean **disable_asynchronous_write**

  Disable asynchronous publishing.
- final **ThreadSettings_t thread**

  Settings of the publishing thread.
- boolean **disable_asynchronous_batch**

  Disable asynchronous batch flushing.
- final **ThreadSettings_t asynchronous_batch_thread**

  Settings of the batch flushing thread.
- boolean **disable_topic_query_publication**

  Disable topic query publication.
- final **ThreadSettings_t topic_query_publication_thread**

  *Settings of the **com.rti.dds.subscription.TopicQuery*** (p. *1830*) *publication thread.*

8.9.1 Detailed Description

Configures the mechanism that sends user data in an external middleware thread.

Specifies the asynchronous publishing and asynchronous batch flushing settings of the **com.rti.dds.publication.↩ Publisher** (p. 1466) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **com.↩ rti.dds.publication.DataWriter** (p. 553) entities belonging to this **com.rti.dds.publication.Publisher** (p. 1466). If so, the publisher will spawn up to two threads, one for asynchronous publishing and one for asynchronous batch flushing.

This policy also configures the settings of the **com.rti.dds.subscription.TopicQuery** (p. 1830) publication thread. The publisher will spawn this thread only if one or more DataWriters enable TopicQueries.

See also

com.rti.dds.infrastructure.BatchQosPolicy (p. 355).

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496).

Entity:

com.rti.dds.publication.Publisher (p. 1466)

Properties:

RxO (p. 256) = N/A

Changeable (p. 256) = **NO** (p. 256)

8.9.2 Usage

You can use this QoS policy to reduce the amount of time your application thread spends sending data.

You can also use it, along with **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496) and a **com.rti.dds.↩ publication.FlowController** (p. 1055), to send large data reliably. "Large" in this context means that the data that cannot be sent as a single packet by a network transport. For example, to send data larger than 63K reliably using UDP/IP, you must configure RTI Connext to fragment the data and send it asynchronously.

The asynchronous *publisher* thread is shared by all com.rti.dds.infrastructure.PublishModeQosPolicyKind.Publish↩ ModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 553) instances that belong to this publisher and handles their data transmission chores.

The asynchronous *batch flushing* thread is shared by all **com.rti.dds.publication.DataWriter** (p. 553) instances with batching enabled that belong to this publisher.

This QoS policy also allows you to adjust the settings of the asynchronous publishing and the asynchronous batch flushing threads. To use different threads for two different **com.rti.dds.publication.DataWriter** (p. 553) entities, the instances must belong to different **com.rti.dds.publication.Publisher** (p. 1466) instances.

A **com.rti.dds.publication.Publisher** (p. 1466) must have asynchronous publishing enabled for its **com.rti.dds.↩ publication.DataWriter** (p. 553) instances to write asynchronously.

A **com.rti.dds.publication.Publisher** (p. 1466) must have asynchronous batch flushing enabled in order to flush the batches of its **com.rti.dds.publication.DataWriter** (p. 553) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **com.rti.dds.publication.DataWriter** (p. 553) instance with batching enabled is created from this **com.rti.dds.publication.Publisher** (p. 1466).

8.9.3 Member Data Documentation

8.9.3.1 disable_asynchronous_write

```
boolean disable_asynchronous_write
```

Disable asynchronous publishing.

If set to com.rti.dds.infrastructure.true, any **com.rti.dds.publication.DataWriter** (p. 553) created with com.rti.dds.↩ infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

[default] com.rti.dds.infrastructure.false

8.9.3.2 thread

```
final ThreadSettings_t thread
```

Settings of the publishing thread.

There is only one asynchronous publishing thread per **com.rti.dds.publication.Publisher** (p. 1466).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 267)

8.9.3.3 disable_asynchronous_batch

```
boolean disable_asynchronous_batch
```

Disable asynchronous batch flushing.

If set to com.rti.dds.infrastructure.true, any **com.rti.dds.publication.DataWriter** (p. 553) created with batching enabled will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

If **com.rti.dds.infrastructure.BatchQosPolicy.max_flush_delay** (p. 358) is different than **com.rti.dds.infrastructure.↩ Duration_t.DURATION_INFINITE** (p. 846), **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.↩ disable_asynchronous_batch** (p. 341) must be set com.rti.dds.infrastructure.false.

[default] com.rti.dds.infrastructure.false

8.9.3.4 asynchronous_batch_thread

`final` **`ThreadSettings_t`** `asynchronous_batch_thread`

Settings of the batch flushing thread.

There is only one asynchronous batch flushing thread per **com.rti.dds.publication.Publisher** (p. 1466).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 267)

8.9.3.5 disable_topic_query_publication

`boolean disable_topic_query_publication`

Disable topic query publication.

If set to com.rti.dds.infrastructure.true, any **com.rti.dds.publication.DataWriter** (p. 553) created with **com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy.enable** (p. 1834) set to com.rti.dds.infrastructure.true will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

[default] com.rti.dds.infrastructure.false

8.9.3.6 topic_query_publication_thread

`final` **`ThreadSettings_t`** `topic_query_publication_thread`

Settings of the **com.rti.dds.subscription.TopicQuery** (p. 1830) publication thread.

There is only one TopicQuery publication thread per **com.rti.dds.publication.Publisher** (p. 1466). This thread will exist as long as one or more **com.rti.dds.publication.DataWriter** (p. 553) enables TopicQueries (via **com.rti.dds.publication.DataWriterQos.topic_query_dispatch** (p. 622)).

[default] priority below normal.
The actual value depends on your architecture:

For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`. **[default]** The actual value depends on your architecture:

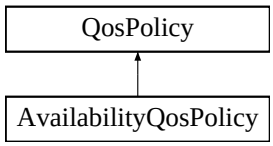
For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask = **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 267)

8.10 AvailabilityQosPolicy Class Reference

Configures the availability of data.

Inheritance diagram for AvailabilityQosPolicy:



Public Attributes

- boolean **enable_required_subscriptions**

  *Enables support for required subscriptions in a **com.rti.dds.publication.DataWriter*** (p. *553*).
- final **Duration_t max_data_availability_waiting_time**

  Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
- final **Duration_t max_endpoint_availability_waiting_time**

  Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
- final **EndpointGroupSeq required_matched_endpoint_groups**

  A sequence of endpoint groups.

8.10.1 Detailed Description

Configures the availability of data.

Entity:

com.rti.dds.subscription.DataReader (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **YES** (p. 256) (only on a **com.rti.dds.publication.DataWriter** (p. 553) except for the member **com.rti.dds.infrastructure.AvailabilityQosPolicy.enable_required_subscriptions** (p. 345))

8.10.2 Usage

This QoS policy is used in the context of two features:

- Collaborative DataWriters
- Required Subscriptions

Collaborative DataWriters

The Collaborative DataWriters feature allows having multiple DataWriters publishing samples from a common logical data source. The DataReaders will combine the samples coming from the DataWriters in order to reconstruct the correct order at the source.

This QoS policy allows you to configure the ordering and combination process in the DataReader and can be used to support two different use cases:

- **Ordered delivery of samples in high-availability scenarios** One example of this is RTI Persistence Service. When a late-joining DataReader configured with **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) set to com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS joins a DDS domain, it will start receiving historical samples from multiple DataWriters. For example, if the original DataWriter is still alive, the newly created DataReader will receive samples from the original DataWriter and one or more RTI Persistence Service DataWriters (PRSTDataWriters). This policy can be used to configure the sample ordering process on the DataReader.

- **Ordered delivery of samples in load-balanced scenarios** Multiple instances of the same application can work together to process and deliver samples. When the samples arrive through different data-paths out of order, the DataReader will be able to reconstruct the order at the destination. An example of this is when multiple instances of RTI Persistence Service are used to persist the data. Persisting data to a database on disk can be a bottleneck for throughput. You can improve scalability and performance by dividing the workload across different instances of RTI Persistence Service that use different databases. For example, samples larger than 10 are persisted by Persistence Service 1, samples less than or equal to 10 are persisted by Persistence Service 2.

- **Ordered delivery of samples with Group Ordered Access** This policy can also be used to configure the sample ordering process when the Subscriber is configured with **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1379) access_scope set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.Presentation↩QosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS. In this case, the Subscriber must deliver in order the samples published by a group of DataWriters that belong to the same Publisher and have access_scope set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScope↩Kind.GROUP_PRESENTATION_QOS.

Each sample published in a DDS domain for a given logical data source is uniquely identified by a pair (virtual GUID, virtual sequence number). Samples from the same data source (same virtual GUID) can be published by different DataWriters. A DataReader will deliver a sample (VGUIDn, VSNm) to the application if one of the following conditions is satisfied:

- (VGUIDn, VSNm-1) has already been delivered to the application.
- All the known DataWriters publishing VGUIDn have announced that they do not have (VGUIDn, VSNm-1).

- None of the known DataWriters publishing GUIDn have announced potential availability of (VGUIDn, VSNm-1) and both timeouts in this QoS policy have expired.

A DataWriter announces potential availability of samples by using virtual heartbeats (HBs).

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is set to com.rti.dds.↩ infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION↩ _QOS or com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScope↩ Kind.INSTANCE_PRESENTATION_QOS, the virtual HB contains information about the samples contained in the **com.rti.dds.publication.DataWriter** (p. 553) history.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is set to com.rti.dds.↩ infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS, the virtual HB contains information about all DataWriters in the **com.rti.dds.publication.Publisher** (p. 1466).

The frequency at which virtual HBs are sent is controlled by the protocol parameters **com.rti.dds.infrastructure.Rtps**↩ **ReliableWriterProtocol_t.virtual_heartbeat_period** (p. 1610) and **com.rti.dds.infrastructure.RtpsReliableWriter**↩ **Protocol_t.samples_per_virtual_heartbeat** (p. 1610).

Required Subscriptions

In the context of Required Subscriptions, this QoS policy can be used to configure a set of Required Subscriptions on a **com.rti.dds.publication.DataWriter** (p. 553).

Required subscriptions are preconfigured, named subscriptions that may leave and subsequently rejoin the network from time to time, at the same or different physical locations. Any time a required subscription is disconnected, any samples that would have been delivered to it are stored for delivery if and when the subscription rejoins the network.

8.10.3 Consistency

For a DataWriter, the setting of **AVAILABILITY** (p. 169) must be set consistenly with that of the **RELIABILITY** (p. 258) and **DURABILITY** (p. 230).

If **com.rti.dds.infrastructure.AvailabilityQosPolicy.enable_required_subscriptions** (p. 345) is set to com.rti.dds.↩ infrastructure.true, **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) must be set to **com.rti.dds.↩ infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), **com.rti.dds.infrastructure.↩ DurabilityQosPolicy** (p. 830) must be set to a value different than com.rti.dds.infrastructure.DurabilityQosPolicyKind.↩ DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS, and **com.rti.dds.infrastructure.DurabilityQosPolicy.↩ writer_depth** (p. 834) must be set to either **com.rti.dds.infrastructure.DurabilityQosPolicy.AUTO_WRITER_DEPTH** (p. 231) or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

8.10.4 Member Data Documentation

8.10.4.1 enable_required_subscriptions

```
boolean enable_required_subscriptions
```

Enables support for required subscriptions in a **com.rti.dds.publication.DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.false

8.10.4.2 max_data_availability_waiting_time

```
final Duration_t max_data_availability_waiting_time
```

Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.

Collaborative DataWriters

A sample identified by (VGUIDn, VSNm) will be delivered to the application if this timeout expires for the sample and the following two conditions are satisfied:

- None of the known DataWriters publishing VGUIDn have announced potential availability of (VGUIDn, VSNm-1).
- The DataWriters for all the endpoint groups specified in **required_matched_endpoint_groups** (p. 346) have been discovered or **max_endpoint_availability_waiting_time** (p. 346) has expired.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] com.rti.dds.infrastructure.Duration_t.AUTO (**com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) for com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScope↩ Kind.GROUP_PRESENTATION_QOS. Otherwise, 0 seconds)

[range] [0, **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)], com.rti.dds.infrastructure.↩ Duration_t.AUTO

8.10.4.3 max_endpoint_availability_waiting_time

```
final Duration_t max_endpoint_availability_waiting_time
```

Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).

Collaborative DataWriters

The set of endpoint groups that are required to provide samples for a data source can be configured using **required_↩ matched_endpoint_groups** (p. 346).

A non-consecutive sample identified by (VGUIDn, VSNm) cannot be delivered to the application unless DataWriters for all the endpoint groups in **required_matched_endpoint_groups** (p. 346) are discovered or this timeout expires.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] com.rti.dds.infrastructure.Duration_t.AUTO (**com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) for com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScope↩ Kind.GROUP_PRESENTATION_QOS. Otherwise, 0 seconds)

[range] [0, **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)], com.rti.dds.infrastructure.↩ Duration_t.AUTO

8.10.4.4 required_matched_endpoint_groups

```
final EndpointGroupSeq required_matched_endpoint_groups
```

A sequence of endpoint groups.

Collaborative DataWriters

In the context of Collaborative DataWriters, it specifies the set of endpoint groups that are expected to provide samples for the same data source.

The quorum count in a group represents the number of DataWriters that must be discovered for that group before the DataReader is allowed to provide non consecutive samples to the application.

A DataWriter becomes a member of an endpoint group by configuring the role_name in **com.rti.dds.publication.↩ DataWriterQos.publication_name** (p. 622).

Required Subscriptions

In the context of Required Subscriptions, it specifies the set of Required Subscriptions on a **com.rti.dds.publication.↩ DataWriter** (p. 553).

Each Required Subscription is specified by a name and a quorum count.

The quorum count represents the number of DataReaders that have to acknowledge the sample before it can be considered fully acknowledged for that Required Subscription.

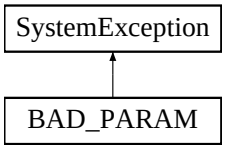
A DataReader is associated with a Required Subscription by configuring the role_name in **com.rti.dds.subscription.↩ DataReaderQos.subscription_name** (p. 525).

[default] Empty sequence

8.11 BAD_PARAM Class Reference

Exception thrown when a parameter passed to a call is considered illegal.

Inheritance diagram for BAD_PARAM:



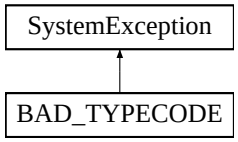
8.11.1 Detailed Description

Exception thrown when a parameter passed to a call is considered illegal.

8.12 BAD_TYPECODE Class Reference

The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.

Inheritance diagram for BAD_TYPECODE:



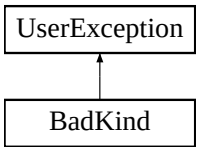
8.12.1 Detailed Description

The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.

8.13 BadKind Class Reference

The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.

Inheritance diagram for BadKind:



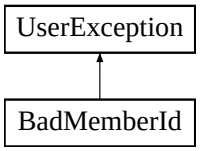
8.13.1 Detailed Description

The exception **BadKind** (p. 348) is thrown when an inappropriate operation is invoked on a TypeCode object.

8.14 BadMemberId Class Reference

The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member ID is invalid.

Inheritance diagram for BadMemberId:



8.14.1 Detailed Description

The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member ID is invalid.

This failure can occur, for example, when querying a field by ID when no such ID is defined in the type.

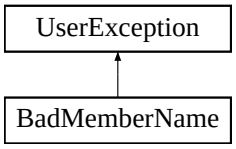
See also

com.rti.dds.infrastructure.ExceptionCode_t.BadMemberName

8.15 BadMemberName Class Reference

The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member name is invalid.

Inheritance diagram for BadMemberName:



8.15.1 Detailed Description

The specified **com.rti.dds.typecode.TypeCode** (p. 1873) member name is invalid.

This failure can occur, for example, when querying a field by name when no such name is defined in the type.

See also

com.rti.dds.infrastructure.ExceptionCode_t.BadMemberId

8.16 Base64 Class Reference

Static Public Member Functions

- static final char[ ] **encodeToChar** (byte[ ] sArr, boolean lineSep)
- static final byte[ ] **decode** (char[ ] sArr)
- static final byte[ ] **decodeFast** (char[ ] sArr)
- static final byte[ ] **encodeToByte** (byte[ ] sArr, boolean lineSep)
- static final byte[ ] **decode** (byte[ ] sArr)
- static final byte[ ] **decodeFast** (byte[ ] sArr)
- static final String **encodeToString** (byte[ ] sArr, boolean lineSep)
- static final byte[ ] **decode** (String str)
- static final byte[ ] **decodeFast** (String s)

8.16.1 Detailed Description

A very fast and memory efficient class to encode and decode to and from BASE64 in full accordance with RFC 2045.

On Windows XP sp1 with 1.4.2_04 and later ;), this encoder and decoder is about 10 times faster on small arrays (10 - 1000 bytes) and 2-3 times as fast on larger arrays (10000 - 1000000 bytes) compared to `sun.misc.↩ Encoder()/Decoder()`.

On byte arrays the encoder is about 20% faster than Jakarta Commons **Base64** (p. 349) Codec for encode and about 50% faster for decoding large arrays. This implementation is about twice as fast on very small arrays (&lt 30 bytes). If source/destination is a `String` this version is about three times as fast due to the fact that the Commons Codec result has to be recoded to a `String` from `byte[]`, which is very expensive.

This encode/decode algorithm doesn't create any temporary arrays as many other codecs do, it only allocates the resulting array. This produces less garbage and it is possible to handle arrays twice as large as algorithms that create a temporary array. (E.g. Jakarta Commons Codec). It is unknown whether Sun's `sun.misc.Encoder()/↩ Decoder()` produce temporary arrays but since performance is quite low it probably does.

The encoder produces the same output as the Sun one except that the Sun's encoder appends a trailing line separator if the last character isn't a pad. Unclear why but it only adds to the length and is probably a side effect. Both are in conformance with RFC 2045 though.
Commons codec seem to always att a trailing line separator.

Note! The encode/decode method pairs (types) come in three versions with the **exact** same algorithm and thus a lot of code redundancy. This is to not create any temporary arrays for transcoding to/from different format types. The methods not used can simply be commented out.

There is also a "fast" version of all decode methods that works the same way as the normal ones, but har a few demands on the decoded input. Normally though, these fast verions should be used if the source if the input is known and it hasn't bee tampered with.

If you find the code useful or you find a bug, please send me a note at base64 @ miginfocom . com.

8.16.2 Licence (BSD):

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and

the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the MiG InfoCom AB nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Version

2.2

Author

Mikael Grev Date: 2004-aug-02 Time: 11:31:11

8.16.3 Member Function Documentation

8.16.3.1 encodeToChar()

```
static final char[] encodeToChar (
            byte[] sArr,
            boolean lineSep )  [static]
```

Encodes a raw byte array into a BASE64 `char[]` representation i accordance with RFC 2045.

Parameters

sArr	The bytes to convert. If `null` or length 0 an empty array will be returned.
lineSep	Optional "\r\n" after 76 characters, unless end of file. No line separator will be in breach of RFC 2045 which specifies max 76 per line but will be a little faster.

Returns

A BASE64 encoded array. Never `null`.

Referenced by **Base64.encodeToString()**.

8.16.3.2 decode() [1/3]

```
static final byte[] decode (
            char[] sArr )  [static]
```

Decodes a BASE64 encoded char array. All illegal characters will be ignored and can handle both arrays with and without line separators.

Parameters

sArr	The source array. `null` or length 0 will return an empty array.

Returns

The decoded array of bytes. May be of length 0. Will be `null` if the legal characters (including '=') isn't divideable by 4. (I.e. definitely corrupted).

8.16.3.3 decodeFast() [1/3]

```
static final byte[] decodeFast (
            char[] sArr )  [static]
```

Decodes a BASE64 encoded char array that is known to be resonably well formatted. The method is about twice as fast as **decode(char[])** (p. 351). The preconditions are:

- The array must have a line length of 76 chars OR no line separators at all (one line).
- Line separator must be "\r\n", as specified in RFC 2045
- The array must not contain illegal characters within the encoded string
- The array CAN have illegal characters at the beginning and end, those will be dealt with appropriately.

Parameters

sArr	The source array. Length 0 will return an empty array. `null` will throw an exception.

Returns

The decoded array of bytes. May be of length 0.

8.16.3.4 encodeToByte()

```
static final byte[] encodeToByte (
            byte[] sArr,
            boolean lineSep )  [static]
```

Encodes a raw byte array into a BASE64 `byte[]` representation i accordance with RFC 2045.

Parameters

sArr	The bytes to convert. If `null` or length 0 an empty array will be returned.
lineSep	Optional "\r\n" after 76 characters, unless end of file. No line separator will be in breach of RFC 2045 which specifies max 76 per line but will be a little faster.

Returns

A BASE64 encoded array. Never `null`.

8.16.3.5 decode() [2/3]

```
static final byte[] decode (
            byte[] sArr )  [static]
```

Decodes a BASE64 encoded byte array. All illegal characters will be ignored and can handle both arrays with and without line separators.

Parameters

sArr	The source array. Length 0 will return an empty array. `null` will throw an exception.

Returns

The decoded array of bytes. May be of length 0. Will be `null` if the legal characters (including '=') isn't divideable by 4. (I.e. definitely corrupted).

8.16.3.6 decodeFast() [2/3]

```
static final byte[] decodeFast (
            byte[] sArr )  [static]
```

Decodes a BASE64 encoded byte array that is known to be resonably well formatted. The method is about twice as fast as **decode(byte[])** (p. 353). The preconditions are:

- The array must have a line length of 76 chars OR no line separators at all (one line).
- Line separator must be "\r\n", as specified in RFC 2045
- The array must not contain illegal characters within the encoded string
- The array CAN have illegal characters at the beginning and end, those will be dealt with appropriately.

Parameters

sArr	The source array. Length 0 will return an empty array. `null` will throw an exception.

Returns

The decoded array of bytes. May be of length 0.

8.16.3.7 encodeToString()

```
static final String encodeToString (
            byte[] sArr,
            boolean lineSep )  [static]
```

Encodes a raw byte array into a BASE64 `String` representation i accordance with RFC 2045.

Parameters

sArr	The bytes to convert. If `null` or length 0 an empty array will be returned.
lineSep	Optional "\r\n" after 76 characters, unless end of file. No line separator will be in breach of RFC 2045 which specifies max 76 per line but will be a little faster.

Returns

A BASE64 encoded array. Never `null`.

References **Base64.encodeToChar()**.

8.16.3.8 decode() [3/3]

```
static final byte[] decode (
            String str )  [static]
```

Decodes a BASE64 encoded `String`. All illegal characters will be ignored and can handle both strings with and without line separators.
Note! It can be up to about 2x the speed to call `decode(str.toCharArray())` instead. That will create a temporary array though. This version will use `str.charAt(i)` to iterate the string.

Parameters

str	The source string. `null` or length 0 will return an empty array.

Returns

The decoded array of bytes. May be of length 0. Will be `null` if the legal characters (including '=') isn't divideable by 4. (I.e. definitely corrupted).

8.16.3.9 decodeFast() [3/3]

```
static final byte[] decodeFast (
            String s )  [static]
```

Decodes a BASE64 encoded string that is known to be resonably well formatted. The method is about twice as fast as **`decode(String)`** (p. 354). The preconditions are:

- The array must have a line length of 76 chars OR no line separators at all (one line).
- Line separator must be "\r\n", as specified in RFC 2045
- The array must not contain illegal characters within the encoded string
- The array CAN have illegal characters at the beginning and end, those will be dealt with appropriately.

Parameters

s	The source string. Length 0 will return an empty array. `null` will throw an exception.

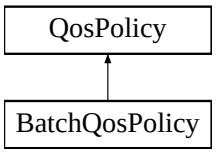
Returns

The decoded array of bytes. May be of length 0.

8.17 BatchQosPolicy Class Reference

Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

Inheritance diagram for BatchQosPolicy:



Public Attributes

- boolean **enable**

  Specifies whether or not batching is enabled.
- int **max_data_bytes**

  The maximum cumulative length of all serialized samples in a batch.
- int **max_samples**

  The maximum number of samples in a batch.
- final **Duration_t max_flush_delay**

  The maximum flush delay.
- final **Duration_t source_timestamp_resolution**

  Batch source timestamp resolution.
- boolean **thread_safe_write**

  Determines whether or not the write operation is thread safe.

8.17.1 Detailed Description

Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

This QoS policy configures the ability of the middleware to collect multiple user data samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

This QoS policy can be used to dramatically increase effective throughput for small data samples. Usually, throughput for small samples (size $<$ 2048 bytes) is limited by CPU capacity and not by network bandwidth. Batching many smaller samples to be sent in a single large packet will increase network utilization, and thus throughput, in terms of samples per second.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.17.2 Member Data Documentation

8.17.2.1 enable

```
boolean enable
```

Specifies whether or not batching is enabled.

[default] com.rti.dds.infrastructure.false

8.17.2.2 max_data_bytes

```
int max_data_bytes
```

The maximum cumulative length of all serialized samples in a batch.

A batch is flushed automatically when this maximum is reached.

max_data_bytes does not include the meta data associated with the batch samples. Each sample has at least 8 bytes of meta data containing information such as the timestamp and sequence number. The meta data can be as large as 52 bytes for keyed topics and 20 bytes for unkeyed topics.

Note: Batches must contain whole samples. If a new batch is started and its initial sample causes the serialized size to exceed max_data_bytes, RTI Connext will send the sample in a single batch.

[default] 1024

[range] [1,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.17.3 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_data_bytes** (p. 357) must be consistent with **com.↩ rti.dds.infrastructure.BatchQosPolicy.max_samples** (p. 357). For these two values to be consistent, they cannot be both **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

8.17.3.1 max_samples

```
int max_samples
```

The maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.17.4 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_samples** (p. 357) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.max_data_bytes** (p. 357). For these two values to be consistent, they cannot be both **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

8.17.4.1 max_flush_delay

```
final Duration_t max_flush_delay
```

The maximum flush delay.

A batch is flushed automatically after the delay specified by this parameter.

The delay is measured from the time the first sample in the batch is written by the application.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [0,**com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)]

8.17.5 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_flush_delay** (p. 358) must be consistent with **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.disable_asynchronous_batch** (p. 341) and **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 359). If the delay is different than **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.disable_asynchronous_batch** (p. 341) must be set to com.rti.dds.infrastructure.false and **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 359) must be set to com.rti.dds.infrastructure.true.

8.17.5.1 source_timestamp_resolution

```
final Duration_t source_timestamp_resolution
```

Batch source timestamp resolution.

The value of this field determines how the source timestamp is associated with the samples in a batch.

A sample written with timestamp 't' inherits the source timestamp 't2' associated with the previous sample unless ('t' - 't2') $>$ source_timestamp_resolution.

If source_timestamp_resolution is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), every sample in the batch will share the source timestamp associated with the first sample.

If source_timestamp_resolution is set to zero, every sample in the batch will contain its own source timestamp corresponding to the moment when the sample was written.

The performance of the batching process is better when source_timestamp_resolution is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [0,**com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)]

8.17.6 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 358) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 359). If **com.rti.dds.infrastructure.↩ BatchQosPolicy.thread_safe_write** (p. 359) is set to com.rti.dds.infrastructure.false, **com.rti.dds.infrastructure.↩ BatchQosPolicy.source_timestamp_resolution** (p. 358) must be set to **com.rti.dds.infrastructure.Duration_t.↩ DURATION_INFINITE** (p. 846).

8.17.6.1 thread_safe_write

```
boolean thread_safe_write
```

Determines whether or not the write operation is thread safe.

If this parameter is set to com.rti.dds.infrastructure.true, multiple threads can call write on the **com.rti.dds.↩ publication.DataWriter** (p. 553) concurrently.

[default] com.rti.dds.infrastructure.true

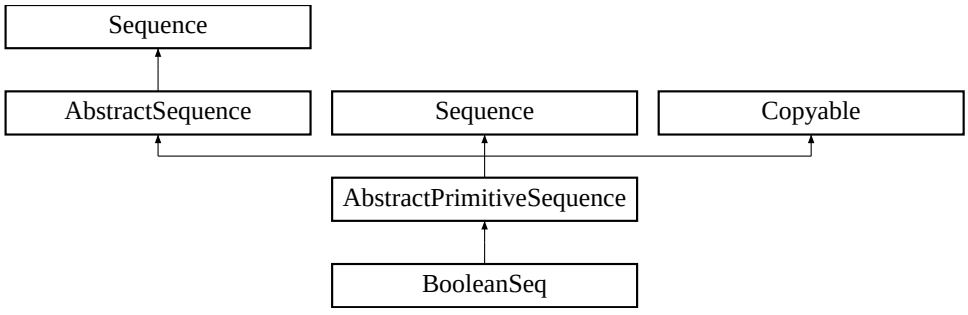
8.17.7 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 359) must be consistent with **com.↩ rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 358). If **com.rti.dds.infrastructure.↩ BatchQosPolicy.thread_safe_write** (p. 359) is set to com.rti.dds.infrastructure.false, **com.rti.dds.infrastructure.↩ BatchQosPolicy.source_timestamp_resolution** (p. 358) must be set to **com.rti.dds.infrastructure.Duration_t.↩ DURATION_INFINITE** (p. 846).

8.18 BooleanSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < boolean >

Inheritance diagram for BooleanSeq:



Public Member Functions

- **BooleanSeq** ()

  Constructs an empty sequence of booleans with an initial maximum of zero.
- **BooleanSeq** (int initialMaximum)

  Constructs an empty sequence of booleans with the given initial maximum.
- **BooleanSeq** (boolean[ ] booleans)

  Constructs a new sequence containing the given booleans.
- boolean **addAllBoolean** (boolean[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- boolean **addAllBoolean** (boolean[ ] elements)
- void **addBoolean** (boolean element)

  Append the element to the end of the sequence.
- void **addBoolean** (int index, boolean element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- boolean **getBoolean** (int index)

  Returns the boolean at the given index.
- boolean **setBoolean** (int index, boolean element)

  Set the new boolean at the given index and return the old boolean.
- void **setBoolean** (int dstIndex, boolean[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- boolean[ ] **toArrayBoolean** (boolean[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  *A wrapper for **getBoolean(int)** (p. 362) that returns a java.lang.Boolean.*
- Object **set** (int index, Object element)

  *A wrapper for **setBoolean()** (p. 363).*
- void **add** (int index, Object element)

  A wrapper for addBoolean(int, int).

8.18.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < boolean >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

boolean

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.18.2 Constructor & Destructor Documentation

8.18.2.1 BooleanSeq() [1/3]

```
BooleanSeq ( )
```

Constructs an empty sequence of booleans with an initial maximum of zero.

8.18.2.2 BooleanSeq() [2/3]

```
BooleanSeq (
            int initialMaximum )
```

Constructs an empty sequence of booleans with the given initial maximum.

8.18.2.3 BooleanSeq() [3/3]

```
BooleanSeq (
            boolean[] booleans )
```

Constructs a new sequence containing the given booleans.

Parameters

booleans	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **BooleanSeq.addAllBoolean()**.

8.18.3 Member Function Documentation

8.18.3.1 addAllBoolean() [1/2]

```
boolean addAllBoolean (
            boolean[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **BooleanSeq.addAllBoolean()**, and **BooleanSeq.BooleanSeq()**.

8.18.3.2 addAllBoolean() [2/2]

```
boolean addAllBoolean (
            boolean[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **BooleanSeq.addAllBoolean()**.

8.18.3.3 addBoolean() [1/2]

```
void addBoolean (
            boolean element )
```

Append the element to the end of the sequence.

Referenced by **BooleanSeq.add()**.

8.18.3.4 addBoolean() [2/2]

```
void addBoolean (
            int index,
            boolean element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.18.3.5 getBoolean()

```
boolean getBoolean (
            int index )
```

Returns the boolean at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **BooleanSeq.get()**.

8.18.3.6 setBoolean() [1/2]

```
boolean setBoolean (
            int index,
            boolean element )
```

Set the new boolean at the given index and return the old boolean.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **BooleanSeq.set()**.

8.18.3.7 setBoolean() [2/2]

```
void setBoolean (
            int dstIndex,
            boolean[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.18.3.8 toArrayBoolean()

```
boolean[] toArrayBoolean (
            boolean[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.18.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 365), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_boolean_seq()**, and **DynamicData.set_boolean_seq()**.

8.18.3.10 get()

```
Object get (
            int index )
```

A wrapper for **getBoolean(int)** (p. 362) that returns a java.lang.Boolean.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **BooleanSeq.getBoolean()**.

8.18.3.11 set()

```
Object set (
            int index,
            Object element )
```

A wrapper for **setBoolean()** (p. 363).

Exceptions

ClassCastException	if the element is not of type Boolean.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **BooleanSeq.setBoolean()**.

8.18.3.12 add()

```
void add (
            int index,
            Object element )
```

A wrapper for addBoolean(int, int).

Exceptions

ClassCastException	if the element is not of type Boolean.

See also

java.util.List::add(int, java.lang.Object)

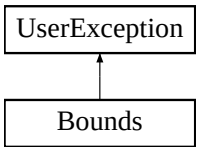
Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **BooleanSeq.addBoolean()**.

8.19 Bounds Class Reference

A user exception thrown when a parameter is not within the legal bounds.

Inheritance diagram for Bounds:



8.19.1 Detailed Description

A user exception thrown when a parameter is not within the legal bounds.

8.20 BuiltinQosProfiles Class Reference

The available built-in QoS libraries, profiles, and snippets.

Static Public Attributes

- static final String **BUILTIN_QOS_LIB**

  A library of non-experimental QoS profiles.
- static final String **PROFILE_BASELINE_ROOT**

  The root baseline QoS values from which all other Baseline.X.X.X profiles inherit.
- static final String **PROFILE_BASELINE**

  The most up-to-date QoS default values.
- static final String **PROFILE_BASELINE_5_0_0**

  The QoS default values for version 5.0.0.
- static final String **PROFILE_BASELINE_5_1_0**

  The QoS default values for version 5.1.0.
- static final String **PROFILE_BASELINE_5_2_0**

  The QoS default values for version 5.2.0.
- static final String **PROFILE_BASELINE_5_3_0**

  The QoS default values for version 5.3.0.
- static final String **PROFILE_BASELINE_6_0_0**

  The QoS default values for version 6.0.0.
- static final String **PROFILE_BASELINE_6_1_0**

  The QoS default values for version 6.1.0.
- static final String **PROFILE_BASELINE_7_0_0**

  The QoS default values for version 7.0.0.
- static final String **PROFILE_BASELINE_7_1_0**

  The QoS default values for version 7.1.0.
- static final String **PROFILE_GENERIC_COMMON**

  A common Participant base profile.
- static final String **PROFILE_GENERIC_MONITORING_COMMON**

  Enables RTI Monitoring Library.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY**

  Sets the values necessary to communicate with RTI Connext Micro.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.4 through at least 2.4.9.
- static final String **PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3**

  Sets the values necessary to communicate with RTI Connext Micro versions 2.4.3 and ealier.
- static final String **PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY**

  Sets the values necessary to interoperate with other DDS vendors.
- static final String **PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY**

  Sets the values necessary to interoperate with RTI Connext 5.1.0 using the UDPv6 and/or SHMEM transports.
- static final String **PROFILE_GENERIC_SECURITY**

  Loads the DDS Secure builtin plugins.
- static final String **BUILTIN_QOS_LIB_EXP**

  A library of experimental QoS profiles.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE**

  Enables strict reliability.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE**

  Enables keep-last reliability.
- static final String **PROFILE_GENERIC_BEST_EFFORT**

Enables best-effort reliability kind.

- static final String **PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT**

  A profile that can be used to achieve high throughput.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY**

  A profile that can be used to achieve low latency.
- static final String **PROFILE_GENERIC_PARTICIPANT_LARGE_DATA**

  A common Participant base profile to facilitate sending large data.
- static final String **PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING**

  Configures Participants for large data and monitoring.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with strict reliability.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA**

  Configures endpoints for sending large data with keep-last reliability.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures strictly reliable communication for large data with a fast flow controller.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures strictly reliable communication for large data with a medium flow controller.
- static final String **PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures strictly reliable communication for large data with a slow flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW**

  Configures keep-last reliable communication for large data with a fast flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW**

  Configures keep-last reliable communication for large data with a medium flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW**

  Configures keep-last reliable communication for large data with a slow flow controller.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL**

  Persists the samples of a DataWriter as long as the entity exists.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT**

  Persists samples using RTI Persistence Service.
- static final String **PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT**

  Persists samples in permanent storage, like a disk, using RTI Persistence Service.
- static final String **PROFILE_GENERIC_AUTO_TUNING**

  Enables the Turbo Mode batching and Auto Throttle experimental features.
- static final String **PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT**

  Uses a set of QoS which reduces the memory footprint of the application.
- static final String **PROFILE_GENERIC_MONITORING2**

  The default QoS profile that the DDS entities created by the RTI Monitoring Library 2.0 use.
- static final String **PROFILE_PATTERN_PERIODIC_DATA**

  Used for applications that expect periodic data.
- static final String **PROFILE_PATTERN_STREAMING**

  Used for applications that stream data.
- static final String **PROFILE_PATTERN_RELIABLE_STREAMING**

  Used for applications that stream data and require reliable communication.
- static final String **PROFILE_PATTERN_EVENT**

  Used for applications that handle events.
- static final String **PROFILE_PATTERN_ALARM_EVENT**

  Used for applications that handle alarm events.

- static final String **PROFILE_PATTERN_STATUS**

  Used for applications whose samples represent statuses.
- static final String **PROFILE_PATTERN_ALARM_STATUS**

  Used for applications in which samples represent alarm statuses.
- static final String **PROFILE_PATTERN_LAST_VALUE_CACHE**

  Used for applications that only need the last published value.
- static final String **BUILTIN_QOS_SNIPPET_LIB**

  A library of QoS Snippets.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON**

  QoS Snippet that configures the reliability protocol with a common configuration.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL**

  QoS Snippet that configures the reliability protocol for KEEP_ALL.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST**

  QoS Snippet that configures the reliability protocol for KEEP_LAST.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE**

  QoS Snippet that configures the reliability protocol for sending data at a high rate.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY**

  QoS Snippet that configures the reliability protocol for sending data at low latency.
- static final String **SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA**

  QoS Snippet that configures the reliability protocol for large data.
- static final String **SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC**

  Configures the DataWriter and DataReader caches to use dynamic memory allocation for Large Data samples.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_COMMON**

  QoS Snippet that optimizes discovery with a common configuration.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT**

  QoS Snippet that optimizes the Participant QoS to send less discovery information.
- static final String **SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST**

  QoS Snippet that optimizes the Endpoint Discovery to be faster.
- static final String **SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS**

  QoS Snippet that increases the Participant default buffer that shmem and udpv4 use.
- static final String **SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE**

  QoS Snippet that sets RELIABILITY QoS to RELIABLE.
- static final String **SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT**

  QoS Snippet that sets RELIABILITY QoS to BEST_EFFORT.
- static final String **SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1**

  QoS Snippet that sets HISTORY QoS to KEEP_LAST kind with depth 1.
- static final String **SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL**

  *QoS Snippet that sets HISTORY **QosPolicy** (*p. *1501) to KEEP_ALL kind.*
- static final String **SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS**

  *QoS Snippet that sets PUBLISH_MODE **QosPolicy** (*p. *1501) to ASYNCHRONOUS kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL**

  *QoS Snippet that sets DURABILITY **QosPolicy** (*p. *1501) to TRANSIENT_LOCAL kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT**

  *QoS Snippet that sets DURABILITY **QosPolicy** (*p. *1501) to TRANSIENT kind.*
- static final String **SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT**

  *QoS Snippet that sets DURABILITY **QosPolicy** (*p. *1501) to PERSISTENT kind.*
- static final String **SNIPPET_QOS_POLICY_BATCHING_ENABLE**

*QoS Snippet that sets BATCH **QosPolicy** (p. 1501) to true.*

- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS**

  QoS Snippet that configures and set a FlowController of 838 Mbps.
- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS**

  QoS Snippet that configures and sets a FlowController of 209 Mbps.
- static final String **SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS**

  QoS Snippet that configures and sets a FlowController of 52 Mbps.
- static final String **SNIPPET_FEATURE_AUTO_TUNING_ENABLE**

  QoS Snippet that enables auto_throttle and turbo_mode to true.
- static final String **SNIPPET_FEATURE_MONITORING_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library.
- static final String **SNIPPET_FEATURE_MONITORING2_ENABLE**

  QoS Snippet that enables the use of the RTI Monitoring Library 2.0.
- static final String **SNIPPET_FEATURE_SECURITY_ENABLE**

  QoS Snippet that enables security using the Builtin Security Plugins.
- static final String **SNIPPET_FEATURE_TOPIC_QUERY_ENABLE**

  QoS Snippet that enables Topic Query.
- static final String **SNIPPET_TRANSPORT_TCP_LAN_CLIENT** = "Transport.TCP.LAN.Client"

  QoS Snippet that configures a TCP LAN Client over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT**

  QoS Snippet that configures a symmetric WAN TCP Client over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER**

  QoS Snippet that an asymmetric WAN TCP Server over DDS.
- static final String **SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT**

  QoS Snippet that configures an asymmetric WAN TCP Client over DDS.
- static final String **SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION**

  QoS Snippet that configures RTI Connext and the UDP built-in transports (UDPv4, UDPv6, UDPv4_WAN) to avoid IP fragmentation.
- static final String **SNIPPET_TRANSPORT_UDP_WAN**

  QoS Snippet that enables the RTI Real-Time WAN Transport (UDPv4_WAN).
- static final String **SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3**

  QoS Snippet that configures sets LIVELINESS QoS to be compatible with RTI Connext Micro 2.4.3.
- static final String **SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with other DDS vendors.
- static final String **SNIPPET_5_1_0_TRANSPORT_ENABLE**

  QoS Snippet that configures RTI Connext to interoperate with RTI Connext 5.1.0 and below for UDPv6 and SHMEM transports.

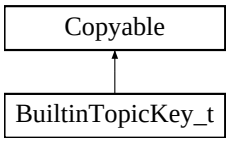
8.20.1 Detailed Description

The available built-in QoS libraries, profiles, and snippets.

8.21 BuiltinTopicKey_t Class Reference

The key type of the built-in topic types.

Inheritance diagram for BuiltinTopicKey_t:



Public Member Functions

- Object **copy_from** (Object src)

  Copy value of a data type from source.
- int[ ] **to_int_array** ()

  *Returns an array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity*** (p. *1029).*
- void **from_int_array** (int[ ] iValue)

  *Initializes this key from an array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity*** (p. *1029).*
- boolean **is_keyed_reader** ()

  Returns true if this key identifies a keyed DataReader.
- boolean **is_keyed_writer** ()

  Returns true if this key identifies a keyed DataWriter.
- boolean **is_unkeyed_reader** ()

  Returns true if this key identifies an unkeyed DataReader.
- boolean **is_unkeyed_writer** ()

  Returns true if this key identifies an unkeyed DataWriter.
- boolean **is_participant** ()

  Returns true if this key identifies a DomainParticipant.
- boolean **is_user_entity** ()

  Returns true if this key identifies an user Entity.
- boolean **is_builtin_entity** ()

  Returns true if this key identifies a built-in Entity.
- boolean **is_vendor_entity** ()

  Returns true if this key identifies a vendor Entity.
- boolean **is_vendor_builtin_entity** ()

  Returns true if this key identifies a built-in vendor Entity.
- int **get_entity_kind** ()

  Returns an integer identifying the entity kind for the key. The integer can take one of the following values:
- void **to_guid** (**GUID_t** dst)

  Converts this Key into a GUID.
- void **from_guid** (**GUID_t** src)

  Initializes this Key from the input GUID.

Public Attributes

- final int[ ] **value**

  An array of four integers that uniquely represents a remote ***com.rti.dds.infrastructure.Entity*** *(*p. *1029)**.*

Static Public Attributes

- static final int **KEYED_READER_ENTITY_KIND** = 0x07

  Constant representing a keyed DataReader.
- static final int **KEYED_WRITER_ENTITY_KIND** = 0x02

  Constant representing a keyed DataWriter.
- static final int **UNKEYED_READER_ENTITY_KIND** = 0x04

  Constant representing an unkeyed DataReader.
- static final int **UNKEYED_WRITER_ENTITY_KIND** = 0x03

  Constant representing an unkeyed DataWriter.
- static final int **PARTICIPANT_ENTITY_KIND** = 0x01

  Constant representing a DomainParticipant.

8.21.1 Detailed Description

The key type of the built-in topic types.

Each remote **com.rti.dds.infrastructure.Entity** (p. 1029) to be discovered can be uniquely identified by this key. This is the key of all the built-in topic data types.

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

builtin.TopicBuiltinTopicData (p. 1813)

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

8.21.2 Member Function Documentation

8.21.2.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

8.21.2.2 to_int_array()

```
int[] to_int_array ( )
```

Returns an array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity** (p. 1029).

References **BuiltinTopicKey_t.value**.

8.21.2.3 from_int_array()

```
void from_int_array (
             int[] iValue )
```

Initializes this key from an array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity** (p. 1029).

References **BuiltinTopicKey_t.value**.

8.21.2.4 is_keyed_reader()

```
boolean is_keyed_reader ( )
```

Returns true if this key identifies a keyed DataReader.

References **BuiltinTopicKey_t.KEYED_READER_ENTITY_KIND**, and **BuiltinTopicKey_t.value**.

8.21.2.5 is_keyed_writer()

```
boolean is_keyed_writer ( )
```

Returns true if this key identifies a keyed DataWriter.

References **BuiltinTopicKey_t.KEYED_WRITER_ENTITY_KIND**, and **BuiltinTopicKey_t.value**.

8.21.2.6 is_unkeyed_reader()

```
boolean is_unkeyed_reader ( )
```

Returns true if this key identifies an unkeyed DataReader.

References **BuiltinTopicKey_t.UNKEYED_READER_ENTITY_KIND**, and **BuiltinTopicKey_t.value**.

8.21.2.7 is_unkeyed_writer()

```
boolean is_unkeyed_writer ( )
```

Returns true if this key identifies an unkeyed DataWriter.

References **BuiltinTopicKey_t.UNKEYED_WRITER_ENTITY_KIND**, and **BuiltinTopicKey_t.value**.

8.21.2.8 is_participant()

```
boolean is_participant ( )
```

Returns true if this key identifies a DomainParticipant.

References **BuiltinTopicKey_t.value**.

8.21.2.9 is_user_entity()

```
boolean is_user_entity ( )
```

Returns true if this key identifies an user Entity.

References **BuiltinTopicKey_t.value**.

8.21.2.10 is_builtin_entity()

```
boolean is_builtin_entity ( )
```

Returns true if this key identifies a built-in Entity.

References **BuiltinTopicKey_t.value**.

8.21.2.11 is_vendor_entity()

```
boolean is_vendor_entity ( )
```

Returns true if this key identifies a vendor Entity.

References **BuiltinTopicKey_t.value**.

8.21.2.12 is_vendor_builtin_entity()

```
boolean is_vendor_builtin_entity ( )
```

Returns true if this key identifies a built-in vendor Entity.

References **BuiltinTopicKey_t.value**.

8.21.2.13 get_entity_kind()

```
int get_entity_kind ( )
```

Returns an integer identifying the entity kind for the key. The integer can take one of the following values:

- **builtin.BuiltinTopicKey_t.KEYED_READER_ENTITY_KIND** (p. 376)
- **builtin.BuiltinTopicKey_t.KEYED_WRITER_ENTITY_KIND** (p. 376)
- **builtin.BuiltinTopicKey_t.UNKEYED_READER_ENTITY_KIND** (p. 376)
- **builtin.BuiltinTopicKey_t.UNKEYED_WRITER_ENTITY_KIND** (p. 377)
- **builtin.BuiltinTopicKey_t.PARTICIPANT_ENTITY_KIND**<**P**> (p. 377)

References **BuiltinTopicKey_t.value**.

8.21.2.14 to_guid()

```
void to_guid (
            GUID_t dst )
```

Converts this Key into a GUID.

Parameters

dst	<<***out***>> *(*p. *156)* The destination GUID.

References **GUID_t.value**, and **BuiltinTopicKey_t.value**.

8.21.2.15 from_guid()

```
void from_guid (
            GUID_t src )
```

Initializes this Key from the input GUID.

Parameters

src	<<***in***>> *(*p. *156)* The GUID to be used to initialize this Key.

References **GUID_t.value**, and **BuiltinTopicKey_t.value**.

8.21.3 Member Data Documentation

8.21.3.1 KEYED_READER_ENTITY_KIND

```
final int KEYED_READER_ENTITY_KIND = 0x07  [static]
```

Constant representing a keyed DataReader.

The user can use this constant to inspect the returned value of the method **builtin.BuiltinTopicKey_t.get_entity_kind** (p. 375).

Referenced by **BuiltinTopicKey_t.is_keyed_reader()**.

8.21.3.2 KEYED_WRITER_ENTITY_KIND

```
final int KEYED_WRITER_ENTITY_KIND = 0x02  [static]
```

Constant representing a keyed DataWriter.

The user can use this constant to inspect the returned value of the method **builtin.BuiltinTopicKey_t.get_entity_kind** (p. 375).

Referenced by **BuiltinTopicKey_t.is_keyed_writer()**.

8.21.3.3 UNKEYED_READER_ENTITY_KIND

```
final int UNKEYED_READER_ENTITY_KIND = 0x04  [static]
```

Constant representing an unkeyed DataReader.

The user can use this constant to inspect the returned value of the method **builtin.BuiltinTopicKey_t.get_entity_kind** (p. 375).

Referenced by **BuiltinTopicKey_t.is_unkeyed_reader()**.

8.21.3.4 UNKEYED_WRITER_ENTITY_KIND

```
final int UNKEYED_WRITER_ENTITY_KIND = 0x03  [static]
```

Constant representing an unkeyed DataWriter.

The user can use this constant to inspect the returned value of the method **builtin.BuiltinTopicKey_t.get_entity_kind** (p. 375).

Referenced by **BuiltinTopicKey_t.is_unkeyed_writer()**.

8.21.3.5 PARTICIPANT_ENTITY_KIND

```
final int PARTICIPANT_ENTITY_KIND = 0x01  [static]
```

Constant representing a DomainParticipant.

The user can use this constant to inspect the returned value of the method **builtin.BuiltinTopicKey_t.get_entity_kind** (p. 375).

8.21.3.6 value

```
final int [] value
```

An array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity** (p. 1029).

Referenced by **BuiltinTopicKey_t.from_guid()**, **BuiltinTopicKey_t.from_int_array()**, **BuiltinTopicKey_t.get_↩ entity_kind()**, **BuiltinTopicKey_t.is_builtin_entity()**, **BuiltinTopicKey_t.is_keyed_reader()**, **BuiltinTopicKey_↩ t.is_keyed_writer()**, **BuiltinTopicKey_t.is_participant()**, **BuiltinTopicKey_t.is_unkeyed_reader()**, **BuiltinTopic↩ Key_t.is_unkeyed_writer()**, **BuiltinTopicKey_t.is_user_entity()**, **BuiltinTopicKey_t.is_vendor_builtin_entity()**, **BuiltinTopicKey_t.is_vendor_entity()**, **BuiltinTopicKey_t.to_guid()**, and **BuiltinTopicKey_t.to_int_array()**.

8.22 BuiltinTopicReaderResourceLimits_t Class Reference

Built-in topic reader's resource limits.

Inherits Struct.

Public Member Functions

- **BuiltinTopicReaderResourceLimits_t** ()

  Constructor with default initial and maximum values.
- **BuiltinTopicReaderResourceLimits_t** (int **initial_samples**, int **max_samples**, int **initial_infos**, int **max_infos**, int initial_outstanding_reads, int max_outstanding_reads, int max_samples_per_read, boolean **disable_fragmentation_support**, int **max_fragmented_samples**, int **initial_fragmented_samples**, int **max_fragmented_samples_per_remote_writer**, int **max_fragments_per_sample**, boolean **dynamically_allocate_fragmented_samples**)

  Constructor with the given initial and maximum values.

Public Attributes

- int **initial_samples**

  Initial number of samples.
- int **max_samples**

  Maximum number of samples.
- int **initial_infos**

  Initial number of sample infos.
- int **max_infos**

  Maximum number of sample infos.
- boolean **disable_fragmentation_support**

  *Determines whether the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) can receive fragmented samples.*
- int **max_fragmented_samples**

  *The maximum number of samples for which the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments at a given point in time.*
- int **initial_fragmented_samples**

  *The initial number of samples for which a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.*
- int **max_fragmented_samples_per_remote_writer**

  *The maximum number of samples per remote writer for which a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.*
- int **max_fragments_per_sample**

  Maximum number of fragments for a single sample.
- boolean **dynamically_allocate_fragmented_samples**

  *Determines whether the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) pre-allocates storage for storing fragmented samples.*

8.22.1 Detailed Description

Built-in topic reader's resource limits.

Defines the resources that can be used for a built-in-topic data reader.

A built-in topic data reader subscribes reliably to built-in topics containing declarations of new entities or updates to existing entities in the domain. Keys are used to differentiate among entities of the same type. RTI Connext assigns a unique key to each entity in a domain.

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

QoS:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 646)

8.22.2 Constructor & Destructor Documentation

8.22.2.1 BuiltinTopicReaderResourceLimits_t() [1/2]

```
BuiltinTopicReaderResourceLimits_t ( )
```

Constructor with default initial and maximum values.

8.22.2.2 BuiltinTopicReaderResourceLimits_t() [2/2]

```
BuiltinTopicReaderResourceLimits_t (
            int initial_samples,
            int max_samples,
            int initial_infos,
            int max_infos,
            int initial_outstanding_reads,
            int max_outstanding_reads,
            int max_samples_per_read,
            boolean disable_fragmentation_support,
            int max_fragmented_samples,
            int initial_fragmented_samples,
            int max_fragmented_samples_per_remote_writer,
```

```
int max_fragments_per_sample,
boolean dynamically_allocate_fragmented_samples )
```

Constructor with the given initial and maximum values.

References **BuiltinTopicReaderResourceLimits_t.disable_fragmentation_support**, **BuiltinTopicReader↩ ResourceLimits_t.dynamically_allocate_fragmented_samples**, **BuiltinTopicReaderResourceLimits_t.initial_↩ fragmented_samples**, **BuiltinTopicReaderResourceLimits_t.initial_infos**, **BuiltinTopicReaderResourceLimits↩ _t.initial_samples**, **BuiltinTopicReaderResourceLimits_t.max_fragmented_samples**, **BuiltinTopicReader↩ ResourceLimits_t.max_fragmented_samples_per_remote_writer**, **BuiltinTopicReaderResourceLimits_t.max↩ _fragments_per_sample**, **BuiltinTopicReaderResourceLimits_t.max_infos**, and **BuiltinTopicReaderResource↩ Limits_t.max_samples**.

8.22.3 Member Data Documentation

8.22.3.1 initial_samples

```
int initial_samples
```

Initial number of samples.

This should be a value between 1 and initial number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently.

[default] 64

[range] [1, 1 million], <= max_samples

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.2 max_samples

```
int max_samples
```

Maximum number of samples.

This should be a value between 1 and max number of instance of the built-in-topic reader, depending on how many instances are sending data concurrently. Also, it should not be less than initial_samples.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_samples

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.3 initial_infos

```
int initial_infos
```

Initial number of sample infos.

The initial number of info units that a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) can have. Info units are used to store **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] 64

[range] [1, 1 million] <= max_infos

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.4 max_infos

```
int max_infos
```

Maximum number of sample infos.

The maximum number of info units that a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) can use to store **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_infos

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.5 disable_fragmentation_support

```
boolean disable_fragmentation_support
```

Determines whether the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] com.rti.dds.infrastructure.false

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.6 max_fragmented_samples

```
int max_fragmented_samples
```

The maximum number of samples for which the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments at a given point in time.

At any given time, a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments for up to `max_↩ fragmented_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different built-in topic **com.rti.dds.publication.DataWriter** (p. 553) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t.max_samples** (p. 380).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t.disable_fragmentation_support** (p. 381) is com.rti.dds.infrastructure.false.

[default] 1024

[range] [1, 1 million]

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.7 initial_fragmented_samples

```
int initial_fragmented_samples
```

The initial number of samples for which a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.

Only applies if **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t.disable_fragmentation_support** (p. 381) is com.rti.dds.infrastructure.false.

[default] 4

[range] [1,1024], <= max_fragmented_samples

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.8 max_fragmented_samples_per_remote_writer

```
int max_fragmented_samples_per_remote_writer
```

The maximum number of samples per remote writer for which a built-in topic **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t.disable_fragmentation_support** (p. 381) is com.rti.dds.infrastructure.false.

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.9 max_fragments_per_sample

```
int max_fragments_per_sample
```

Maximum number of fragments for a single sample.

Only applies if **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t.disable_fragmentation_support** (p. 381) is com.rti.dds.infrastructure.false.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.22.3.10 dynamically_allocate_fragmented_samples

```
boolean dynamically_allocate_fragmented_samples
```

Determines whether the built-in topic **com.rti.dds.subscription.DataReader** (p. 450) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to com.rti.dds.infrastructure.false, the middleware will allocate memory upfront for storing fragments for up to **com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.initial_fragmented_samples** (p. 535) samples. This memory may grow up to **com.rti.dds.↩ infrastructure.DataReaderResourceLimitsQosPolicy.max_fragmented_samples** (p. 535) if needed.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

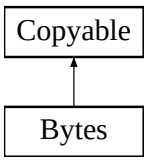
[default] com.rti.dds.infrastructure.true

Referenced by **BuiltinTopicReaderResourceLimits_t.BuiltinTopicReaderResourceLimits_t()**.

8.23 Bytes Class Reference

Built-in type consisting of a variable-length array of opaque bytes.

Inheritance diagram for Bytes:



Public Member Functions

- **Bytes** ()

  Default Constructor.
- **Bytes** (**Bytes** src)

  Copy constructor.
- **Bytes** (int theLength)

  Constructor that specifies the size of the allocated bytes array.
- **Bytes** (byte[ ] src)
- Object **copy_from** (Object src)

  Copy src into this object.

Public Attributes

- int **length**

  Number of bytes to serialize.
- int **offset**

  Offset from which to start serializing bytes.
- byte[ ] **value**

  com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes array value.

8.23.1 Detailed Description

Built-in type consisting of a variable-length array of opaque bytes.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 Bytes() [1/4]

```
Bytes ( )
```

Default Constructor.

The default constructor initializes the newly created object with null value, zero length, and zero offset.

Referenced by **Bytes.copy_from()**.

8.23.2.2 Bytes() [2/4]

```
Bytes (
            Bytes src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Object to copy from.

Exceptions

NullPointerException	if src is null.

References **Bytes.copy_from()**.

8.23.2.3 Bytes() [3/4]

```
Bytes (
            int theLength )
```

Constructor that specifies the size of the allocated bytes array.

After this method is called, length and offset are set to zero.

Parameters

theLength	<<***in***>> *(*p. *156)* Size of the allocated bytes array bytes array

Exceptions

IllegalArgumentException	if size is negative

References **Bytes.length**, **Bytes.offset**, and **Bytes.value**.

8.23.2.4 Bytes() [4/4]

```
Bytes (
            byte[] src )
```

Create a new **Bytes** (p. 384) object to wrap the given array.

The `value` field will point to the given array. The `offset` will be set to 0. The `length` will be set to the length of the array unless the array is null, in which case the `length` will be set to 0 also.

References **Bytes.length**, **Bytes.offset**, and **Bytes.value**.

8.23.3 Member Function Documentation

8.23.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy src into this object.

This method performs a deep copy of `src` and it allocates memory for the value if required.

Parameters

src	<<***in***>> *(p. 156)* Object to copy from.

Returns

this if success. Otherwise, null.

Exceptions

NullPointerException	if `src` is null.

Implements **Copyable** (p. 445).

References **Bytes.Bytes()**, **Bytes.length**, **Bytes.offset**, and **Bytes.value**.

Referenced by **Bytes.Bytes()**.

8.23.4 Member Data Documentation

8.23.4.1 length

```
int length
```

Number of bytes to serialize.

Referenced by **Bytes.Bytes()**, **Bytes.copy_from()**, **BytesDataWriter.write()**, and **BytesDataWriter.write_w_↩ timestamp()**.

8.23.4.2 offset

```
int offset
```

Offset from which to start serializing bytes.

The first position of the bytes array has offset 0.

Referenced by **Bytes.Bytes()**, **Bytes.copy_from()**, **BytesDataWriter.write()**, and **BytesDataWriter.write_w_↩ timestamp()**.

8.23.4.3 value

```
byte [] value
```

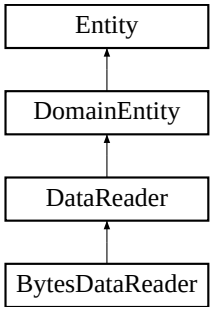
com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes array value.

Referenced by **Bytes.Bytes()**, **Bytes.copy_from()**, **BytesDataWriter.write()**, and **BytesDataWriter.write_w_↩ timestamp()**.

8.24 BytesDataReader Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.

Inheritance diagram for BytesDataReader:



Public Member Functions

- void **read** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view↩ _states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view↩ _states, int instance_states)

  *Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_w_condition** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩ **Condition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_w_condition** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩ **Condition** condition)

  Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.take operation.
- void **read_next_sample** (**Bytes** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_next_sample** (**Bytes** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **return_loan** (**BytesSeq** received_data, **SampleInfoSeq** info_seq)

  *Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.**↩ **subscription.DataReader** (p. 450).*

Additional Inherited Members

8.24.1 Detailed Description

<<***interface***>> *(p. 156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

com.rti.dds.subscription.DataReader (p. 450)

8.24.2 Member Function Documentation

8.24.2.1 read()

```
void read (
             BytesSeq received_data,
             SampleInfoSeq info_seq,
            int max_samples,
            int sample_states,
            int view_states,
            int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_untyped()**.

8.24.2.2 take()

```
void take (
             BytesSeq received_data,
             SampleInfoSeq info_seq,
            int max_samples,
            int sample_states,
            int view_states,
            int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_untyped()**.

8.24.2.3 read_w_condition()

```
void read_w_condition (
              BytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076)

References **DataReader.read_w_condition_untyped()**.

8.24.2.4 take_w_condition()

```
void take_w_condition (
              BytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataReader.take operation.

See also

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

References **DataReader.take_w_condition_untyped()**.

8.24.2.5 read_next_sample()

```
void read_next_sample (
              Bytes received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_sample (p. 1078)

References **DataReader.read_next_sample_untyped()**.

8.24.2.6 take_next_sample()

```
void take_next_sample (
            Bytes received_data,
            SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_next_sample (p. 1079)

References **DataReader.take_next_sample_untyped()**.

8.24.2.7 return_loan()

```
void return_loan (
            BytesSeq received_data,
            SampleInfoSeq info_seq )
```

Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.↩ subscription.DataReader** (p. 450).

See also

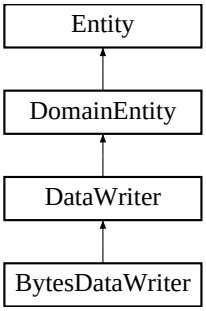
com.rti.ndds.example.FooDataReader.return_loan (p. 1094)

References **DataReader.return_loan_untyped()**.

8.25 BytesDataWriter Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.

Inheritance diagram for BytesDataWriter:



Public Member Functions

- void **write** (**Bytes** instance_data, **InstanceHandle_t** handle)

  Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.
- void **write** (byte[ ] octets, int offset, int length, **InstanceHandle_t** handle)

  <<***extension***>> (p. *155) Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.*
- void **write** (**ByteSeq** octets, **InstanceHandle_t** handle)

  <<***extension***>> (p. *155) Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.*
- void **write_w_timestamp** (**Bytes** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataWriter.write except that it also provides the value for the `source_timestamp`.
- void **write_w_timestamp** (byte[ ] octets, int offset, int length, **InstanceHandle_t** handle, **Time_t** source_↩ timestamp)

  <<***extension***>> (p. *155) Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesData↩ Writer.BytesDataWriter.write except that it also provides the value for the* `source_timestamp`.
- void **write_w_timestamp** (**ByteSeq** octets, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  <<***extension***>> (p. *155) Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesData↩ Writer.BytesDataWriter.write except that it also provides the value for the* `source_timestamp`.

8.25.1 Detailed Description

<<***interface***>> *(*p. *156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes >.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.publication.DataWriter (p. 553)

8.25.2 Member Function Documentation

8.25.2.1 write() [1/3]

```
void write (
              Bytes instance_data,
              InstanceHandle_t handle )
```

Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **DataWriter.write_untyped()**.

8.25.2.2 write() [2/3]

```
void write (
            byte[] octets,
            int offset,
            int length,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.

Parameters

octets	<<***in***>> *(p. 156)* Array of bytes to be published.
offset	<<***in***>> *(p. 156)* Offset from which to start publishing.
length	<<***in***>> *(p. 156)* Number of bytes to be published.
handle	<<***in***>> *(p. 156)* The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) should be used always.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **Bytes.length**, **Bytes.offset**, **Bytes.value**, and **DataWriter.write_untyped()**.

8.25.2.3 write() [3/3]

```
void write (
             ByteSeq octets,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data instance.

Parameters

octets	<<***in***>> *(p. 156)* Sequence of bytes to be published.
handle	<<***in***>> *(p. 156)* The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) should be used always.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **Bytes.length**, **Bytes.offset**, **AbstractPrimitiveSequence.size()**, **Bytes.value**, and **DataWriter.write_untyped()**.

8.25.2.4 write_w_timestamp() [1/3]

```
void write_w_timestamp (
              Bytes instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesDataWriter.write except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **DataWriter.write_w_timestamp_untyped()**.

8.25.2.5 write_w_timestamp() [2/3]

```
void write_w_timestamp (
             byte[] octets,
             int offset,
             int length,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

<<***extension***>> *(p. 155)* Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesData↩ Writer.BytesDataWriter.write except that it also provides the value for the `source_timestamp`.

Parameters

octets	<<***in***>> *(p. 156)* Array of bytes to be published.
offset	<<***in***>> *(p. 156)* Offset from which to start publishing.
length	<<***in***>> *(p. 156)* Number of bytes to be published.
handle	<<***in***>> *(p. 156)* The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) should be used always.
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109). Cannot be NULL.

See also

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **Bytes.length**, **Bytes.offset**, **Bytes.value**, and **DataWriter.write_w_timestamp_untyped()**.

8.25.2.6 write_w_timestamp() [3/3]

```
void write_w_timestamp (
            ByteSeq octets,
            InstanceHandle_t handle,
            Time_t source_timestamp )
```

<<***extension***>> *(*p. *155)* Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesData↩ Writer.BytesDataWriter.write except that it also provides the value for the `source_timestamp`.

Parameters

octets	<<***in***>> *(*p. *156)* Sequence of bytes to be published.
handle	<<***in***>> *(*p. *156)* The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) should be used always.
source_timestamp	<<***in***>> *(*p. *156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109). Cannot be NULL.

See also

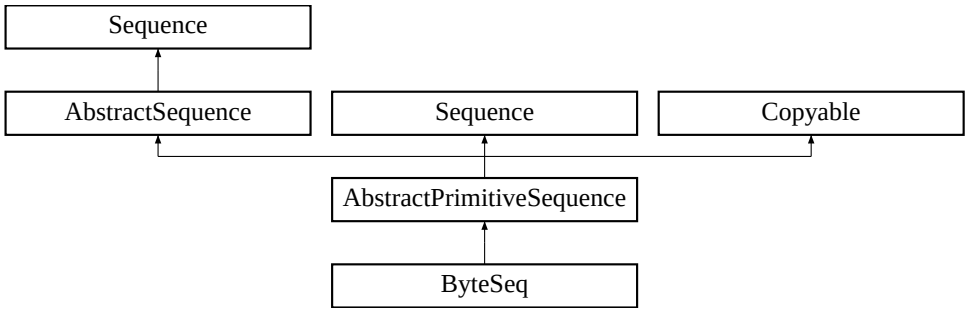
com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **Bytes.length**, **Bytes.offset**, **AbstractPrimitiveSequence.size()**, **Bytes.value**, and **DataWriter.write↩ _w_timestamp_untyped()**.

8.26 ByteSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < byte >

Inheritance diagram for ByteSeq:



Public Member Functions

- **ByteSeq** ()

  Constructs an empty sequence of bytes with an initial maximum of zero.
- **ByteSeq** (int initialMaximum)

  Constructs an empty sequence of bytes with the given initial maximum.
- **ByteSeq** (byte[ ] bytes)

  Construct a new sequence containing the given bytes.
- boolean **addAllByte** (byte[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- boolean **addAllByte** (byte[ ] elements)
- void **addByte** (byte element)

  Append the element to the end of the sequence.
- void **addByte** (int index, byte element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- byte **getByte** (int index)

  Returns the byte at the given index.
- byte **setByte** (int index, byte element)

  Set the new byte at the given index and return the old byte.
- void **setByte** (int dstIndex, byte[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- byte[ ] **toArrayByte** (byte[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  *A wrapper for **getByte(int)** (p. 398) that returns a java.lang.Byte.*
- Object **set** (int index, Object element)

  *A wrapper for **setByte()** (p. 399).*
- void **add** (int index, Object element)

  A wrapper for addByte(int, int).

8.26.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < byte >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

byte

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.26.2 Constructor & Destructor Documentation

8.26.2.1 ByteSeq() [1/3]

```
ByteSeq ( )
```

Constructs an empty sequence of bytes with an initial maximum of zero.

8.26.2.2 ByteSeq() [2/3]

```
ByteSeq (
            int initialMaximum )
```

Constructs an empty sequence of bytes with the given initial maximum.

8.26.2.3 ByteSeq() [3/3]

```
ByteSeq (
            byte[] bytes )
```

Construct a new sequence containing the given bytes.

Parameters

bytes	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **ByteSeq.addAllByte()**.

8.26.3 Member Function Documentation

8.26.3.1 addAllByte() [1/2]

```
boolean addAllByte (
            byte[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **ByteSeq.addAllByte()**, and **ByteSeq.ByteSeq()**.

8.26.3.2 addAllByte() [2/2]

```
boolean addAllByte (
            byte[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **ByteSeq.addAllByte()**.

8.26.3.3 addByte() [1/2]

```
void addByte (
            byte element )
```

Append the element to the end of the sequence.

Referenced by **ByteSeq.add()**.

8.26.3.4 addByte() [2/2]

```
void addByte (
            int index,
            byte element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.26.3.5 getByte()

```
byte getByte (
             int index )
```

Returns the byte at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **ByteSeq.get()**, and **DynamicData.get_uint8_seq()**.

8.26.3.6 setByte() [1/2]

```
byte setByte (
             int index,
             byte element )
```

Set the new byte at the given index and return the old byte.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **ByteSeq.set()**.

8.26.3.7 setByte() [2/2]

```
void setByte (
             int dstIndex,
             byte[] elements,
             int srcIndex,
             int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.26.3.8 toArrayByte()

```
byte[] toArrayByte (
            byte[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.26.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 401), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_byte_seq()**, **DynamicData.get_uint8_seq()**, **DynamicData.set_byte_seq()**, and **DynamicData.to_cdr_buffer()**.

8.26.3.10 get()

```
Object get (
            int index )
```

A wrapper for **getByte(int)** (p. 398) that returns a java.lang.Byte.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **ByteSeq.getByte()**.

8.26.3.11 set()

```
Object set (
            int index,
            Object element )
```

A wrapper for **setByte()** (p. 399).

Exceptions

ClassCastException	if the element is not of type Byte.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **ByteSeq.setByte()**.

Referenced by **DynamicData.set_uint8_seq()**.

8.26.3.12 add()

```
void add (
            int index,
            Object element )
```

A wrapper for addByte(int, int).

Exceptions

ClassCastException	if the element is not of type Byte.

See also

java.util.List::add(int, java.lang.Object)

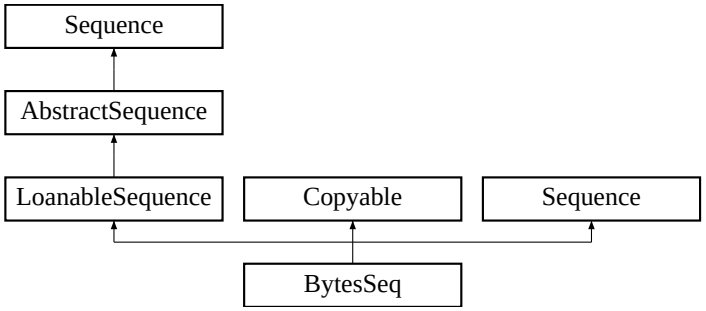
Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **ByteSeq.addByte()**.

8.27 BytesSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.Bytes > .

Inheritance diagram for BytesSeq:



Public Member Functions

- **BytesSeq** ()

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects with an initial maximum of zero.
- **BytesSeq** (int initialMaximum)

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects with the given initial maximum.
- **BytesSeq** (Collection elements)

  Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects.
- **Bytes get** (int index)

  Returns the element at the specified position in this sequence.
- Object **copy_from** (Object src)

8.27.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.Bytes > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes

8.27.2 Constructor & Destructor Documentation

8.27.2.1 BytesSeq() [1/3]

```
BytesSeq ( )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects with an initial maximum of zero.

8.27.2.2 BytesSeq() [2/3]

```
BytesSeq (
            int initialMaximum )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects with the given initial maximum.

8.27.2.3 BytesSeq() [3/3]

```
BytesSeq (
            Collection elements )
```

Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes objects.

Parameters

elements	the initial contents of this sequence.

Exceptions

NullPointerException	if the input collection is null

8.27.3 Member Function Documentation

8.27.3.1 get()

```
Bytes get (
            int index )
```

Returns the element at the specified position in this sequence.

See also

java.util.List::get(int)

Reimplemented from **LoanableSequence** (p. 1252).

8.27.3.2 copy_from()

```
Object copy_from (
            Object src )
```

Copy data into `this` object from another. The result of this method is that both `this` and `src` will be the same size and contain the same data.

Parameters

src	The Object which contains the data to be copied

Returns

`this`

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not a `Sequence` OR if one of the objects contained in the `Sequence` is not of the expected type.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

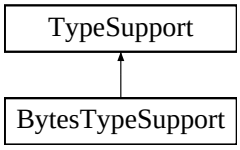
References **AbstractSequence.add()**, **BytesSeq.copy_from()**, **LoanableSequence.getMaximum()**, **Loanable↩ Sequence.setMaximum()**, and **LoanableSequence.size()**.

Referenced by **BytesSeq.copy_from()**.

8.28 BytesTypeSupport Class Reference

<<***interface***>> *(*p. *156)* com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type support.

Inheritance diagram for BytesTypeSupport:



Public Member Functions

- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **Bytes** src)

  <<***extension***>> *(*p. *155) Serializes the input sample into a CDR buffer of octets.*
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **Bytes** src, short representation)

  <<***extension***>> (p. *155) Serializes the input sample into a buffer of octets.*
- String **data_to_string** (**Bytes** src, **PrintFormatProperty** property)

  <<***extension***>> *(*p. *155) Get the string representation of an input sample.*
- String **data_to_string** (**Bytes** src)

  <<***extension***>> (p. *155) Get the string representation of an input sample.*
- void **deserialize_from_cdr_buffer** (**Bytes** dst, byte[ ] buffer, long length)

  <<***extension***>> (p. *155) Deserializes a sample from a buffer of octets.*

Static Public Member Functions

- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data type.
- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.
- static String **get_type_name** ()

  Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type.

8.28.1 Detailed Description

<<***interface***>> *(*p. *156)* com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type support.

8.28.2 Member Function Documentation

8.28.2.1 register_type()

```
static void register_type (
               DomainParticipant participant,
              String type_name )  [static]
```

Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data type.

By default, The com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes built-in type is automatically registered when a DomainParticipant is created using the type_name returned by com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesTypeSupport.get_type_name. Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesTypeSupport with a **com.rti.dds.domain.DomainParticipant** (p. 670) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 670) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to register the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes with. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1807). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706).) The name may not be null or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.28.2.2 unregister_type()

```
static void unregister_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1807) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **com.rti.dds.topic.Topic** (p. 1807) is associated with the type, the operation will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	*<<**in**>> (p. 156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to unregister the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes from. Cannot be null.
type_name	*<<**in**>> (p. 156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) or **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595)

MT Safety:

SAFE.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesTypeSupport.register_type

8.28.2.3 get_type_name()

```
static String get_type_name ( )  [static]
```

Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type.

Can be used for calling com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesTypeSupport.register_type or creating **com.rti.dds.topic.Topic** (p. 1807).

Returns

default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes type.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.BytesTypeSupport.register_type

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.28.2.4 serialize_to_cdr_buffer() [1/2]

```
long serialize_to_cdr_buffer (
             byte[] buffer,
             long length,
              Bytes src )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a CDR buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

Referenced by **BytesTypeSupport.data_to_string()**.

8.28.2.5 serialize_to_cdr_buffer() [2/2]

```
long serialize_to_cdr_buffer (
            byte[] buffer,
            long length,
             Bytes src,
            short representation )
```

<<***extension***>> *(*p. *155)* Serializes the input sample into a buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

8.28.2.6 data_to_string() [1/2]

```
String data_to_string (
             Bytes src,
             PrintFormatProperty property )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **DynamicData.from_cdr_buffer()**, **DynamicData.PROPERTY_DEFAULT**, **BytesTypeSupport.↩ serialize_to_cdr_buffer()**, and **DynamicData.to_string()**.

Referenced by **BytesTypeSupport.data_to_string()**.

8.28.2.7 data_to_string() [2/2]

```
String data_to_string (
             Bytes src )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

Use the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387) to create the output string.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **BytesTypeSupport.data_to_string()**.

8.28.2.8 deserialize_from_cdr_buffer()

```
void deserialize_from_cdr_buffer (
              Bytes dst,
             byte[] buffer,
             long length )
```

<<***extension***>> *(*p. *155)* Deserializes a sample from a buffer of octets.

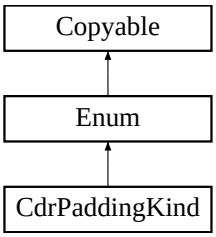
See also

com.rti.ndds.example.FooTypeSupport.deserialize_from_cdr_buffer (p. 1123)

8.29 CdrPaddingKind Class Reference

The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.

Inheritance diagram for CdrPaddingKind:



Static Public Attributes

- static final **CdrPaddingKind ZERO_CDR_PADDING**

  Padding bytes will be set to zeros during CDR serialization.
- static final **CdrPaddingKind NOT_SET_CDR_PADDING**

  Padding bytes will not be set to any value during CDR serialization.
- static final **CdrPaddingKind AUTO_CDR_PADDING**

  *When set on a **com.rti.dds.domain.DomainParticipant** (*p. *670) the default behavior is com.rti.dds.infrastructure.Cdr↩ PaddingKind.CdrPaddingKind.NOT_SET_CDR_PADDING. When set on a **com.rti.dds.publication.DataWriter** (*p. *553) or **com.rti.dds.subscription.DataReader** (*p. *450) the behavior is to inherit the value from the **com.rti.dds.domain.↩ DomainParticipant** (*p. *670).*

Additional Inherited Members

8.29.1 Detailed Description

The CDR padding kind determines whether or not the padding bytes will be set to zero during CDR serialization.

8.29.2 Member Data Documentation

8.29.2.1 ZERO_CDR_PADDING

```
final CdrPaddingKind ZERO_CDR_PADDING [static]
```

Initial value:

```
=
        new CdrPaddingKind("ZERO_CDR_PADDING", 0)
```

Padding bytes will be set to zeros during CDR serialization.

8.29.2.2 NOT_SET_CDR_PADDING

```
final CdrPaddingKind NOT_SET_CDR_PADDING [static]
```

Initial value:

```
=
        new CdrPaddingKind("NOT_SET_CDR_PADDING", 1)
```

Padding bytes will not be set to any value during CDR serialization.

8.29.2.3 AUTO_CDR_PADDING

```
final CdrPaddingKind AUTO_CDR_PADDING [static]
```

Initial value:

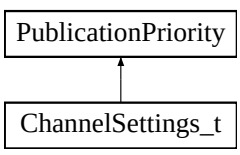
```
=
        new CdrPaddingKind("AUTO_CDR_PADDING", 2)
```

When set on a **com.rti.dds.domain.DomainParticipant** (p. 670) the default behavior is com.rti.dds.infrastructure.↩ CdrPaddingKind.CdrPaddingKind.NOT_SET_CDR_PADDING. When set on a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) the behavior is to inherit the value from the **com.rti.dds.**↩ **domain.DomainParticipant** (p. 670).

8.30 ChannelSettings_t Class Reference

Type used to configure the properties of a channel.

Inheritance diagram for ChannelSettings_t:



Public Member Functions

- **ChannelSettings_t** ()

  Constructor.
- **ChannelSettings_t** (**ChannelSettings_t** src)

  Constructor.
- **ChannelSettings_t** (**TransportMulticastSettingsSeq multicast_settings**, String **filter_expression**, int **priority**)

  Constructor.

Public Attributes

- **TransportMulticastSettingsSeq multicast_settings**

  *A sequence of **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) used to configure the multicast addresses associated with a channel.*
- String **filter_expression**

  A logical expression used to determine the data that will be published in the channel.
- int **priority** = **UNDEFINED**

  Publication priority.

Additional Inherited Members

8.30.1 Detailed Description

Type used to configure the properties of a channel.

QoS:

com.rti.dds.infrastructure.MultiChannelQosPolicy (p. 1318)

8.30.2 Constructor & Destructor Documentation

8.30.2.1 ChannelSettings_t() [1/3]

```
ChannelSettings_t ( )
```

Constructor.

8.30.2.2 ChannelSettings_t() [2/3]

```
ChannelSettings_t (
            ChannelSettings_t src )
```

Constructor.

Parameters

src	<<***in***>> *(*p. *156)* Settings used to initialized the new settings.

References **ChannelSettings_t.filter_expression**, **ChannelSettings_t.multicast_settings**, and **ChannelSettings_t.priority**.

8.30.2.3 ChannelSettings_t() [3/3]

```
ChannelSettings_t (
            TransportMulticastSettingsSeq multicast_settings,
            String filter_expression,
            int priority )
```

Constructor.

Parameters

multicast_settings	<<***in***>> *(*p. *156)* Multicast settings.
filter_expression	<<***in***>> *(*p. *156)* Filter expression.
priority	<<***in***>> *(*p. *156)* Priority.

References **ChannelSettings_t.filter_expression**, **ChannelSettings_t.multicast_settings**, and **ChannelSettings_t.priority**.

8.30.3 Member Data Documentation

8.30.3.1 multicast_settings

```
TransportMulticastSettingsSeq multicast_settings
```

Initial value:

```
=
        new TransportMulticastSettingsSeq()
```

A sequence of **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) used to configure the multicast addresses associated with a channel.

The sequence cannot be empty.

The maximum number of multicast locators in a channel is limited to 16 (a locator is defined by a transport alias, a multicast address and a port). Note that this is a hard limit that cannot be increased. However, this limit can

be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **com.↩ rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipantQos** (p. 795).

[default] Empty sequence (invalid value)

Referenced by **ChannelSettings_t.ChannelSettings_t()**.

8.30.3.2 filter_expression

```
String filter_expression
```

A logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

A NULL value is not allowed.

The syntax of the expression will depend on the value of **com.rti.dds.infrastructure.MultiChannelQosPolicy.filter↩ _name** (p. 1319)

The filter expression length (including NULL-terminated character) cannot be greater than **com.rti.dds.↩ infrastructure.DomainParticipantResourceLimitsQosPolicy.channel_filter_expression_max_length** (p. 819).

See also

Queries and Filters Syntax (p. 104)

[default] NULL (invalid value)

Referenced by **ChannelSettings_t.ChannelSettings_t()**.

8.30.3.3 priority

```
int priority = UNDEFINED
```

Publication priority.

A positive integer value designating the relative priority of the **com.rti.dds.publication.DataWriter** (p. 553), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (com.rti.dds.infrastructure.PublishModeQosPolicy↩ Kind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS) with **com.rti.dds.publication.Flow↩ ControllerProperty_t.scheduling_policy** (p. 1059) set to com.rti.dds.publication.FlowControllerSchedulingPolicy.↩ FlowControllerSchedulingPolicy.HPF_FLOW_CONTROLLER_SCHED_POLICY.

Larger numbers have higher priority.

For multi-channel DataWriters, if the publication priority of any channel is set to any value other than com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will take precedence over that of the DataWriter.

For multi-channel DataWriters, if the publication priority of any channel is com.rti.dds.infrastructure.PUBLICATION_↩ PRIORITY_UNDEFINED, then the channel will inherit the publication priority of the DataWriter.

If the publicaton priority of the DataWriter, and of any channel of a multi-channel DataWriter, are com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the priority of the DataWriter or DataWriter channel will be assigned the lowest priority value.

If the publication priority of the DataWriter is com.rti.dds.infrastructure.PUBLICATION_PRIORITY_AUTOMATIC, then the DataWriter will be assigned the priority of the largest publication priority of all samples in the DataWriter.

The publication priority of each sample can be set in the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of the **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110) function.

For dispose and unregister samples, use the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of **com.rti.ndds.↩ example.FooDataWriter.dispose_w_params** (p. 1114) and **com.rti.ndds.example.FooDataWriter.unregister_↩ instance_w_params** (p. 1104).

[default] com.rti.dds.infrastructure.PUBLICATION_PRIORITY_UNDEFINED

[range] [-1, MAX_INT]

Referenced by **ChannelSettings_t.ChannelSettings_t()**.

8.31 ChannelSettingsSeq Class Reference

Declares IDL sequence< **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411) >

Inherits ArraySequence.

8.31.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411) `>`

A sequence of **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411) used to configure the channels' properties. If the length of the sequence is zero, the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) has no effect.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

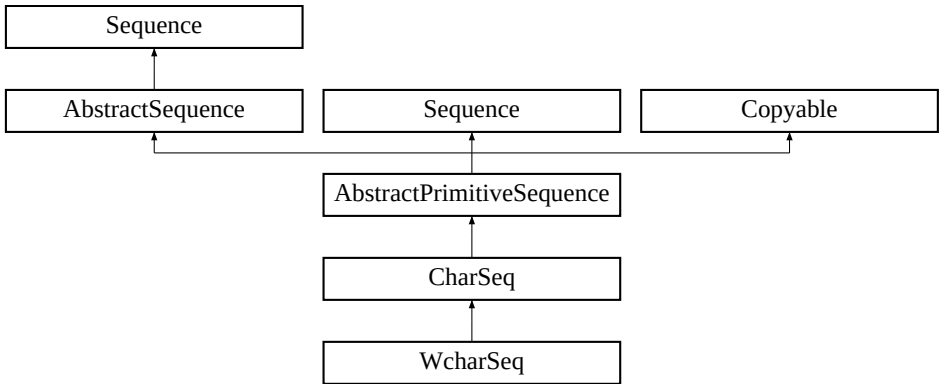
See also

com.rti.dds.infrastructure.ChannelSettings_t (p. 411)

8.32 CharSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < char >

Inheritance diagram for CharSeq:



Public Member Functions

- **CharSeq** ()

  Constructs an empty sequence of single-byte (serialized) characters with an initial maximum of zero.
- **CharSeq** (int initialMaximum)

  Constructs an empty sequence of single-byte (serialized) characters with the given initial maximum.
- **CharSeq** (char[ ] chars)

  Constructs a new sequence containing the given single-byte (serialized) characters.
- final boolean **addAllChar** (char[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- final boolean **addAllChar** (char[ ] elements)

  Append the elements of the given array into this sequence.

- final void **addChar** (char element)

  Append the element to the end of the sequence.
- final void **addChar** (int index, char element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- final char **getChar** (int index)

  Returns the character at the given index.
- final char **setChar** (int index, char element)

  Set the new character at the given index and return the old character.
- final void **setChar** (int dstIndex, char[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- final char[ ] **toArrayChar** (char[ ] array)

  Return an array containing copy of the contents of this sequence.
- final int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- final Object **get** (int index)

  *A wrapper for **getChar(int)** (p. 419) that returns a java.lang.Character.*
- final Object **set** (int index, Object element)

  *A wrapper for **setChar()** (p. 420).*
- final void **add** (int index, Object element)

  A wrapper for addChar(int, int).

8.32.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < char >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

char

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.32.2 Constructor & Destructor Documentation

8.32.2.1 CharSeq() [1/3]

```
CharSeq ( )
```

Constructs an empty sequence of single-byte (serialized) characters with an initial maximum of zero.

8.32.2.2 CharSeq() [2/3]

```
CharSeq (
            int initialMaximum )
```

Constructs an empty sequence of single-byte (serialized) characters with the given initial maximum.

8.32.2.3 CharSeq() [3/3]

```
CharSeq (
            char[] chars )
```

Constructs a new sequence containing the given single-byte (serialized) characters.

Parameters

chars	the initial contents of this sequence

References **CharSeq.addAllChar()**.

8.32.3 Member Function Documentation

8.32.3.1 addAllChar() [1/2]

```
final boolean addAllChar (
            char[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **CharSeq.addAllChar()**, and **CharSeq.CharSeq()**.

8.32.3.2 addAllChar() [2/2]

```
final boolean addAllChar (
            char[] elements )
```

Append the elements of the given array into this sequence.

Exceptions

NullPointerException	if the given array is null

References **CharSeq.addAllChar()**.

8.32.3.3 addChar() [1/2]

```
final void addChar (
            char element )
```

Append the element to the end of the sequence.

Referenced by **CharSeq.add()**.

8.32.3.4 addChar() [2/2]

```
final void addChar (
            int index,
            char element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.32.3.5 getChar()

```
final char getChar (
            int index )
```

Returns the character at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **CharSeq.get()**.

8.32.3.6 setChar() [1/2]

```
final char setChar (
            int index,
            char element )
```

Set the new character at the given index and return the old character.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **CharSeq.set()**.

8.32.3.7 setChar() [2/2]

```
final void setChar (
            int dstIndex,
            char[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.32.3.8 toArrayChar()

```
final char[] toArrayChar (
             char[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.32.3.9 getMaximum()

```
final int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 422), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_char_seq()**, and **DynamicData.set_char_seq()**.

8.32.3.10 get()

```
final Object get (
            int index )
```

A wrapper for **getChar(int)** (p. 419) that returns a java.lang.Character.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **CharSeq.getChar()**.

8.32.3.11 set()

```
final Object set (
            int index,
            Object element )
```

A wrapper for **setChar()** (p. 420).

Exceptions

ClassCastException	if the element is not of type Character.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **CharSeq.setChar()**.

8.32.3.12 add()

```
final void add (
            int index,
            Object element )
```

A wrapper for addChar(int, int).

Exceptions

ClassCastException	if the element is not of type Character.

See also

java.util.List::add(int, java.lang.Object)

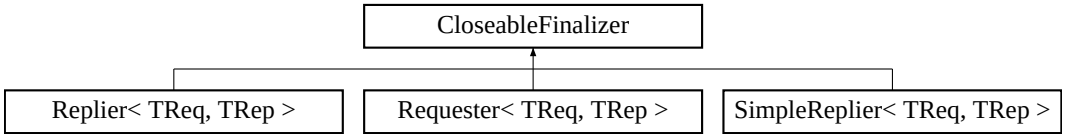
Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **CharSeq.addChar()**.

8.33 CloseableFinalizer Class Reference

Ensures that subclasses get automatically closed when they are garbage collected.

Inheritance diagram for CloseableFinalizer:



8.33.1 Detailed Description

Ensures that subclasses get automatically closed when they are garbage collected.

This class extends `java.io.Closeable` and subclasses implement `close()`

8.34 CoherentSetInfo_t Class Reference

<<***extension***>> *(*p. *155)* Type definition for a coherent set info.

Inherits Struct.

Public Attributes

- **GUID_t group_guid** = new **GUID_t**(**GUID_t.GUID_UNKNOWN**)
  The coherent set group GUID.
- **SequenceNumber_t coherent_set_sequence_number**
  Sequence number that identifies a sample as part of a ***com.rti.dds.publication.DataWriter*** *(*p. *553) coherent set.*
- **SequenceNumber_t group_coherent_set_sequence_number**
  Sequence number that identifies a sample as part of a group coherent set.
- boolean **incomplete_coherent_set** = true
  Indicates if a sample is part of an incomplete coherent set.

8.34.1 Detailed Description

<<***extension***>> *(p. 155)* Type definition for a coherent set info.

A CoherentSetInfo provides information about the coherent set associated with a sample.

8.34.2 Member Data Documentation

8.34.2.1 group_guid

```
GUID_t group_guid = new  GUID_t( GUID_t.GUID_UNKNOWN)
```

The coherent set group GUID.

This GUID identifies the **com.rti.dds.publication.DataWriter** (p. 553) or the group of DataWriters publishing the coherent set, depending on the value of **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) in the **com.rti.dds.subscription.Subscriber** (p. 1730).

8.34.2.2 coherent_set_sequence_number

```
SequenceNumber_t coherent_set_sequence_number
```

Initial value:

```
=
            new SequenceNumber_t(SequenceNumber_t.SEQUENCE_NUMBER_UNKNOWN)
```

Sequence number that identifies a sample as part of a **com.rti.dds.publication.DataWriter** (p. 553) coherent set.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) in the **com.rti.dds.subscription.↩ Subscriber** (p. 1730) is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQos↩ PolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS or com.rti.dds.infrastructure.PresentationQosPolicy↩ AccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS, the coherent set associated with a sample is identified by the pair (group_guid, coherent_set_sequence_number).

8.34.2.3 group_coherent_set_sequence_number

```
SequenceNumber_t group_coherent_set_sequence_number
```

Initial value:

```
=
            new SequenceNumber_t(SequenceNumber_t.SEQUENCE_NUMBER_UNKNOWN)
```

Sequence number that identifies a sample as part of a group coherent set.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) in the **com.rti.dds.subscription.↩ Subscriber** (p. 1730) is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQos↩ PolicyAccessScopeKind.GROUP_PRESENTATION_QOS, the coherent set associated with a sample is identified by the pair (group_guid, group_coherent_set_sequence_number).

8.34.2.4 incomplete_coherent_set

```
boolean incomplete_coherent_set = true
```

Indicates if a sample is part of an incomplete coherent set.

8.35 CompressionSettings_t Class Reference

<<***extension***>> *(*p. *155)* Settings related to compressing user data.

Inherits Struct.

Public Member Functions

- **CompressionSettings_t** ()

  Constructor with default values of ***com.rti.dds.infrastructure.CompressionSettings_t.compression_ids*** *(*p. *427),* ***com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level*** *(*p. *428) and* ***com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_threshold*** *(*p. *428).*
- **CompressionSettings_t** (int **compression_ids**, int **writer_compression_level**, int **writer_compression_threshold**)

  Constructor with the given values for the ***com.rti.dds.infrastructure.CompressionSettings_t.compression_ids*** *(*p. *427),* ***com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level*** *(*p. *428) and* ***com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_threshold*** *(*p. *428).*

Public Attributes

- int **compression_ids**

  <<***extension***>> *(*p. *155) Mask that represents the compression algorithms enabled.*
- int **writer_compression_level**

  <<***extension***>> *(*p. *155) The level of compression to use when compressing data.*
- int **writer_compression_threshold**

  <<***extension***>> *(*p. *155) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.*

Static Public Attributes

- static final int **COMPRESSION_LEVEL_BEST_COMPRESSION** = 10

  <<***extension***>> *(*p. *155) The best compression ratio possible for a compression algorithm.*
- static final int **COMPRESSION_LEVEL_BEST_SPEED** = 0

  <<***extension***>> *(*p. *155) The best compression speed possible for a compression algorithm.*
- static final int **COMPRESSION_LEVEL_DEFAULT** = **COMPRESSION_LEVEL_BEST_COMPRESSION**

  <<***extension***>> *(*p. *155) Default level of compression.*

8.35.1 Detailed Description

<<***extension***>> *(*p. *155)* Settings related to compressing user data.

QoS:

com.rti.dds.infrastructure.DataRepresentationQosPolicy (p. 544)

8.35.2 Constructor & Destructor Documentation

8.35.2.1 CompressionSettings_t() [1/2]

```
CompressionSettings_t ( )
```

Constructor with default values of **com.rti.dds.infrastructure.CompressionSettings_t.compression_ids** (p. 427), **com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level** (p. 428) and **com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_threshold** (p. 428).

References **CompressionSettings_t.COMPRESSION_LEVEL_DEFAULT**.

8.35.2.2 CompressionSettings_t() [2/2]

```
CompressionSettings_t (
            int compression_ids,
            int writer_compression_level,
            int writer_compression_threshold )
```

Constructor with the given values for the **com.rti.dds.infrastructure.CompressionSettings_t.compression_ids** (p. 427), **com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level** (p. 428) and **com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_threshold** (p. 428).

References **CompressionSettings_t.compression_ids**, **CompressionSettings_t.writer_compression_level**, and **CompressionSettings_t.writer_compression_threshold**.

8.35.3 Member Data Documentation

8.35.3.1 COMPRESSION_LEVEL_BEST_COMPRESSION

```
final int COMPRESSION_LEVEL_BEST_COMPRESSION = 10  [static]
```

<<***extension***>> *(*p. *155)* The best compression ratio possible for a compression algorithm.

See also

com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level (p. 428)

8.35.3.2 COMPRESSION_LEVEL_BEST_SPEED

```
final int COMPRESSION_LEVEL_BEST_SPEED = 0  [static]
```

<<***extension***>> *(*p. *155)* The best compression speed possible for a compression algorithm.

See also

com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level (p. 428)

8.35.3.3 COMPRESSION_LEVEL_DEFAULT

```
final int COMPRESSION_LEVEL_DEFAULT = COMPRESSION_LEVEL_BEST_COMPRESSION  [static]
```

<<***extension***>> *(*p. *155)* Default level of compression.

See also

com.rti.dds.infrastructure.CompressionSettings_t.writer_compression_level (p. 428)

Referenced by **CompressionSettings_t.CompressionSettings_t()**.

8.35.3.4 compression_ids

```
int compression_ids
```

<<***extension***>> *(*p. *155)* Mask that represents the compression algorithms enabled.

A bitmap that represents the compression algorithm IDs (com.rti.dds.infrastructure.CompressionIdMask) that are supported by the endpoint. The **com.rti.dds.publication.DataWriter** (p. 553) creation will fail if more than one algorithm is provided.

If a **com.rti.dds.publication.DataWriter** (p. 553) inherits multiple compression IDs from a **com.rti.dds.topic.Topic** (p. 1807), only the least significant bit enabled will be inherited. This forces the following order of preference: com.↩ rti.dds.infrastructure.DDS_COMPRESSION_ID_ZLIB, com.rti.dds.infrastructure.DDS_COMPRESSION_ID_BZIP2, com.rti.dds.infrastructure.DDS_COMPRESSION_ID_LZ4.

Interactions with Security and Batching: Currently, the only algorithm that is supported when compression and batching are enabled on the same **com.rti.dds.publication.DataWriter** (p. 553) is com.rti.dds.infrastructure.DDS_↩ COMPRESSION_ID_ZLIB.

The combination of compression, batching, and data protection is supported. First, compression is applied to the entire batch. Then, data protection is applied to the compressed batch.

Note: When **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.serialize_key_with_dispose** (p. 599) is enabled and a dispose message is sent, the serialized key is not compressed.

[default] For **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.DataWriter** (p. 553) a com.rti.dds.↩ infrastructure.CompressionIdMask mask set to com.rti.dds.infrastructure.CompressionIdMaskBits.MASK_NONE

[default] For **com.rti.dds.subscription.DataReader** (p. 450) a com.rti.dds.infrastructure.CompressionIdMask mask set to com.rti.dds.infrastructure.CompressionIdMaskBits.MASK_ALL.

Referenced by **CompressionSettings_t.CompressionSettings_t()**.

8.35.3.5 writer_compression_level

```
int writer_compression_level
```

<<***extension***>> *(*p. *155)* The level of compression to use when compressing data.

Compression algorithms typically allow you to choose a level with which to compress the data. Each level has trade-offs between the resulting compression ratio and the speed of compression.

[range] [0, 10]

The value 1 represents the fastest compression time and the lowest compression ratio. The value 10 represents the slowest compression time but the highest compression ratio.

A value of 0 disables compression.

[default] com.rti.dds.infrastructure.COMPRESSION_LEVEL_BEST_COMPRESSION

Note

Only available for a **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.topic.Topic** (p. 1807).

Referenced by **CompressionSettings_t.CompressionSettings_t()**.

8.35.3.6 writer_compression_threshold

```
int writer_compression_threshold
```

<<***extension***>> *(*p. *155)* The threshold, in bytes, above which a serialized sample will be eligible to be compressed.

Any sample with a serialized size greater than or equal to the threshold will be eligible to be compressed. All samples with an eligible serialized size will be compressed. Only if the compressed size is smaller than the serialized size will the sample be stored and sent compressed on the wire.

For batching we check the maximum serialized size of the batch, calculated as serialized_sample_max_size ∗ **com.rti.dds.infrastructure.BatchQosPolicy.max_samples** (p. 357)

[range] [0, 2147483647] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

Setting the threshold to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) disables the compression.

[default] com.rti.dds.infrastructure.COMPRESSION_THRESHOLD_DEFAULT (8192)

Note

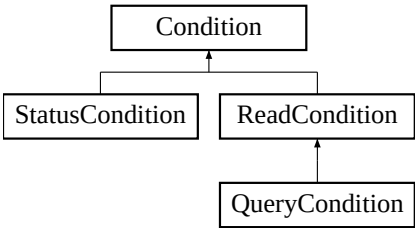
Only available for a **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.topic.Topic** (p. 1807).

Referenced by **CompressionSettings_t.CompressionSettings_t()**.

8.36 Condition Interface Reference

<<***interface***>> *(*p. *156)* Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Inheritance diagram for Condition:



Public Member Functions

- boolean **get_trigger_value** ()

  Retrieve the `trigger_value`.

8.36.1 Detailed Description

<<***interface***>> *(*p. *156)* Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.Wait↩ Set** (p. 1973).

This basic class is specialised in three classes:

com.rti.dds.infrastructure.GuardCondition (p. 1129), **com.rti.dds.infrastructure.StatusCondition** (p. 1699), and **com.rti.dds.subscription.ReadCondition** (p. 1514).

A **com.rti.dds.infrastructure.Condition** (p. 429) has a `trigger_value` that can be com.rti.dds.infrastructure.true or com.rti.dds.infrastructure.false and is set automatically by RTI Connext.

See also

com.rti.dds.infrastructure.WaitSet (p. 1973)

8.36.2 Member Function Documentation

8.36.2.1 get_trigger_value()

```
boolean get_trigger_value ( )
```

Retrieve the `trigger_value`.

Returns

the trigger value.

Implemented in **GuardCondition** (p. 1131).

8.37 ConditionSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.↩ Condition** (p. 429) >

Inherits AbstractNativeSequence.

Public Member Functions

- **ConditionSeq** (Collection conditions)

  *Create a **Condition** (p. 429) Sequence with the given contents. The size and the maximum of the new sequence will match the size of the given collection.*
- **ConditionSeq** ()

  *Create a empty **Condition** (p. 429) Sequence with a initial maximum of zero.*
- **ConditionSeq** (int initial_maximum)

  *Create a **Condition** (p. 429) Sequence with the specified initial maximum.*
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.37.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.↩ Condition** (p. 429) >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.WaitSet (p. 1973)

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.37.2 Constructor & Destructor Documentation

8.37.2.1 ConditionSeq() [1/3]

```
ConditionSeq (
            Collection conditions )
```

Create a **Condition** (p. 429) Sequence with the given contents. The size and the maximum of the new sequence will match the size of the given collection.

Parameters

conditions	The initial contents of the sequence

Exceptions

NullPointerException	if the given collection is null

8.37.2.2 ConditionSeq() [2/3]

```
ConditionSeq ( )
```

Create a empty **Condition** (p. 429) Sequence with a initial maximum of zero.

8.37.2.3 ConditionSeq() [3/3]

```
ConditionSeq (
            int initial_maximum )
```

Create a **Condition** (p. 429) Sequence with the specified initial maximum.

Parameters

initial_maximum	The initial maximum of the sequence.

8.37.3 Member Function Documentation

8.37.3.1 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 329), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

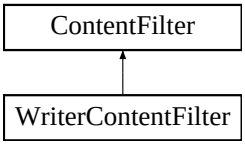
com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

8.38 ContentFilter Interface Reference

<<***interface***>> *(p. 156)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436)

Inheritance diagram for ContentFilter:



Public Member Functions

- void **compile** (**ObjectHolder** new_compile_data, String expression, **StringSeq** parameters, **TypeCode** type↩ _code, String type_class_name, Object old_compile_data)

  Compile an instance of the content filter according to the filter expression and parameters of the given data type.
- boolean **evaluate** (Object compile_data, Object sample, **FilterSampleInfo** meta_data)

  Evaluate whether the sample is passing the filter or not according to the sample content.
- void **finalize** (Object compile_data)

  A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

8.38.1 Detailed Description

<<***interface***>> *(p. 156)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436)

Entity:

com.rti.dds.topic.ContentFilteredTopic (p. 436)

This interface can be implemented by an application-provided class and then registered with the **com.rti.dds.domain.**↩ **DomainParticipant** (p. 670) such that samples can be filtered for a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) with the given filter name.

Note: the API for using a custom content filter is subject to change in a future release.

See also

com.rti.dds.topic.ContentFilteredTopic (p. 436)

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 740)

8.38.2 Member Function Documentation

8.38.2.1 compile()

```
void compile (
              ObjectHolder new_compile_data,
             String expression,
              StringSeq parameters,
              TypeCode type_code,
             String type_class_name,
             Object old_compile_data )
```

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

This method is called when an instance of the locally registered content filter is created or when the expression parameter for the locally registered content filter instance is changed.

An instance of the locally registered content filter is created every time a local **com.rti.dds.topic.ContentFiltered↩ Topic** (p. 436) with the matching filter name is created, or when a **com.rti.dds.subscription.DataReader** (p. 450) with a matching filter name is discovered.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

new_compile_data	*<<**out**>> (p. 156)* User specified opaque pointer of this instance of the content filter. This value is then passed to the **com.rti.dds.topic.ContentFilter.evaluate** (p. 435) and **com.rti.dds.topic.ContentFilter.finalize** (p. 435) functions for this instance of the content filter. Can be set to null .
expression	*<<**in**>> (p. 156)* An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connext and **must** not be freed. If you want to manipulate this string, you **must** first make a copy of it.
parameters	*<<**in**>> (p. 156)* A string sequence with the expression parameters the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) was created with. The string sequence is **equal** (but **not** identical) to the string sequence passed to **com.rti.dds.domain.DomainParticipant.create_contentfilteredtopic** (p. 710). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	*<<**in**>> (p. 156)* A pointer to the type code for the related **com.rti.dds.topic.Topic** (p. 1807) of the **com.rti.dds.topic.ContentFilteredTopic** (p. 436). A type_code is a description of a type in terms of which types it contains (such as long, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type. This parameter is always null in Java.
type_class_name	*<<**in**>> (p. 156)* Fully qualified class name of the related **com.rti.dds.topic.Topic** (p. 1807).
old_compile_data	*<<**in**>> (p. 156)* The previous new_compile_data value from a previous call to this instance of a content filter. If the compile function is called more than once for an instance of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), e.g., if the expression parameters are changed, then the new_compile_data value returned by the previous invocation is passed in the old_compile_data parameter (which can be null). If this is a new instance of the filter, NULL is passed. This parameter is useful for freeing or reusing previously allocated resources.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.38.2.2 evaluate()

```
boolean evaluate (
             Object compile_data,
             Object sample,
              FilterSampleInfo meta_data )
```

Evaluate whether the sample is passing the filter or not according to the sample content.

This method is called when a sample for a locally created **com.rti.dds.subscription.DataReader** (p. 450) associated with the filter is received, or when a sample for a discovered **com.rti.dds.subscription.DataReader** (p. 450) assocated with the filter needs to be sent.

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

Parameters

compile_data	<<***in***>> *(*p. *156)* The last return value of the **com.rti.dds.topic.ContentFilter.compile** (p. 434) function for this instance of the content filter. Can be null .
sample	<<***in***>> *(*p. *156)* Pointer to a deserialized sample to be filtered
meta_data	<<***in***>> *(*p. *156)* Pointer to meta data associated with the sample.

Returns

The function must return 0 if the sample should be filtered out, non zero otherwise

8.38.2.3 finalize()

```
void finalize (
             Object compile_data )
```

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

This method is called when an instance of the locally registered content filter is deleted.

An instance of the locally registered content filter is deleted every time a local **com.rti.dds.topic.ContentFiltered↩ Topic** (p. 436) with the matching filter name is deleted, or when a **com.rti.dds.subscription.DataReader** (p. 450) with a matching filter name is removed due to discovery.

This method is also called on all instances of the discovered **com.rti.dds.subscription.DataReader** (p. 450) with a matching filter name if the filter is unregistered with **com.rti.dds.domain.DomainParticipant.unregister_contentfilter** (p. 741).

It is possible for multiple threads to be calling into this function at the same time. However, this function will never be called on a content filter that has been unregistered.

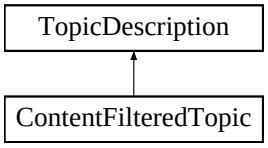
Parameters

compile_data	*<<**in**>> (p. 156)* The last return value of the **com.rti.dds.topic.ContentFilter.compile** (p. 434) function for this instance of the content filter. Can be null

8.39 ContentFilteredTopic Interface Reference

*<<**interface**>> (p. 156)* Specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows for content-based subscriptions.

Inheritance diagram for ContentFilteredTopic:



Public Member Functions

- String **get_filter_expression** ()

  Get the `filter_expression`*.*
- void **get_expression_parameters** (**StringSeq** parameters)

  Get the `expression_parameters`*.*
- void **set_expression_parameters** (**StringSeq** parameters)

  Set the `expression_parameters`*.*
- void **set_expression** (String expression, **StringSeq** parameters)

  Set the `filter_expression` *and* `expression_parameters`*.*
- **Topic get_related_topic** ()

  Get the `related_topic`*.*
- void **append_to_expression_parameter** (int index, String val)

  *<<**extension**>> (p. 155) Appends a string term to the specified parameter string.*
- void **remove_from_expression_parameter** (int index, String val)

  *<<**extension**>> (p. 155) Removes a string term from the specified parameter string.*

8.39.1 Detailed Description

<<***interface***>> *(p. 156)* Specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows for content-based subscriptions.

It describes a more sophisticated subscription that indicates a **com.rti.dds.subscription.DataReader** (p. 450) does not want to necessarily see all values of each instance published under the **com.rti.dds.topic.Topic** (p. 1807). Rather, it wants to see only the values whose contents satisfy certain criteria. This class therefore can be used to request content-based subscriptions.

The selection of the content is done using the `filter_expression` with parameters `expression_parameters`.

- The `filter_expression` attribute is a string that specifies the criteria to select the data samples of interest. It is similar to the WHERE part of an SQL clause.
- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.e. "%n" tokens) in the `filter_expression`. The number of supplied parameters must fit with the requested values in the `filter_expression` (i.e. the number of n tokens).

Queries and Filters Syntax (p. 104) describes the syntax of `filter_expression` and `expression_parameters`.

8.39.2 Member Function Documentation

8.39.2.1 get_filter_expression()

```
String get_filter_expression ( )
```

Get the `filter_expression`.

Return the `filter_expression` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 436). `filter_expression` is either specified on the last successful call to **com.rti.dds.topic.ContentFilteredTopic.set_expression** (p. 438) or, if that method is never called, the expression specified when the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) was created.

Returns

the `filter_expression`.

8.39.2.2 get_expression_parameters()

```
void get_expression_parameters (
               StringSeq parameters )
```

Get the `expression_parameters`.

Return the `expression_parameters` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 436). `expression_parameters` is either specified on the last successful call to **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 438), **com.rti.dds.topic.ContentFilteredTopic.set_expression** (p. 438) or, if that method is never called, the parameters specified when the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) was created.

Parameters

parameters	<<***inout***>> *(*p. *156)* the filter expression parameters. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.create_contentfilteredtopic (p. 710)

com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters (p. 438)

8.39.2.3 set_expression_parameters()

```
void set_expression_parameters (
              StringSeq parameters )
```

Set the `expression_parameters`.

Change the `expression_parameters` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

Parameters

parameters	<<***in***>> *(*p. *156)* the filter expression parameters Cannot be NULL. Length of sequence cannot be greater than 100.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.39.2.4 set_expression()

```
void set_expression (
             String expression,
              StringSeq parameters )
```

Set the `filter_expression` and `expression_parameters`.

Changes the `filter_expression` and `expression_parameters` associated with the **com.rti.dds.topic.↩ ContentFilteredTopic** (p. 436).

Parameters

expression	<<***in***>> *(p. 156)* the filter expression. Cannot be NULL.
parameters	<<***in***>> *(p. 156)* the filter expression parameters Cannot be NULL. Length of sequence cannot be greater than 100.

8.39.2.5 get_related_topic()

```
Topic get_related_topic ( )
```

Get the `related_topic`.

Return the **com.rti.dds.topic.Topic** (p. 1807) specified when the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) was created.

Returns

The **com.rti.dds.topic.Topic** (p. 1807) assocated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

8.39.2.6 append_to_expression_parameter()

```
void append_to_expression_parameter (
             int index,
             String val )
```

<<***extension***>> *(p. 155)* Appends a string term to the specified parameter string.

Appends the input string to the end of the specified parameter string, separated by a comma. If the original parameter string is enclosed in quotation marks (''), the resultant string will also be enclosed in quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to add a pattern to the match pattern list. For example, if the filter expression parameter value is:

'IBM'

Then append_to_expression_parameter(0, "MSFT") would generate the new value:

'IBM,MSFT'

Parameters

index	<<***in***>> *(p. 156)* The index of the parameter string to be modified. The first index is index 0. When using the **com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50) filter, `index` *must* be 0.
val	<<***in***>> *(p. 156)* The string term to be appended to the parameter string.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.39.2.7 remove_from_expression_parameter()

```
void remove_from_expression_parameter (
            int index,
            String val )
```

<<***extension***>> *(p. 155)* Removes a string term from the specified parameter string.

Removes the input string from the specified parameter string. To be found and removed, the input string must exist as a complete term, bounded by comma separators or the strong boundary. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks. If the removed term was the last entry in the string, the result will be a string of empty quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to remove a pattern from the match pattern list. For example, if the filter expression paremeter value is:

'IBM,MSFT'

Then remove_from_expression_parameter(0, "IBM") would generate the expression:

'MSFT'

Parameters

index	<<***in***>> *(p. 156)* The index of the parameter string to be modified. The first index is index 0. When using the **com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50) filter, `index` *must* be 0.
val	<<***in***>> *(p. 156)* The string term to be removed from the parameter string.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.40 ContentFilterProperty_t Class Reference

<<***extension***>> *(p. 155)* Type used to provide all the required information to enable content filtering.

Inherits Struct.

Public Member Functions

- **ContentFilterProperty_t** ()

  Constructor.

Public Attributes

- String **content_filter_topic_name**

  Name of the Content-filtered Topic associated with the Reader.
- String **related_topic_name**

  Name of the Topic related to the Content-filtered Topic.
- String **filter_class_name**

  Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).
- String **filter_expression**

  The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.
- final **StringSeq expression_parameters**

  Defines the value for each parameter in the filter expression.

8.40.1 Detailed Description

<<***extension***>> *(*p. *155)* Type used to provide all the required information to enable content filtering.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 ContentFilterProperty_t()

```
ContentFilterProperty_t ( )
```

Constructor.

8.40.3 Member Data Documentation

8.40.3.1 content_filter_topic_name

```
String content_filter_topic_name
```

Name of the Content-filtered Topic associated with the Reader.

8.40.3.2 related_topic_name

```
String related_topic_name
```

Name of the Topic related to the Content-filtered Topic.

8.40.3.3 filter_class_name

```
String filter_class_name
```

Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).

8.40.3.4 filter_expression

```
String filter_expression
```

The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

8.40.3.5 expression_parameters

```
final  StringSeq expression_parameters
```

Defines the value for each parameter in the filter expression.

8.41 Cookie_t Class Reference

<<***extension***>> *(*p. *155)* Sequence of bytes.

Inherits Struct.

Public Attributes

- final **ByteSeq value**
  a sequence of octets

8.41.1 Detailed Description

<<***extension***>> *(*p. *155)* Sequence of bytes.

8.41.2 Member Data Documentation

8.41.2.1 value

```
final  ByteSeq value
```

a sequence of octets

[default] Empty (zero-sized)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.42 CookieSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.Cookie_t** (p. 442) > .

Inherits ArraySequence.

8.42.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.Cookie_t** (p. 442) > .

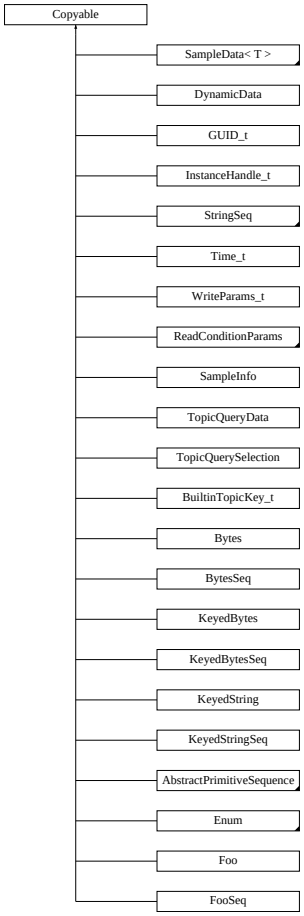
See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.43 Copyable Interface Reference

*<<**extension**>> (p. 155) <<**interface**>> (p. 156)* Interface for all the user-defined data type classes that support copy.

Inheritance diagram for Copyable:



Public Member Functions

- Object **copy_from** (Object src)

  Copy value of a data type from source.

8.43.1 Detailed Description

*<<**extension**>> (p. 155) <<**interface**>> (p. 156)* Interface for all the user-defined data type classes that support copy.

A class implements the **com.rti.dds.infrastructure.Copyable** (p. 444) interface to indicate that it allows its entire state to be replaced with the state of another object. This state copy is a deep copy, such that subsequent changes to any part of one object will not be observed in the other.

Therefore, in general, object references in this object cannot simply be reassigned to those in the source object. (Strings are an exception to this rule, because they are immutable.)

8.43.2 Member Function Documentation

8.43.2.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

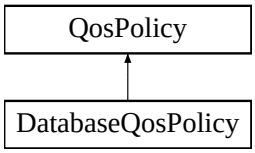
NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implemented in **SampleInfo** (p. 1638), **DynamicData** (p. 861), **GUID_t** (p. 1133), **InstanceHandle_t** (p. 1154), **StringSeq** (p. 1720), **Time_t** (p. 1802), **WriteParams_t** (p. 1996), **QueryConditionParams** (p. 1513), **ReadConditionParams** (p. 1518), **TopicQueryData** (p. 1831), **TopicQuerySelection** (p. 1837), **BuiltinTopicKey_t** (p. 372), **Bytes** (p. 386), **BytesSeq** (p. 404), **KeyedBytes** (p. 1174), **KeyedBytesSeq** (p. 1197), **KeyedString** (p. 1205), **KeyedStringSeq** (p. 1225), **AbstractPrimitiveSequence** (p. 327), **Enum** (p. 1041), **Foo** (p. 1066), and **FooSeq** (p. 1117).

8.44 DatabaseQosPolicy Class Reference

Various threads and resource limits settings used by RTI Connext to control its internal database.

Inheritance diagram for DatabaseQosPolicy:



Public Attributes

- final **ThreadSettings_t thread**

  Database thread settings.
- final **Duration_t shutdown_timeout**

  The maximum wait time during a shutdown.
- final **Duration_t cleanup_period**

  The period at which the database thread wakes up to remove deleted records.
- final **Duration_t shutdown_cleanup_period**

  The clean-up period used during database shut-down.
- int **initial_records**

  The initial number of total records.
- int **max_skiplist_level**

  The maximum level of the skiplist.
- int **initial_weak_references**

  The initial number of weak references.
- int **max_weak_references**

  The maximum number of weak references.

8.44.1 Detailed Description

Various threads and resource limits settings used by RTI Connext to control its internal database.

RTI uses an internal in-memory "database" to store information about entities created locally as well as remote entities found during the discovery process. This database uses a background thread to garbage-collect records related to deleted entities. When the **com.rti.dds.domain.DomainParticipant** (p. 670) that maintains this database is deleted, it shuts down this thread.

The Database QoS policy is used to configure how RTI Connext manages its database, including how often it cleans up, the priority of the database thread, and limits on resources that may be allocated by the database.

You may be interested in modifying the **com.rti.dds.infrastructure.DatabaseQosPolicy.shutdown_timeout** (p. 447) and **com.rti.dds.infrastructure.DatabaseQosPolicy.shutdown_cleanup_period** (p. 447) parameters to decrease the time it takes to delete a **com.rti.dds.domain.DomainParticipant** (p. 670) when your application is shutting down.

The **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 803) controls the memory allocation for elements stored in the database.

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) **NO** (p. 256)

8.44.2 Member Data Documentation

8.44.2.1 thread

```
final ThreadSettings_t thread
```

Database thread settings.

There is only one database thread: the clean-up thread.

[default] Priority: LOW.
The actual value depends on your architecture:

For Windows: -3
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Stack Size: The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] Mask: **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1797)

8.44.2.2 shutdown_timeout

```
final Duration_t shutdown_timeout
```

The maximum wait time during a shutdown.

The domain participant will exit after the timeout, even if the database has not been fully cleaned up.

[default] 15 seconds

[range] [0,**com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)]

8.44.2.3 cleanup_period

```
final Duration_t cleanup_period
```

The period at which the database thread wakes up to remove deleted records.

[default] 61 seconds

[range] [0,1 year]

8.44.2.4 shutdown_cleanup_period

```
final  Duration_t shutdown_cleanup_period
```

The clean-up period used during database shut-down.

If you would like to shorten the time taken for a DomainParticipant to shutdown, you can decrease this value.

It is recommended to set this value to something other than 0 if running in an RTOS environment, to avoid CPU starvation.

[default] 10 milliseconds

[range] [0,1 year]

8.44.2.5 initial_records

```
int initial_records
```

The initial number of total records.

[default] 1024

[range] [1,10 million]

8.44.2.6 max_skiplist_level

```
int max_skiplist_level
```

The maximum level of the skiplist.

The skiplist is used to keep records in the database. Usually, the search time is log2(N), where N is the total number of records in one skiplist. However, once N exceeds 2^n, where n is the maximum skiplist level, the search time will become more and more linear. Therefore, the maximum level should be set such that 2^n is larger than the maximum(N among all skiplists). Usually, the maximum N is the maximum number of remote and local writers or readers.

[default] 7

[range] [1,31]

8.44.2.7 initial_weak_references

```
int initial_weak_references
```

The initial number of weak references.

See **com.rti.dds.infrastructure.DatabaseQosPolicy.max_weak_references** (p. 449) for more information about what a weak reference is.

If the QoS set contains an initial_weak_references value that is too small to ever grow to **com.rti.dds.infrastructure.↩ DatabaseQosPolicy.max_weak_references** (p. 449) using RTI Connext' internal algorithm, this value will be adjusted upwards as necessary. Subsequent accesses of this value will reveal the actual initial value used.

Changing the value of this field is an advanced feature; it is recommended that you consult with RTI support personnel before doing so.

[default] 2049, which is the minimum initial value imposed by REDA when the maximum is unlimited. If a lower value is specified, it will simply be increased to 2049 automatically.

[range] [1, 100 million], <= max_weak_references

See also

com.rti.dds.infrastructure.DatabaseQosPolicy.max_weak_references (p. 449)

8.44.2.8 max_weak_references

```
int max_weak_references
```

The maximum number of weak references.

A weak reference is an internal data structure that refers to a record within RTI Connext' internal database. This field configures the maximum number of such references that RTI Connext may create.

The actual number of weak references is permitted to grow from an initial value (indicated by **com.rti.dds.↩ infrastructure.DatabaseQosPolicy.initial_weak_references** (p. 448)) to this maximum. To prevent RTI Connext from allocating any weak references after the system has reached a steady state, set the initial and maximum values equal to one another. To indicate that the number of weak references should continue to grow as needed indefinitely, set this field to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259). Be aware that although a single weak reference occupies very little memory, allocating a very large number of them can have a significant impact on your overall memory usage.

Tuning this value precisely is difficult without intimate knowledge of the structure of RTI Connext' database; doing so is an advanced feature not required by most applications. The default value has been chosen to be sufficient for reasonably large systems. If you believe you may need to modify this value, please consult with RTI support personnel for assistance.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_weak_references

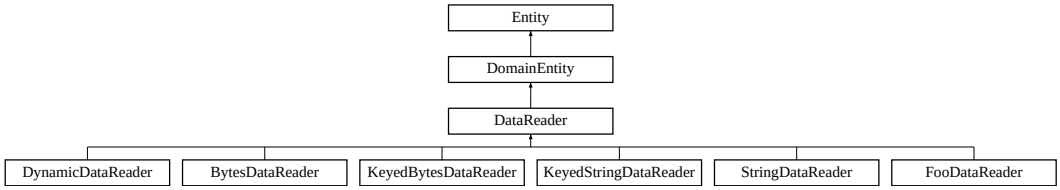
See also

com.rti.dds.infrastructure.DatabaseQosPolicy.initial_weak_references (p. 448)

8.45 DataReader Interface Reference

<<***interface***>> *(*p. *156)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1730).

Inheritance diagram for DataReader:



Public Member Functions

- **ReadCondition create_readcondition** (int sample_states, int view_states, int instance_states)

  *Creates a **com.rti.dds.subscription.ReadCondition** (*p. *1514).*
- **ReadCondition create_readcondition_w_params** (**ReadConditionParams** params)

  <<***extension***>> *(*p. *155) Creates a **com.rti.dds.subscription.ReadCondition** (*p. *1514) with parameters.*
- **QueryCondition create_querycondition** (int sample_states, int view_states, int instance_states, String query_expression, **StringSeq** query_parameters)

  *Creates a **com.rti.dds.subscription.QueryCondition** (*p. *1510).*
- **QueryCondition create_querycondition_w_params** (**QueryConditionParams** params)

  <<***extension***>> *(*p. *155) Creates a **com.rti.dds.subscription.QueryCondition** (*p. *1510) with parameters.*
- void **delete_readcondition** (**ReadCondition** condition)

  *Deletes a **com.rti.dds.subscription.ReadCondition** (*p. *1514) or **com.rti.dds.subscription.QueryCondition** (*p. *1510) attached to the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **set_qos** (**DataReaderQos** qos)

  Sets the reader QoS.
- void **set_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Changes the QoS of this reader using the input XML QoS profile.*
- void **get_qos** (**DataReaderQos** qos)

  Gets the reader QoS.
- void **set_listener** (**DataReaderListener** l, int mask)

  Sets the reader listener.
- **DataReaderListener get_listener** ()

  Gets the reader listener.
- void **call_listenerT** (int mask)

  Calls the reader listener.
- void **get_sample_rejected_status** (**SampleRejectedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.
- void **get_liveliness_changed_status** (**LivelinessChangedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.
- void **get_requested_deadline_missed_status** (**RequestedDeadlineMissedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

- void **get_requested_incompatible_qos_status** (**RequestedIncompatibleQosStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.
- void **get_sample_lost_status** (**SampleLostStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.
- void **get_subscription_matched_status** (**SubscriptionMatchedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.
- void **get_datareader_cache_status** (**DataReaderCacheStatus** status)

  *<<**extension**>> (*p. *155) Gets the datareader cache status for this reader.*
- void **get_datareader_protocol_status** (**DataReaderProtocolStatus** status)

  *<<**extension**>> (*p. *155) Gets the datareader protocol status for this reader.*
- void **get_matched_publication_datareader_protocol_status** (**DataReaderProtocolStatus** status, **InstanceHandle_t** publication_handle)

  *<<**extension**>> (*p. *155) Gets the datareader protocol status for this reader, per matched publication identified by the publication_handle.*
- void **get_matched_publications** (**InstanceHandleSeq** publication_handles)

  *Retrieves the list of publications currently "associated" with this **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **get_matched_publication_data** (**PublicationBuiltinTopicData** publication_data, **InstanceHandle_t** publication_handle)

  *Retrieves the information on a publication that is currently "associated" with the **com.rti.dds.subscription.DataReader** (*p. *450).*
- boolean **is_matched_publication_alive** (**InstanceHandle_t** publication_handle)

  *Check if a publication currently matched with a **DataReader** (*p. *450) is alive.*
- void **get_matched_publication_participant_data** (**ParticipantBuiltinTopicData** participant_data, **InstanceHandle_t** publication_handle)

  *<<**extension**>> (*p. *155) Retrieves the information on the discovered **com.rti.dds.domain.DomainParticipant** (*p. *670) associated with the publication that is currently matching with the **com.rti.dds.subscription.DataReader** (*p. *450).*
- **TopicDescription get_topicdescription** ()

  *Returns the **com.rti.dds.topic.TopicDescription** (*p. *1820) associated with the **com.rti.dds.subscription.DataReader** (*p. *450).*
- **Subscriber get_subscriber** ()

  *Returns the **com.rti.dds.subscription.Subscriber** (*p. *1730) to which the **com.rti.dds.subscription.DataReader** (*p. *450) belongs.*
- void **delete_contained_entities** ()

  *Deletes all the entities that were created by means of the "create" operations on the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **wait_for_historical_data** (**Duration_t** max_wait)

  *Waits until all "historical" data is received for **com.rti.dds.subscription.DataReader** (*p. *450) entities that have a non-VOLATILE Durability Qos kind.*
- void **acknowledge_sample** (**SampleInfo** sample_info)

  *<<**extension**>> (*p. *155) Acknowledges a single sample explicitly.*
- void **acknowledge_all** ()

  *<<**extension**>> (*p. *155) Acknowledges all previously accessed samples.*
- void **acknowledge_sample** (**SampleInfo** sample_info, **AckResponseData_t** response_data)

  *<<**extension**>> (*p. *155) Acknowledges a single sample explicitly.*
- void **acknowledge_all** (**AckResponseData_t** response_data)

  *<<**extension**>> (*p. *155) Acknowledges all previously accessed samples.*
- **TopicQuery create_topic_query** (**TopicQuerySelection** selection)

*Creates a **com.rti.dds.subscription.TopicQuery** (p. 1830).*

- void **delete_topic_query** (**TopicQuery** query)

  *Deletes a **com.rti.dds.subscription.TopicQuery** (p. 1830).*

- **TopicQuery lookup_topic_query** (**GUID_t** guid)

  *Retrieves an existing **com.rti.dds.subscription.TopicQuery** (p. 1830).*

- void **read_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  Reads data samples, if any are available.

- void **take_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  Takes data samples, if any are available.

- void **read_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** read_condition)

  Reads data samples, if any are available.

- void **take_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** read_condition)

  Takes data samples, if any are available.

- void **read_next_sample_untyped** (Object received_data, **SampleInfo** sample_info)

  Reads data samples, if any are available.

- void **take_next_sample_untyped** (Object received_data, **SampleInfo** sample_info)

  Takes data samples, if any are available.

- void **read_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  Reads data samples, if any are available.

- void **take_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  Takes data samples, if any are available.

- void **read_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **GUID_t** topic_query_guid, int sample_states, int view_states, int instance_↩ states)

  Reads data samples, if any are available.

- void **take_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **GUID_t** topic_query_guid, int sample_states, int view_states, int instance_↩ states)

  Takes data samples, if any are available.

- void **read_instance_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_↩ samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

  Reads data samples, if any are available.

- void **take_instance_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_↩ samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

  Takes data samples, if any are available.

- void **read_next_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  Reads data samples, if any are available.

- void **take_next_instance_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  Takes data samples, if any are available.

- void **read_next_instance_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

Reads data samples, if any are available.

- void **take_next_instance_w_condition_untyped** (List<?> received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

  Takes data samples, if any are available.
- void **return_loan_untyped** (List<?> received_data, **SampleInfoSeq** info_seq)

  Returns loaned sample data and meta-data.
- void **get_key_value_untyped** (Object key_holder, **InstanceHandle_t** handle)

  Fills in the key fields of the given data sample.
- **InstanceHandle_t lookup_instance_untyped** (Object key_value)
- void **take_discovery_snapshot** ()

  Take a snapshot of the compatible and incompatible remote writers matched by a local reader.
- void **take_discovery_snapshot** (String file_name)

  Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

Static Public Attributes

- static final **TopicQuerySelection TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER**

  *Special value for creating a **com.rti.dds.subscription.TopicQuery** (p. 1830) that applies the same filter as the **DataReader** (p. 450)'s **com.rti.dds.topic.ContentFilteredTopic** (p. 436).*
- static final **TopicQuerySelection TOPIC_QUERY_SELECTION_SELECT_ALL**

  *Special value for creating a **com.rti.dds.subscription.TopicQuery** (p. 1830) that selects all the samples in a **com.rti.dds.publication.DataWriter** (p. 553) cache.*

8.45.1 Detailed Description

<<***interface***>> *(p. 156)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a subscription) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1730).

QoS:

com.rti.dds.subscription.DataReaderQos (p. 517)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS;
com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS, **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1239);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS, **com.rti.dds.subscription.RequestedDeadlineMissedStatus** (p. 1560);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.subscription.RequestedIncompatibleQosStatus** (p. 1561);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS, **com.rti.dds.subscription.SampleLostStatus** (p. 1648);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS, **com.rti.dds.subscription.SampleRejectedStatus** (p. 1657);
com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS, **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773);

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

A **com.rti.dds.subscription.DataReader** (p. 450) refers to exactly one **com.rti.dds.topic.TopicDescription** (p. 1820) (either a **com.rti.dds.topic.Topic** (p. 1807), a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) or a **com.rti.dds.topic.MultiTopic** (p. 1320)) that identifies the data to be read.

The subscription has a unique resulting type. The data-reader may give access to several instances of the resulting type, which can be distinguished from each other by their `key`.

com.rti.dds.subscription.DataReader (p. 450) is an abstract class. It must be specialised for each particular application data-type (see **USER_DATA** (p. 278)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **`com.rti.ndds.example.Foo`** (p. 1066) are specified in the generic type **com.rti.ndds.example.FooDataReader** (p. 1067).

The following operations may be called even if the **com.rti.dds.subscription.DataReader** (p. 450) is not enabled. Other operations will fail with the value **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) if called on a disabled **com.rti.dds.subscription.DataReader** (p. 450):

- The base-class operations **com.rti.dds.subscription.DataReader.set_qos** (p. 457), **com.rti.dds.subscription.DataReader.get_qos** (p. 459), **com.rti.dds.subscription.DataReader.set_listener** (p. 459), **com.rti.dds.subscription.DataReader.get_listener** (p. 460), **com.rti.dds.infrastructure.Entity.enable** (p. 1032), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034),
- **com.rti.dds.subscription.DataReader.get_liveliness_changed_status** (p. 461), **com.rti.dds.subscription.DataReader.get_requested_deadline_missed_status** (p. 461), **com.rti.dds.subscription.DataReader.get_requested_incompatible_qos_status** (p. 462), **com.rti.dds.subscription.DataReader.get_sample_lost_status** (p. 462), **com.rti.dds.subscription.DataReader.get_sample_rejected_status** (p. 460), **com.rti.dds.subscription.DataReader.get_subscription_matched_status** (p. 463)

All sample-accessing operations, namely: **com.rti.ndds.example.FooDataReader.read** (p. 1069), **com.rti.ndds.example.FooDataReader.take** (p. 1071), **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076), and **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077) may fail with the error **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) as described in **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749).

See also

Operations Allowed in Listener Callbacks (p. 1238)

Access to data samples (p. 78)

8.45.2 Member Function Documentation

8.45.2.1 create_readcondition()

```
ReadCondition create_readcondition (
            int sample_states,
            int view_states,
            int instance_states )
```

Creates a **com.rti.dds.subscription.ReadCondition** (p. 1514).

The returned **com.rti.dds.subscription.ReadCondition** (p. 1514) will be attached and belong to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

sample_states	<<***in***>> *(p. 156)* sample state of the data samples that are of interest
view_states	<<***in***>> *(p. 156)* view state of the data samples that are of interest
instance_states	<<***in***>> *(p. 156)* instance state of the data samples that are of interest

Returns

return **com.rti.dds.subscription.ReadCondition** (p. 1514) created. Returns NULL in case of failure.

8.45.2.2 create_readcondition_w_params()

```
ReadCondition create_readcondition_w_params (
            ReadConditionParams params )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.subscription.ReadCondition** (p. 1514) with parameters.

The returned **com.rti.dds.subscription.ReadCondition** (p. 1514) will be attached and belong to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

params	<<***in***>> *(p. 156)* creation parameters.

Returns

return **com.rti.dds.subscription.ReadCondition** (p. 1514) created. Returns NULL in case of failure.

8.45.2.3 create_querycondition()

```
QueryCondition create_querycondition (
            int sample_states,
            int view_states,
            int instance_states,
            String query_expression,
             StringSeq query_parameters )
```

Creates a **com.rti.dds.subscription.QueryCondition** (p. 1510).

The returned **com.rti.dds.subscription.QueryCondition** (p. 1510) will be attached and belong to the **com.rti.dds.subscription.DataReader** (p. 450).

Queries and Filters Syntax (p. 104) describes the syntax of `query_expression` and `query_parameters`.

Parameters

sample_states	<<***in***>> *(p. 156)* sample state of the data samples that are of interest
view_states	<<***in***>> *(p. 156)* view state of the data samples that are of interest
instance_states	<<***in***>> *(p. 156)* instance state of the data samples that are of interest
query_expression	<<***in***>> *(p. 156)* Expression for the query. Cannot be NULL.
query_parameters	<<***in***>> *(p. 156)* Parameters for the query expression. Cannot be NULL.

Returns

return **com.rti.dds.subscription.QueryCondition** (p. 1510) created. Returns NULL in case of failure.

8.45.2.4 create_querycondition_w_params()

```
QueryCondition create_querycondition_w_params (
            QueryConditionParams params )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.subscription.QueryCondition** (p. 1510) with parameters.

The returned **com.rti.dds.subscription.QueryCondition** (p. 1510) will be attached and belong to the **com.rti.dds.↩ subscription.DataReader** (p. 450).

Parameters

params	<<***in***>> *(p. 156)* creation parameters.

Returns

return **com.rti.dds.subscription.QueryCondition** (p. 1510) created. Returns NULL in case of failure.

8.45.2.5 delete_readcondition()

```
void delete_readcondition (
            ReadCondition condition )
```

Deletes a **com.rti.dds.subscription.ReadCondition** (p. 1514) or **com.rti.dds.subscription.QueryCondition** (p. 1510) attached to the **com.rti.dds.subscription.DataReader** (p. 450).

Since **com.rti.dds.subscription.QueryCondition** (p. 1510) specializes **com.rti.dds.subscription.ReadCondition** (p. 1514), it can also be used to delete a **com.rti.dds.subscription.QueryCondition** (p. 1510).

Precondition

The **com.rti.dds.subscription.ReadCondition** (p. 1514) must be attached to the **com.rti.dds.subscription.↩ DataReader** (p. 450), or the operation will fail with the error **com.rti.dds.infrastructure.RETCODE_↩ PRECONDITION_NOT_MET** (p. 1598).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

condition	*<<**in**>> (p. 156)* Condition to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

8.45.2.6 set_qos()

```
void set_qos (
              DataReaderQos qos )
```

Sets the reader QoS.

Modifies the QoS of the **com.rti.dds.subscription.DataReader** (p. 450).

The **com.rti.dds.subscription.DataReaderQos.user_data** (p. 523), **com.rti.dds.subscription.DataReaderQos.↩ deadline** (p. 521), **com.rti.dds.subscription.DataReaderQos.latency_budget** (p. 522), **com.rti.dds.subscription.↩ DataReaderQos.time_based_filter** (p. 523), **com.rti.dds.subscription.DataReaderQos.reader_data_lifecycle** (p. 523) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 156)* The **com.rti.dds.subscription.DataReaderQos** (p. 517) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.subscription.DataReader** (p. 450) is enabled. The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) can be used to indicate that the QoS of the **com.rti.dds.subscription.DataReader** (p. 450) should be changed to match the current default **com.rti.dds.subscription.DataReaderQos** (p. 517) set in the **com.rti.dds.subscription.Subscriber** (p. 1730). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

set_qos (abstract) (p. 1030)

com.rti.dds.subscription.DataReader.set_qos (p. 457)

Operations Allowed in Listener Callbacks (p. 1238)

8.45.2.7 set_qos_with_profile()

```
void set_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Changes the QoS of this reader using the input XML QoS profile.

This operation modifies the QoS of the **com.rti.dds.subscription.DataReader** (p. 450).

The **com.rti.dds.subscription.DataReaderQos.user_data** (p. 523), **com.rti.dds.subscription.DataReaderQos.↩ deadline** (p. 521), **com.rti.dds.subscription.DataReaderQos.latency_budget** (p. 522), **com.rti.dds.subscription.↩ DataReaderQos.time_based_filter** (p. 523), **com.rti.dds.subscription.DataReaderQos.reader_data_lifecycle** (p. 523) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1746)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

com.rti.dds.subscription.DataReader.set_qos (p. 457)

Operations Allowed in Listener Callbacks (p. 1238)

8.45.2.8 get_qos()

```
void get_qos (
              DataReaderQos qos )
```

Gets the reader QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 156)* The **com.rti.dds.subscription.DataReaderQos** (p. 517) to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.45.2.9 set_listener()

```
void set_listener (
              DataReaderListener l,
             int mask )
```

Sets the reader listener.

Parameters

l	<<***in***>> *(p. 156)* **com.rti.dds.subscription.DataReaderListener** (p. 497) to set to
mask	<<***in***>> *(p. 156)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.subscription.DataReaderListener** (p. 497).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.45.2.10 get_listener()

```
DataReaderListener get_listener ( )
```

Gets the reader listener.

Returns

com.rti.dds.subscription.DataReaderListener (p. 497) of the **com.rti.dds.subscription.DataReader** (p. 450).

See also

get_listener (abstract) (p. 1032)

8.45.2.11 call_listenerT()

```
void call_listenerT (
             int mask )
```

Calls the reader listener.

See also

get_listener (abstract) (p. 1032)

8.45.2.12 get_sample_rejected_status()

```
void get_sample_rejected_status (
              SampleRejectedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.SampleRejectedStatus** (p. 1657) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.13 get_liveliness_changed_status()

```
void get_liveliness_changed_status (
              LivelinessChangedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1239) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.14 get_requested_deadline_missed_status()

```
void get_requested_deadline_missed_status (
              RequestedDeadlineMissedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.RequestedDeadlineMissedStatus** (p. 1560) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.15 get_requested_incompatible_qos_status()

```
void get_requested_incompatible_qos_status (
            RequestedIncompatibleQosStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.RequestedIncompatibleQosStatus** (p. 1561) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.16 get_sample_lost_status()

```
void get_sample_lost_status (
            SampleLostStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.SampleLostStatus** (p. 1648) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.17 get_subscription_matched_status()

```
void get_subscription_matched_status (
            SubscriptionMatchedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.18 get_datareader_cache_status()

```
void get_datareader_cache_status (
            DataReaderCacheStatus status )
```

<<***extension***>> *(p. 155)* Gets the datareader cache status for this reader.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.DataReaderCacheStatus** (p. 488) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.45.2.19 get_datareader_protocol_status()

```
void get_datareader_protocol_status (
            DataReaderProtocolStatus status )
```

<<***extension***>> *(*p. *155)* Gets the datareader protocol status for this reader.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.DataReaderProtocolStatus** (p. 505) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.45.2.20 get_matched_publication_datareader_protocol_status()

```
void get_matched_publication_datareader_protocol_status (
                DataReaderProtocolStatus status,
                InstanceHandle_t publication_handle )
```

<<***extension***>> *(p. 155)* Gets the datareader protocol status for this reader, per matched publication identified by the publication_handle.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	<<***inout***>> *(p. 156)*. The information to be filled in on the associated publication. Cannot be NULL.
publication_handle	<<***in***>> *(p. 156)*. Handle to a specific publication associated with the **com.rti.dds.publication.DataWriter** (p. 553). Cannot be NULL. . Must correspond to a publication currently associated with the **com.rti.dds.subscription.DataReader** (p. 450).

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.45.2.21 get_matched_publications()

```
void get_matched_publications (
                InstanceHandleSeq publication_handles )
```

Retrieves the list of publications currently "associated" with this **com.rti.dds.subscription.DataReader** (p. 450).

A publication is considered to be matching if all of the following criteria are true:

- The publication is within the same domain as this subscription.
- The publication has a matching **com.rti.dds.topic.Topic** (p. 1807).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **com.rti.dds.domain.DomainParticipant** (p. 670) has not indicated that the publication's **com.rti.dds.↩ domain.DomainParticipant** (p. 670) should be "ignored" by means of the **com.rti.dds.domain.Domain↩ Participant.ignore_publication** (p. 725) API.
- If the subscription is using a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) and the publication is using the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the reader has completed the key exchange with the writer.

The handles returned in the `publication_handles` list are the ones that are used by the RTI Connext implementation to locally identify the corresponding matched **com.rti.dds.publication.DataWriter** (p. 553) entities. These handles match the ones that appear in the `instance_handle` field of the **com.rti.dds.subscription.SampleInfo** (p. 1634) when reading the **com.rti.dds.publication.builtin.PublicationBuiltinTopicDataTypeSupport.PUBLICATION_↩ TOPIC_NAME** (p. 163) builtin topic.

This API may return the publication handles of publications that are not alive. **com.rti.dds.subscription.DataReader.↩ is_matched_publication_alive** (p. 467) can be used to check the liveliness of the remote publication.

Parameters

publication_handles	*<<**inout**>> (p. 156).* The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all of the matches and the system cannot resize the sequence, this method will fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). The maximum number of matches possible is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 803). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) if the sequence is too small and the system cannot resize it, or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.45.2.22 get_matched_publication_data()

```
void get_matched_publication_data (
               PublicationBuiltinTopicData publication_data,
               InstanceHandle_t publication_handle )
```

Retrieves the information on a publication that is currently "associated" with the **com.rti.dds.subscription.DataReader** (p. 450).

The `publication_handle` must correspond to a publication currently associated with the **com.rti.dds.subscription.DataReader** (p. 450). Otherwise, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). Use the operation **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 465) to find the publications that are currently matched with the **com.rti.dds.subscription.DataReader** (p. 450).

Note: This operation does not retrieve the **builtin.PublicationBuiltinTopicData.type_code** (p. 1457). This information is available through **com.rti.dds.subscription.DataReaderListener::on_data_available()** (p. 500) (if a reader listener is installed on the builtin.PublicationBuiltinTopicDataDataReader).

Parameters

publication_data	*<<**inout**>> (p. 156).* The information to be filled in on the associated publication. Cannot be NULL.
publication_handle	*<<**in**>> (p. 156).* Handle to a specific publication associated with the **com.rti.dds.publication.DataWriter** (p. 553). Must correspond to a publication currently associated with the **com.rti.dds.subscription.DataReader** (p. 450). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.45.2.23 is_matched_publication_alive()

```
boolean is_matched_publication_alive (
               InstanceHandle_t publication_handle )
```

Check if a publication currently matched with a **DataReader** (p. 450) is alive.

This API is used for querying the endpoint liveliness of a matched publication. A matched publication will be marked as not alive if the liveliness that it committed to through its **LIVELINESS** (p. 239) QoS policy was not respected. Note that if the participant associated with the matched publication loses liveliness, the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) will become invalid and this function will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Parameters

publication_handle	*<<**in**>> (p. 156)* The **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of the matched publication. See **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 465) for a description of what is considered a matched publication.

Returns

A boolean indicating whether or not the matched publication is active.

8.45.2.24 get_matched_publication_participant_data()

```
void get_matched_publication_participant_data (
            ParticipantBuiltinTopicData participant_data,
            InstanceHandle_t publication_handle )
```

<<***extension***>> *(p. 155)* Retrieves the information on the discovered **com.rti.dds.domain.DomainParticipant** (p. 670) associated with the publication that is currently matching with the **com.rti.dds.subscription.DataReader** (p. 450).

The `publication_handle` must correspond to a publication currently associated with the **com.rti.dds.subscription.DataReader** (p. 450). Otherwise, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). The operation may also fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if the publication corresponds to the same **com.rti.dds.domain.DomainParticipant** (p. 670) that the **DataReader** (p. 450) belongs to. Use the operation **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 465) to find the publications that are currently matched with the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

participant_data	<<***inout***>> *(p. 156)*. The information to be filled in on the associated participant. Cannot be NULL.
publication_handle	<<***in***>> *(p. 156)*. Handle to a specific publication associated with a **com.rti.dds.publication.DataWriter** (p. 553). Must correspond to a publication currently associated with the **com.rti.dds.subscription.DataReader** (p. 450). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.45.2.25 get_topicdescription()

```
TopicDescription get_topicdescription ( )
```

Returns the **com.rti.dds.topic.TopicDescription** (p. 1820) associated with the **com.rti.dds.subscription.DataReader** (p. 450).

Returns that same **com.rti.dds.topic.TopicDescription** (p. 1820) that was used to create the **com.rti.dds.subscription.DataReader** (p. 450).

Returns

com.rti.dds.topic.TopicDescription (p. 1820) associated with the **com.rti.dds.subscription.DataReader** (p. 450).

8.45.2.26 get_subscriber()

```
Subscriber get_subscriber ( )
```

Returns the **com.rti.dds.subscription.Subscriber** (p. 1730) to which the **com.rti.dds.subscription.DataReader** (p. 450) belongs.

Returns

com.rti.dds.subscription.Subscriber (p. 1730) to which the **com.rti.dds.subscription.DataReader** (p. 450) belongs.

8.45.2.27 delete_contained_entities()

```
void delete_contained_entities ( )
```

Deletes all the entities that were created by means of the "create" operations on the **com.rti.dds.subscription.DataReader** (p. 450).

Deletes all contained **com.rti.dds.subscription.ReadCondition** (p. 1514), **com.rti.dds.subscription.QueryCondition** (p. 1510), and **com.rti.dds.subscription.TopicQuery** (p. 1830) objects.

The operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if the any of the contained entities is in a state where it cannot be deleted.

Once **com.rti.dds.subscription.DataReader.delete_contained_entities** (p. 469) completes successfully, the application may delete the **com.rti.dds.subscription.DataReader** (p. 450).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

8.45.2.28 wait_for_historical_data()

```
void wait_for_historical_data (
            Duration_t max_wait )
```

Waits until all "historical" data is received for **com.rti.dds.subscription.DataReader** (p. 450) entities that have a non-↩ VOLATILE Durability Qos kind.

This operation is intended only for **com.rti.dds.subscription.DataReader** (p. 450) entities that have a non-VOLATILE Durability QoS kind.

As soon as an application enables a non-VOLATILE **com.rti.dds.subscription.DataReader** (p. 450), it will start receiving both "historical" data (i.e., the data that was written prior to the time the **com.rti.dds.subscription.DataReader** (p. 450) joined the domain) as well as any new data written by the **com.rti.dds.publication.DataWriter** (p. 553) entities. There are situations where the application logic may require the application to wait until all "historical" data is received. This is the purpose of the **com.rti.dds.subscription.DataReader.wait_for_historical_data** (p. 470) operations.

com.rti.dds.subscription.DataReader.wait_for_historical_data() (p. 470) blocks the calling thread until either all "historical" data is received, or until the duration specified by the max_wait parameter elapses, whichever happens first. It will return immediately if no DataWriters have been discovered at the time the operation is called; therefore it is advisable to make sure at least one **com.rti.dds.publication.DataWriter** (p. 553) has been discovered before calling this operation. (One way to do this is by using **com.rti.dds.subscription.DataReader.get_subscription_matched_status** (p. 463).)

A successful completion indicates that all the "historical" data was "received"; timing out indicates that max_wait elapsed before all the data was received.

Parameters

max_wait	<<***in***>> *(p. 156)* Timeout value. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.45.2.29 acknowledge_sample() [1/2]

```
void acknowledge_sample (
            SampleInfo sample_info )
```

<<***extension***>> *(p. 155)* Acknowledges a single sample explicitly.

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 450) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.samples_per_app_ack** (p. 1604) and **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.app_ack_period** (p. 1603).

Parameters

sample_info	<<***in***>> *(p. 156)* **com.rti.dds.subscription.SampleInfo** (p. 1634) identifying the sample being acknowledged.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.30 acknowledge_all() [1/2]

```
void acknowledge_all ( )
```

<<***extension***>> *(p. 155)* Acknowledges all previously accessed samples.

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 450) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.samples_per_app_ack** (p. 1604) and **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.app_ack_period** (p. 1603).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.31 acknowledge_sample() [2/2]

```
void acknowledge_sample (
             SampleInfo sample_info,
             AckResponseData_t response_data )
```

<<***extension***>> *(p. 155)* Acknowledges a single sample explicitly.

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 450) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.samples_per_app_ack** (p. 1604) and **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.app_ack_period** (p. 1603).

The maximum length of the response is configured using **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_app_ack_response_length** (p. 539)

Parameters

sample_info	<<***in***>> *(p. 156)* **com.rti.dds.subscription.SampleInfo** (p. 1634) identifying the sample being acknowledged.
response_data	<<***in***>> *(p. 156)* Response data sent to **com.rti.dds.publication.DataWriter** (p. 553) upon acknowledgment (via **com.rti.dds.publication.DataWriterListener.on_application_acknowledgment** (p. 595))

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.32 acknowledge_all() [2/2]

```
void acknowledge_all (
            AckResponseData_t response_data )
```

<<***extension***>> *(p. 155)* Acknowledges all previously accessed samples.

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

A call to this method does not necessarily trigger the sending of an AppAck RTPS message from the **DataReader** (p. 450) to the DataWriter. How and when AppAck messages are sent can be configured using the QoS values **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.samples_per_app_ack** (p. 1604) and **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.app_ack_period** (p. 1603).

The maximum length of the response is configured using **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_app_ack_response_length** (p. 539).

Parameters

response_data	<<***in***>> *(*p. *156)* Response data sent to **com.rti.dds.publication.DataWriter** (p. 553) upon acknowledgment

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.33 create_topic_query()

```
TopicQuery create_topic_query (
            TopicQuerySelection selection )
```

Creates a **com.rti.dds.subscription.TopicQuery** (p. 1830).

The returned **com.rti.dds.subscription.TopicQuery** (p. 1830) will have been issued if the **com.rti.dds.subscription.DataReader** (p. 450) is enabled. Otherwise, the **com.rti.dds.subscription.TopicQuery** (p. 1830) will be issued once the **com.rti.dds.subscription.DataReader** (p. 450) is enabled

Parameters

selection	<<***in***>> *(*p. *156)* The selection with which to create the **com.rti.dds.subscription.TopicQuery** (p. 1830). The special values **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 87) and **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 87) can be used. The expression can start with the special condition "@instance_state = ALIVE AND" followed by the rest of the expression. This restricts the selection to samples of alive instances. Cannot be NULL.

Returns

return Created **com.rti.dds.subscription.TopicQuery** (p. 1830). Returns NULL in case of failure.

8.45.2.34 delete_topic_query()

```
void delete_topic_query (
            TopicQuery query )
```

Deletes a **com.rti.dds.subscription.TopicQuery** (p. 1830).

Cancels an active **com.rti.dds.subscription.TopicQuery** (p. 1830). After deleting a **TopicQuery** (p. 1830), new DataWriters won't discover it and existing DataWriters currently publishing cached samples may stop before delivering all of them.

Parameters

query	<<***in***>> *(*p. *156)* The **com.rti.dds.subscription.TopicQuery** (p. 1830) to delete. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.45.2.35 lookup_topic_query()

```
TopicQuery lookup_topic_query (
             GUID_t guid )
```

Retrieves an existing **com.rti.dds.subscription.TopicQuery** (p. 1830).

Retrieves the **com.rti.dds.subscription.TopicQuery** (p. 1830) that corresponds to the input **com.rti.dds.infrastructure.GUID_t** (p. 1132).

If no **TopicQuery** (p. 1830) is found for the specified GUID or the **TopicQuery** (p. 1830) is marked for deletion, this returns NULL.

To get the **com.rti.dds.infrastructure.GUID_t** (p. 1132) associated with a **com.rti.dds.subscription.TopicQuery** (p. 1830), use the method **com.rti.dds.subscription.TopicQuery.get_guid** (p. 1830).

Parameters

guid	<<***in***>> *(*p. *156)* The **com.rti.dds.subscription.TopicQuery** (p. 1830) GUID. Cannot be NULL.

8.45.2.36 read_untyped()

```
void read_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read** (p. 1069) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_untyped (p. 475)

com.rti.ndds.example.FooDataReader.read (p. 1069)

Referenced by **BytesDataReader.read()**, **DynamicDataReader.read()**, **FooDataReader.read()**, **KeyedBytesData↩ Reader.read()**, **KeyedStringDataReader.read()**, and **StringDataReader.read()**.

8.45.2.37 take_untyped()

```
void take_untyped (
            List<?> received_data,
             SampleInfoSeq info_seq,
            int max_samples,
            int sample_states,
            int view_states,
            int instance_states )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take** (p. 1071) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_untyped (p. 474)

com.rti.ndds.example.FooDataReader.take (p. 1071)

Referenced by **BytesDataReader.take()**, **DynamicDataReader.take()**, **FooDataReader.take()**, **KeyedBytesData↩ Reader.take()**, **KeyedStringDataReader.take()**, and **StringDataReader.take()**.

8.45.2.38 read_w_condition_untyped()

```
void read_w_condition_untyped (
            List<?> received_data,
             SampleInfoSeq info_seq,
            int max_samples,
             ReadCondition read_condition )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_w_condition_untyped (p. 476)

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076)

Referenced by **BytesDataReader.read_w_condition()**, **DynamicDataReader.read_w_condition()**, **FooDataReader.read_w_condition()**, **KeyedBytesDataReader.read_w_condition()**, **KeyedStringDataReader.read_w_condition()**, and **StringDataReader.read_w_condition()**.

8.45.2.39 take_w_condition_untyped()

```
void take_w_condition_untyped (
            List<?> received_data,
             SampleInfoSeq info_seq,
            int max_samples,
             ReadCondition read_condition )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_w_condition_untyped (p. 475)

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

Referenced by **BytesDataReader.take_w_condition()**, **DynamicDataReader.take_w_condition()**, **FooDataReader.take_w_condition()**, **KeyedBytesDataReader.take_w_condition()**, **KeyedStringDataReader.take_w_condition()**, and **StringDataReader.take_w_condition()**.

8.45.2.40 read_next_sample_untyped()

```
void read_next_sample_untyped (
            Object received_data,
             SampleInfo sample_info )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_next_sample_untyped (p. 477)

com.rti.ndds.example.FooDataReader.read_next_sample (p. 1078)

Referenced by **BytesDataReader.read_next_sample()**, **DynamicDataReader.read_next_sample()**, **FooDataReader.read_next_sample()**, **KeyedBytesDataReader.read_next_sample()**, **KeyedStringDataReader.read_next_sample()**, and **StringDataReader.read_next_sample()**.

8.45.2.41 take_next_sample_untyped()

```
void take_next_sample_untyped (
            Object received_data,
             SampleInfo sample_info )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_next_sample_untyped (p. 476)

com.rti.ndds.example.FooDataReader.take_next_sample (p. 1079)

Referenced by **BytesDataReader.take_next_sample()**, **DynamicDataReader.take_next_sample()**, **FooDataReader.take_next_sample()**, **KeyedBytesDataReader.take_next_sample()**, **KeyedStringDataReader.take_next_sample()**, and **StringDataReader.take_next_sample()**.

8.45.2.42 read_instance_untyped() [1/2]

```
void read_instance_untyped (
            List<?> received_data,
             SampleInfoSeq info_seq,
            int max_samples,
             InstanceHandle_t a_handle,
            int sample_states,
            int view_states,
            int instance_states )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_instance_untyped (p. 478)

com.rti.ndds.example.FooDataReader.read_instance (p. 1081)

Referenced by **DynamicDataReader.read_instance()**, **FooDataReader.read_instance()**, **KeyedBytesData↩ Reader.read_instance()**, and **KeyedStringDataReader.read_instance()**.

8.45.2.43 take_instance_untyped() [1/2]

```
void take_instance_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_instance_untyped (p. 477)

com.rti.ndds.example.FooDataReader.take_instance (p. 1082)

Referenced by **DynamicDataReader.take_instance()**, **FooDataReader.take_instance()**, **KeyedBytesData↩ Reader.take_instance()**, and **KeyedStringDataReader.take_instance()**.

8.45.2.44 read_instance_untyped() [2/2]

```
void read_instance_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              GUID_t topic_query_guid,
             int sample_states,
             int view_states,
             int instance_states )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_instance_untyped (p. 478)

com.rti.ndds.example.FooDataReader.read_instance (p. 1081)

8.45.2.45 take_instance_untyped() [2/2]

```
void take_instance_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              GUID_t topic_query_guid,
             int sample_states,
             int view_states,
             int instance_states )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_instance_untyped (p. 477)

com.rti.ndds.example.FooDataReader.take_instance (p. 1082)

8.45.2.46 read_instance_w_condition_untyped()

```
void read_instance_w_condition_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition read_condition )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_next_instance_↩ w_condition** (p. 1091) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_next_instance_w_condition_untyped (p. 482)

com.rti.ndds.example.FooDataReader.read_next_instance_w_condition (p. 1091)

Referenced by **DynamicDataReader.read_instance_w_condition()**, **FooDataReader.read_instance_w_↩ condition()**, **KeyedBytesDataReader.read_instance_w_condition()**, and **KeyedStringDataReader.read_↩ instance_w_condition()**.

8.45.2.47 take_instance_w_condition_untyped()

```
void take_instance_w_condition_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition read_condition )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_next_instance_↩ w_condition** (p. 1092) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_next_instance_w_condition_untyped (p. 481)

com.rti.ndds.example.FooDataReader.take_next_instance_w_condition (p. 1092)

Referenced by **DynamicDataReader.take_instance_w_condition()**, **FooDataReader.take_instance_w_↩ condition()**, **KeyedBytesDataReader.take_instance_w_condition()**, and **KeyedStringDataReader.take_↩ instance_w_condition()**.

8.45.2.48 read_next_instance_untyped()

```
void read_next_instance_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_next_instance_untyped (p. 481)

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

Referenced by **DynamicDataReader.read_next_instance()**, **FooDataReader.read_next_instance()**, **KeyedBytesDataReader.read_next_instance()**, and **KeyedStringDataReader.read_next_instance()**.

8.45.2.49 take_next_instance_untyped()

```
void take_next_instance_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_next_instance_untyped (p. 480)

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

Referenced by **DynamicDataReader.take_next_instance()**, **FooDataReader.take_next_instance()**, **KeyedBytesDataReader.take_next_instance()**, and **KeyedStringDataReader.take_next_instance()**.

8.45.2.50 read_next_instance_w_condition_untyped()

```
void read_next_instance_w_condition_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition read_condition )
```

Reads data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.take_next_instance_w_condition_untyped (p. 482)

com.rti.ndds.example.FooDataReader.read_next_instance_w_condition (p. 1091)

Referenced by **DynamicDataReader.read_next_instance_w_condition()**, **FooDataReader.read_next_instance_w_condition()**, **KeyedBytesDataReader.read_next_instance_w_condition()**, and **KeyedStringDataReader.read_next_instance_w_condition()**.

8.45.2.51 take_next_instance_w_condition_untyped()

```
void take_next_instance_w_condition_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition read_condition )
```

Takes data samples, if any are available.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.subscription.DataReader.read_next_instance_w_condition_untyped (p. 481)

com.rti.ndds.example.FooDataReader.take_next_instance_w_condition (p. 1092)

Referenced by **DynamicDataReader.take_next_instance_w_condition()**, **FooDataReader.take_next_instance_w_condition()**, **KeyedBytesDataReader.take_next_instance_w_condition()**, and **KeyedStringDataReader.take_next_instance_w_condition()**.

8.45.2.52 return_loan_untyped()

```
void return_loan_untyped (
             List<?> received_data,
              SampleInfoSeq info_seq )
```

Returns loaned sample data and meta-data.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) method instead of this one. See that method for detailed documentation.

See also

com.rti.ndds.example.FooDataReader.return_loan (p. 1094)

Referenced by **BytesDataReader.return_loan()**, **DynamicDataReader.return_loan()**, **FooDataReader.return_loan()**, **KeyedBytesDataReader.return_loan()**, and **KeyedStringDataReader.return_loan()**.

8.45.2.53 get_key_value_untyped()

```
void get_key_value_untyped (
             Object key_holder,
              InstanceHandle_t handle )
```

Fills in the key fields of the given data sample.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataReader** (p. 1067) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataReader.get_key_value** (p. 1095) method instead of this one. See that method for detailed documentation.

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

Referenced by **DynamicDataReader.get_key_value()**, **FooDataReader.get_key_value()**, **KeyedBytesData↩ Reader.get_key_value()**, and **KeyedStringDataReader.get_key_value()**.

8.45.2.54 lookup_instance_untyped()

```
InstanceHandle_t lookup_instance_untyped (
             Object key_value )
```

Referenced by **DynamicDataReader.lookup_instance()**, **FooDataReader.lookup_instance()**, **KeyedBytesData↩ Reader.lookup_instance()**, and **KeyedStringDataReader.lookup_instance()**.

8.45.2.55 take_discovery_snapshot() [1/2]

```
void take_discovery_snapshot ( )
```

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

The snapshot will be printed through the **com.rti.ndds.config.Logger** (p. 1267). A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"
```

```
unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

---------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 261).

8.45.2.56 take_discovery_snapshot() [2/2]

```
void take_discovery_snapshot (
            String file_name )
```

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote writers that match the local reader domain=0 name="readerTestName"

guid="0x0101542A,0x2C59B595,0xA1693BDF:0x80000004"

topic="FooTopic" type="FooType"

---------------------------------------------------------------------------

Compatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003 name="writer1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

---------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(p. 156)* Name of the file where snapshot should be printed.

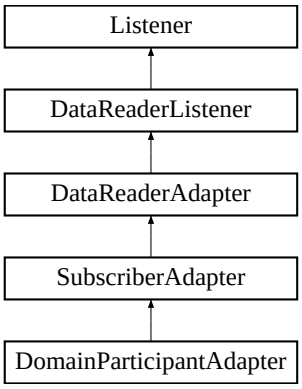
Exceptions

One	of the **Standard Return Codes** (p. 261).

8.46 DataReaderAdapter Class Reference

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Inheritance diagram for DataReaderAdapter:



Public Member Functions

- void **on_requested_deadline_missed** (**DataReader** reader, **RequestedDeadlineMissedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.
- void **on_requested_incompatible_qos** (**DataReader** reader, **RequestedIncompatibleQosStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.
- void **on_sample_rejected** (**DataReader** reader, **SampleRejectedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.
- void **on_liveliness_changed** (**DataReader** reader, **LivelinessChangedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.
- void **on_data_available** (**DataReader** reader)

  Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS communication status.
- void **on_sample_lost** (**DataReader** reader, **SampleLostStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.
- void **on_subscription_matched** (**DataReader** reader, **SubscriptionMatchedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

8.46.1 Detailed Description

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.46.2 Member Function Documentation

8.46.2.1 on_requested_deadline_missed()

```
void on_requested_deadline_missed (
              DataReader reader,
              RequestedDeadlineMissedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

Implements **DataReaderListener** (p. 499).

8.46.2.2 on_requested_incompatible_qos()

```
void on_requested_incompatible_qos (
              DataReader reader,
              RequestedIncompatibleQosStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Implements **DataReaderListener** (p. 499).

8.46.2.3 on_sample_rejected()

```
void on_sample_rejected (
              DataReader reader,
              SampleRejectedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.

Implements **DataReaderListener** (p. 499).

8.46.2.4 on_liveliness_changed()

```
void on_liveliness_changed (
              DataReader reader,
              LivelinessChangedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.

Implements **DataReaderListener** (p. 499).

8.46.2.5 on_data_available()

```
void on_data_available (
              DataReader reader )
```

Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS communication status.

Implements **DataReaderListener** (p. 500).

8.46.2.6 on_sample_lost()

```
void on_sample_lost (
              DataReader reader,
              SampleLostStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

Implements **DataReaderListener** (p. 500).

8.46.2.7 on_subscription_matched()

```
void on_subscription_matched (
              DataReader reader,
              SubscriptionMatchedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Implements **DataReaderListener** (p. 500).

8.47 DataReaderCacheStatus Class Reference

<<***extension***>> *(*p. *155)* The status of the reader's cache.

Inherits Status.

Public Attributes

- long **sample_count_peak**

  The highest number of samples in the reader's queue over the lifetime of the reader.
- long **sample_count**

  The number of samples in the reader's queue.
- long **old_source_timestamp_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample older than the last one, using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.
- long **tolerance_source_timestamp_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample in the future, using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.
- long **ownership_dropped_sample_count**

  The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.
- long **content_filter_dropped_sample_count**

  The number of user samples filtered by the ***DataReader*** *(*p. *450) due to Content-Filtered Topics.*
- long **time_based_filter_dropped_sample_count**

  The number of user samples filtered by the ***DataReader*** *(*p. *450) due to* ***com.rti.dds.infrastructure.TimeBasedFilterQosPolicy*** *(*p. *1804).*
- long **expired_dropped_sample_count**

  The number of samples expired by the ***DataReader*** *(*p. *450) due to* ***com.rti.dds.infrastructure.LifespanQosPolicy*** *(*p. *1234) or the autopurge sample delays.*
- long **virtual_duplicate_dropped_sample_count**

  The number of virtual duplicate samples dropped by the ***DataReader*** *(*p. *450). A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.*
- long **replaced_dropped_sample_count**

  The number of samples replaced by the ***DataReader*** *(*p. *450) due to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS replacement.*
- long **writer_removed_batch_sample_dropped_sample_count**

  The number of batch samples received by the ***DataReader*** *(*p. *450) that were marked as removed by the DataWriter.*
- long **total_samples_dropped_by_instance_replacement**

  The number of samples with sample state com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE that were dropped when removing an instance due to instance replacement. See ***com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.instance_replacement*** *(*p. *540) for more details about when instances are replaced.*
- long **alive_instance_count**

  The number of instances in the ***DataReader*** *(*p. *450)'s queue with an instance state equal to* ***com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE*** *(*p. *1161).*
- long **alive_instance_count_peak**

*The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.alive_instance_count** (p. 492) over the lifetime of the **DataReader** (p. 450).*

- long **no_writers_instance_count**

  *The number of instances in the **DataReader** (p. 450)'s queue with an instance state equal to **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).*

- long **no_writers_instance_count_peak**

  *The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.no_writers_instance_count** (p. 493) over the lifetime of the **DataReader** (p. 450).*

- long **disposed_instance_count**

  *The number of instances in the **DataReader** (p. 450)'s queue with an instance state equal to **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).*

- long **disposed_instance_count_peak**

  *The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.disposed_instance_count** (p. 493) over the lifetime of the **DataReader** (p. 450).*

- long **detached_instance_count**

  *The number of minimal instance states currently being maintained in the **DataReader** (p. 450)'s queue.*

- long **detached_instance_count_peak**

  *The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.detached_instance_count** (p. 494) over the lifetime of the **DataReader** (p. 450).*

- long **compressed_sample_count**

  The number of received samples compressed by a DataWriter.

8.47.1 Detailed Description

<<***extension***>> *(p. 155)* The status of the reader's cache.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

8.47.2 Member Data Documentation

8.47.2.1 sample_count_peak

```
long sample_count_peak
```

The highest number of samples in the reader's queue over the lifetime of the reader.

8.47.2.2 sample_count

```
long sample_count
```

The number of samples in the reader's queue.

includes samples that may not yet be available to be read or taken by the user, due to samples being received out of order or **PRESENTATION** (p. 247)

8.47.2.3 old_source_timestamp_dropped_sample_count

```
long old_source_timestamp_dropped_sample_count
```

The number of samples dropped as a result of receiving a sample older than the last one, using com.rti.↩ dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_↩ DESTINATIONORDER_QOS.

When the **DataReader** (p. 450) is using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrder↩ QosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS:

- If the **DataReader** (p. 450) receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped.
- If the **DataReader** (p. 450) receives a sample for an instance with a source timestamp that is equal to the last source timestamp received for the instance and the writer has a higher virtual GUID, the sample is dropped.

8.47.2.4 tolerance_source_timestamp_dropped_sample_count

```
long tolerance_source_timestamp_dropped_sample_count
```

The number of samples dropped as a result of receiving a sample in the future, using com.rti.dds.infrastructure.↩ DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER↩ _QOS.

When the **DataReader** (p. 450) is using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrder↩ QosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS: the **DataReader** (p. 450) will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than the source_timestamp_↩ tolerance. Otherwise, the sample is dropped.

8.47.2.5 ownership_dropped_sample_count

```
long ownership_dropped_sample_count
```

The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.

When using Exclusive Ownership, the **DataReader** (p. 450) receives data from multiple DataWriters. Each instance can only be owned by one DataWriter.

If other DataWriters write samples on this instance, the samples will be dropped.

8.47.2.6 content_filter_dropped_sample_count

```
long content_filter_dropped_sample_count
```

The number of user samples filtered by the **DataReader** (p. 450) due to Content-Filtered Topics.

When using a content filter on the **DataReader** (p. 450) side, if the sample received by the **DataReader** (p. 450) does not pass the filter, it will be dropped.

8.47.2.7 time_based_filter_dropped_sample_count

```
long time_based_filter_dropped_sample_count
```

The number of user samples filtered by the **DataReader** (p. 450) due to **com.rti.dds.infrastructure.TimeBasedFilter↩ QosPolicy** (p. 1804).

When using **TIME_BASED_FILTER** (p. 267) on the **DataReader** (p. 450) side, if the sample received by the **DataReader** (p. 450) does not pass the minimum_separation filter, it will be dropped.

8.47.2.8 expired_dropped_sample_count

```
long expired_dropped_sample_count
```

The number of samples expired by the **DataReader** (p. 450) due to **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234) or the autopurge sample delays.

- **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234)
  When a sample expires due to **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234), the data is removed from the **DataReader** (p. 450) caches. This sample will be considered dropped if its **com.rti.dds.subscription.↩ SampleStateKind** (p. 1663) was com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_↩ SAMPLE_STATE.
- **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_nowriter_samples_delay** (p. 1522)
  When a sample expires due to **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_↩ nowriter_samples_delay** (p. 1522), this sample will be considered dropped if its **com.rti.dds.subscription.↩ SampleStateKind** (p. 1663) was com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_↩ SAMPLE_STATE.
- **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_disposed_samples_delay** (p. 1522)
  When a sample expires due to **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_↩ disposed_samples_delay** (p. 1522), this sample will be considered dropped if its **com.rti.dds.subscription.↩ SampleStateKind** (p. 1663) was com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_↩ SAMPLE_STATE.

8.47.2.9 virtual_duplicate_dropped_sample_count

```
long virtual_duplicate_dropped_sample_count
```

The number of virtual duplicate samples dropped by the **DataReader** (p. 450). A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.

When two DataWriters with the same logical data source publish a sample with the same sequence_number, one sample will be dropped and the other will be received by the **DataReader** (p. 450).

This can happen when multiple writers are writing on behalf of the same original DataWriter: for example, in systems with redundant Routing Services or when a **DataReader** (p. 450) is receiving samples both directly from the original DataWriter and from an instance of Persistence Service.

8.47.2.10 replaced_dropped_sample_count

```
long replaced_dropped_sample_count
```

The number of samples replaced by the **DataReader** (p. 450) due to com.rti.dds.infrastructure.HistoryQosPolicyKind.↩ HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS replacement.

When the number of samples for an instance in the queue reaches the **com.rti.dds.infrastructure.HistoryQos**↩ **Policy.depth** (p. 1147) value, a new sample for the instance will replace the oldest sample for the instance in the queue.

The new sample will be accepted and the old sample will be dropped.

This counter will only be updated if the replaced sample's **com.rti.dds.subscription.SampleStateKind** (p. 1663) was com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE.

8.47.2.11 writer_removed_batch_sample_dropped_sample_count

```
long writer_removed_batch_sample_dropped_sample_count
```

The number of batch samples received by the **DataReader** (p. 450) that were marked as removed by the DataWriter.

When the **DataReader** (p. 450) receives a batch, the batch can contain samples marked as removed by the DataWriter. Examples of removed samples in a batch could be because of sample replacement due to com.rti.dds.infrastructure.↩ HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS **com.rti.dds.infrastructure.HistoryQos**↩ **Policy** (p. 1144) QoS on the DataWriter or because the duration in **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234) was reached. By default, any sample marked as removed from a batch is dropped (unless you set the dds.↩ data_reader.accept_writer_removed_batch_samples property in the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) to true). Note: Historical data with removed batch samples written before the **DataReader** (p. 450) joined the DDS domain will also be included in the count.

8.47.2.12 total_samples_dropped_by_instance_replacement

```
long total_samples_dropped_by_instance_replacement
```

The number of samples with sample state com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ↩ _SAMPLE_STATE that were dropped when removing an instance due to instance replacement. See **com.rti.dds.**↩ **infrastructure.DataReaderResourceLimitsQosPolicy.instance_replacement** (p. 540) for more details about when instances are replaced.

8.47.2.13 alive_instance_count

```
long alive_instance_count
```

The number of instances in the **DataReader** (p. 450)'s queue with an instance state equal to **com.rti.dds.↩ subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161).

8.47.2.14 alive_instance_count_peak

```
long alive_instance_count_peak
```

The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.alive_instance_count** (p. 492) over the lifetime of the **DataReader** (p. 450).

See also

com.rti.dds.subscription.DataReaderCacheStatus.alive_instance_count (p. 492)

8.47.2.15 no_writers_instance_count

```
long no_writers_instance_count
```

The number of instances in the **DataReader** (p. 450)'s queue with an instance state equal to **com.rti.dds.↩ subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).

8.47.2.16 no_writers_instance_count_peak

```
long no_writers_instance_count_peak
```

The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.no_writers_instance_count** (p. 493) over the lifetime of the **DataReader** (p. 450).

See also

com.rti.dds.subscription.DataReaderCacheStatus.no_writers_instance_count (p. 493)

8.47.2.17 disposed_instance_count

```
long disposed_instance_count
```

The number of instances in the **DataReader** (p. 450)'s queue with an instance state equal to **com.rti.dds.↩ subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).

8.47.2.18 disposed_instance_count_peak

```
long disposed_instance_count_peak
```

The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.disposed_instance_count** (p. 493) over the lifetime of the **DataReader** (p. 450).

See also

com.rti.dds.subscription.DataReaderCacheStatus.disposed_instance_count (p. 493)

8.47.2.19 detached_instance_count

```
long detached_instance_count
```

The number of minimal instance states currently being maintained in the **DataReader** (p. 450)'s queue.

If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.keep_minimum_state_for_instances** (p. 539) is true, the **DataReader** (p. 450) will keep up to a maximum of **com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.max_total_instances** (p. 536) detached instances in its queue. For a more in-depth description of detached instances, refer to **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_total_instances** (p. 536).

8.47.2.20 detached_instance_count_peak

```
long detached_instance_count_peak
```

The highest value of **com.rti.dds.subscription.DataReaderCacheStatus.detached_instance_count** (p. 494) over the lifetime of the **DataReader** (p. 450).

See also

com.rti.dds.subscription.DataReaderCacheStatus.detached_instance_count (p. 494)

8.47.2.21 compressed_sample_count

```
long compressed_sample_count
```

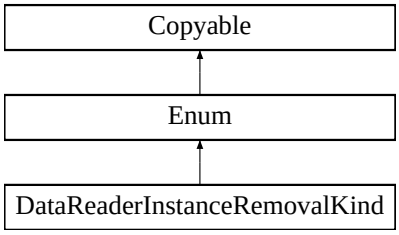
The number of received samples compressed by a DataWriter.

These include data, dispose, and unregister samples sent by a DataWriter.

8.48 DataReaderInstanceRemovalKind Class Reference

Sets the kinds of instances that can be replaced when instance resource limits (**com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.max_instances** (p. 1592)) are reached.

Inheritance diagram for DataReaderInstanceRemovalKind:



Static Public Attributes

- static final **DataReaderInstanceRemovalKind NO_INSTANCE_REMOVAL**

  No instance can be removed.
- static final **DataReaderInstanceRemovalKind EMPTY_INSTANCE_REMOVAL**

  Only empty instances can be removed.
- static final **DataReaderInstanceRemovalKind FULLY_PROCESSED_INSTANCE_REMOVAL**

  Only fully-processed instances can be removed.
- static final **DataReaderInstanceRemovalKind ANY_INSTANCE_REMOVAL**

  Any instance can be removed.

Additional Inherited Members

8.48.1 Detailed Description

Sets the kinds of instances that can be replaced when instance resource limits (**com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.max_instances** (p. 1592)) are reached.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.instance_replacement (p. 540)

8.48.2 Member Data Documentation

8.48.2.1 NO_INSTANCE_REMOVAL

```
final DataReaderInstanceRemovalKind NO_INSTANCE_REMOVAL [static]
```

Initial value:

```
=
            new DataReaderInstanceRemovalKind("NO_INSTANCE_REMOVAL", 0)
```

No instance can be removed.

If an instance resource is required because **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is reached, this setting will disallow instances from being replaced. Samples for new instances will be dropped and reported as lost with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_INSTANCES_LIMIT** (p. 1652).

8.48.2.2 EMPTY_INSTANCE_REMOVAL

```
final DataReaderInstanceRemovalKind EMPTY_INSTANCE_REMOVAL [static]
```

Initial value:

```
=
            new DataReaderInstanceRemovalKind("EMPTY_INSTANCE_REMOVAL", 1)
```

Only empty instances can be removed.

Instances can be replaced only if they are empty. An instance is considered empty when all samples have been taken or removed from the DataReader queue due to the **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234) or sample purging due to the **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy** (p. 1520), and there are no outstanding loans on any of the instance's samples.

8.48.2.3 FULLY_PROCESSED_INSTANCE_REMOVAL

```
final DataReaderInstanceRemovalKind FULLY_PROCESSED_INSTANCE_REMOVAL [static]
```

Initial value:

```
=
            new DataReaderInstanceRemovalKind(
                    "FULLY_PROCESSED_INSTANCE_REMOVAL",
                    2)
```

Only fully-processed instances can be removed.

An instance is considered fully processed if every sample for the instance has been processed by the application. A sample is considered processed by the application depending on the **com.rti.dds.infrastructure.ReliabilityQos↩ Policy.kind** (p. 1528):

- **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) (depends on the **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind** (p. 1530)):
  - **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1530) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530): The sample is considered processed when it has been read or taken by the application and return_loan has been called.
  - **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531): The sample is considered processed when the subscribing application has explicitly acknowledged the DDS sample, the AppAckConf has been received, and the application has called return_loan.
- **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): All samples are considered processed when they have been read or taken by the application and return_loan has been called.

8.48.2.4 ANY_INSTANCE_REMOVAL

```
final DataReaderInstanceRemovalKind ANY_INSTANCE_REMOVAL [static]
```

Initial value:

```
=
            new DataReaderInstanceRemovalKind("ANY_INSTANCE_REMOVAL", 3)
```

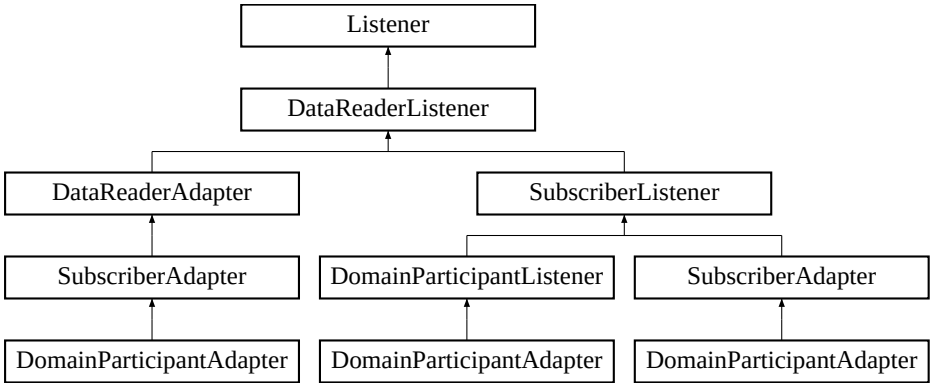
Any instance can be removed.

Instances can be replaced regardless of whether the subscribing application has processed all of the samples. Samples that have not been processed will be removed.

8.49 DataReaderListener Interface Reference

<<***interface***>> *(*p. *156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for reader status.

Inheritance diagram for DataReaderListener:



Public Member Functions

- void **on_requested_deadline_missed** (**DataReader** reader, **RequestedDeadlineMissedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.
- void **on_requested_incompatible_qos** (**DataReader** reader, **RequestedIncompatibleQosStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.
- void **on_sample_rejected** (**DataReader** reader, **SampleRejectedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.
- void **on_liveliness_changed** (**DataReader** reader, **LivelinessChangedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.
- void **on_data_available** (**DataReader** reader)

  Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS communication status.
- void **on_sample_lost** (**DataReader** reader, **SampleLostStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.
- void **on_subscription_matched** (**DataReader** reader, **SubscriptionMatchedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

8.49.1 Detailed Description

<<***interface***>> *(*p. *156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for reader status.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS;
com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS, **com.rti.dds.subscription.↩ LivelinessChangedStatus** (p. 1239);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS, **com.rti.dds.↩ subscription.RequestedDeadlineMissedStatus** (p. 1560);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.↩ subscription.RequestedIncompatibleQosStatus** (p. 1561);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS, **com.rti.dds.subscription.↩ SampleLostStatus** (p. 1648);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS, **com.rti.dds.subscription.↩ SampleRejectedStatus** (p. 1657);
com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS, **com.rti.dds.subscription.↩ SubscriptionMatchedStatus** (p. 1773);

See also

Status Kinds (p. 262)

Operations Allowed in Listener Callbacks (p. 1238)

8.49.2 Member Function Documentation

8.49.2.1 on_requested_deadline_missed()

```
void on_requested_deadline_missed (
              DataReader reader,
              RequestedDeadlineMissedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 486).

8.49.2.2 on_requested_incompatible_qos()

```
void on_requested_incompatible_qos (
              DataReader reader,
              RequestedIncompatibleQosStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 486).

8.49.2.3 on_sample_rejected()

```
void on_sample_rejected (
              DataReader reader,
              SampleRejectedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 486).

8.49.2.4 on_liveliness_changed()

```
void on_liveliness_changed (
              DataReader reader,
              LivelinessChangedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 486).

8.49.2.5 on_data_available()

```
void on_data_available (
              DataReader reader )
```

Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 487).

8.49.2.6 on_sample_lost()

```
void on_sample_lost (
              DataReader reader,
              SampleLostStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 487).

8.49.2.7 on_subscription_matched()

```
void on_subscription_matched (
              DataReader reader,
              SubscriptionMatchedStatus status )
```

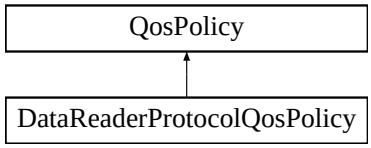
Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 487).

8.50 DataReaderProtocolQosPolicy Class Reference

Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.dds.infrastructure.Data↩ WriterProtocolQosPolicy** (p. 596), this QoS policy configures the DDS on-the-network protocol (RTPS).

Inheritance diagram for DataReaderProtocolQosPolicy:



Public Attributes

- **GUID_t virtual_guid**

  The virtual GUID (Global Unique Identifier).
- int **rtps_object_id**

  The RTPS Object ID.
- boolean **expects_inline_qos**

  Specifies whether this DataReader expects inline QoS with every sample.
- boolean **disable_positive_acks**

  Whether the reader sends positive acknowledgements to writers.
- boolean **propagate_dispose_of_unregistered_instances**

  *Indicates whether or not an instance can move to the **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_↩ DISPOSED_INSTANCE_STATE** (p. 1161) state without being in the **com.rti.dds.subscription.InstanceStateKind.↩ ALIVE_INSTANCE_STATE** (p. 1161) state.*
- boolean **propagate_unregister_of_disposed_instances**

  *Indicates whether or not an instance can move to the **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO↩ _WRITERS_INSTANCE_STATE** (p. 1161) state directly from the **com.rti.dds.subscription.InstanceStateKind.NOT_↩ ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).*
- final **RtpsReliableReaderProtocol_t rtps_reliable_reader**

  *RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **com.rti.dds.subscription.↩ DataReader** (p. 450). This parameter only has effect if the reader is configured with **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.infrastructure.ReliabilityQos↩ PolicyKind** (p. 1531).*

8.50.1 Detailed Description

Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) and **com.rti.dds.infrastructure.Data↩ WriterProtocolQosPolicy** (p. 596), this QoS policy configures the DDS on-the-network protocol (RTPS).

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.↩ subscription.DataReader** (p. 450) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) for more information on the per-DataReader/DataWriter reliability configuration. **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.50.2 Member Data Documentation

8.50.2.1 virtual_guid

```
GUID_t virtual_guid
```

The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **com.rti.dds.subscription.DataReader** (p. 450).

The association between a **com.rti.dds.subscription.DataReader** (p. 450) and its persisted state is done using the virtual GUID.

[default] com.rti.dds.infrastructure.GUID_t.GUID_AUTO (p. 1134)

8.50.2.2 rtps_object_id

```
int rtps_object_id
```

The RTPS Object ID.

This value is used to determine the RTPS object ID of a data reader according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data reader.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID (p. 1990)

[range] [0,0x00ffffff]

8.50.2.3 expects_inline_qos

```
boolean expects_inline_qos
```

Specifies whether this DataReader expects inline QoS with every sample.

RTI Connext DataWriters do not match with DataReaders that set this field to com.rti.dds.infrastructure.true (because RTI Connext DataWriters do not support sending inline QoS), but here is how the field is meant to be used:

In RTI Connext, a **com.rti.dds.subscription.DataReader** (p. 450) nominally relies on Discovery to propagate QoS on a matched **com.rti.dds.publication.DataWriter** (p. 553).

Alternatively, a **com.rti.dds.subscription.DataReader** (p. 450) may get information on a matched **com.rti.dds.↩ publication.DataWriter** (p. 553) through QoS sent inline with a sample.

Asserting **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.expects_inline_qos** (p. 503) indicates to a matching **com.rti.dds.publication.DataWriter** (p. 553) that this **com.rti.dds.subscription.DataReader** (p. 450) expects to receive inline QoS with every sample. The complete set of inline QoS that a **com.rti.dds.publication.↩ DataWriter** (p. 553) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Connext **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) cache Discovery information, inline QoS are largely redundant and thus unecessary. Only for other stateless implementations whose **com.rti.dds.subscription.DataReader** (p. 450) does not cache Discovery information is inline QoS necessary.

Also note that inline QoS are additional wire-payload that consume additional bandwidth and serialization and deserialization time.

[default] com.rti.dds.infrastructure.false

8.50.2.4 disable_positive_acks

```
boolean disable_positive_acks
```

Whether the reader sends positive acknowledgements to writers.

If set to com.rti.dds.infrastructure.true, the reader does not send positive acknowledgments (ACKs) in response to Heartbeat messages. The reader will send negative acknowledgements (NACKs) when a Heartbeat advertises samples that it has not received.

Otherwise, if set to com.rti.dds.infrastructure.false (the default), the reader will send ACKs to writers that expect ACKs (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_positive_acks** (p. 598) = com.rti.dds.↩ infrastructure.false) and it will not send ACKs to writers that disable ACKs (**com.rti.dds.infrastructure.DataWriter**↩ **ProtocolQosPolicy.disable_positive_acks** (p. 598) = com.rti.dds.infrastructure.true)

[default] com.rti.dds.infrastructure.false

8.50.2.5 propagate_dispose_of_unregistered_instances

```
boolean propagate_dispose_of_unregistered_instances
```

Indicates whether or not an instance can move to the **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_**↩ **DISPOSED_INSTANCE_STATE** (p. 1161) state without being in the **com.rti.dds.subscription.InstanceStateKind.**↩ **ALIVE_INSTANCE_STATE** (p. 1161) state.

This field only applies to keyed readers.

When the field is set to com.rti.dds.infrastructure.true, the DataReader will receive dispose notifications even if the instance is not alive.

To guarantee the key availability through the usage of the API **com.rti.ndds.example.FooDataReader.get_key_value** (p. 1095), this option should be used in combination with setting **com.rti.dds.infrastructure.DataWriterProtocolQos**↩ **Policy.serialize_key_with_dispose** (p. 599) on the DataWriter to com.rti.dds.infrastructure.true.

[default] com.rti.dds.infrastructure.false

8.50.2.6 propagate_unregister_of_disposed_instances

```
boolean propagate_unregister_of_disposed_instances
```

Indicates whether or not an instance can move to the **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_**↩ **NO_WRITERS_INSTANCE_STATE** (p. 1161) state directly from the **com.rti.dds.subscription.InstanceStateKind.**↩ **NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).

This field only applies to keyed readers.

When the field is set to com.rti.dds.infrastructure.true, the DataReader will receive unregister notifications even if the instance is not alive.

[default] com.rti.dds.infrastructure.false

8.50.2.7 rtps_reliable_reader

```
final RtpsReliableReaderProtocol_t rtps_reliable_reader
```

RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **com.rti.dds.↩ subscription.DataReader** (p. 450). This parameter only has effect if the reader is configured with **com.rti.dds.↩ infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyKind** (p. 1531).

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default] See **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

8.51 DataReaderProtocolStatus Class Reference

<<***extension***>> *(p. 155)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

Inherits Status.

Public Attributes

- long **received_sample_count**

  *The number of samples received by a **DataReader** (p. 450).*
- long **received_sample_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_count** (p. 507) since the last time the status was read.*
- long **received_sample_bytes**

  *The number of bytes received by a **DataReader** (p. 450).*
- long **received_sample_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_bytes** (p. 508) since the last time the status was read.*
- long **duplicate_sample_count**

  *The number of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 450).*
- long **duplicate_sample_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.duplicate_sample_count** (p. 509) since the last time the status was read.*
- long **duplicate_sample_bytes**

  *The number of bytes of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 450).*
- long **duplicate_sample_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.duplicate_sample_bytes** (p. 509) since the last time the status was read.*
- long **filtered_sample_count**

  ***[DEPRECATED]**. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_↩ count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)*
- long **filtered_sample_count_change**

***[DEPRECATED]**. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)*

- long **filtered_sample_bytes**

  ***[DEPRECATED]**. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)*

- long **filtered_sample_bytes_change**

  ***[DEPRECATED]**. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)*

- long **received_heartbeat_count**

  *The number of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 450).*

- long **received_heartbeat_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_heartbeat_count** (p. 511) since the last time the status was read.*

- long **received_heartbeat_bytes**

  *The number of bytes of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 450).*

- long **received_heartbeat_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_heartbeat_bytes** (p. 511) since the last time the status was read.*

- long **sent_ack_count**

  *The number of ACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.*

- long **sent_ack_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_ack_count** (p. 511) since the last time the status was read.*

- long **sent_ack_bytes**

  *The number of bytes of ACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.*

- long **sent_ack_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_ack_bytes** (p. 512) since the last time the status was read.*

- long **sent_nack_count**

  *The number of NACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.*

- long **sent_nack_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_nack_count** (p. 512) since the last time the status was read.*

- long **sent_nack_bytes**

  *The number of bytes of NACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.*

- long **sent_nack_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_nack_bytes** (p. 513) since the last time the status was read.*

- long **received_gap_count**

  *The number of GAPs received from remote DataWriter to this **DataReader** (p. 450).*

- long **received_gap_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_gap_count** (p. 513) since the last time the status was read.*

- long **received_gap_bytes**

  *The number of bytes of GAPs received from remote DataWriter to this **DataReader** (p. 450).*

- long **received_gap_bytes_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_gap_bytes** (p. 513) since the last time the status was read.*

- long **rejected_sample_count**

*The number of times a sample is rejected because it cannot be accepted by a reliable **DataReader** (p. 450).*

- long **rejected_sample_count_change**

  *The change in **com.rti.dds.subscription.DataReaderProtocolStatus.rejected_sample_count** (p. 514) since the last time the status was read.*

- **SequenceNumber_t first_available_sample_sequence_number**

  Sequence number of the first available sample in a matched DataWriters reliability queue.

- **SequenceNumber_t last_available_sample_sequence_number**

  Sequence number of the last available sample in a matched Datawriter's reliability queue.

- **SequenceNumber_t last_committed_sample_sequence_number**

  *Sequence number of the newest sample received from the matched DataWriter committed to the **DataReader** (p. 450)'s queue.*

- int **uncommitted_sample_count**

  Number of received samples that are not yet available to be read or taken, due to being received out of order.

- long **out_of_range_rejected_sample_count**

  *The number of samples dropped by the **DataReader** (p. 450) due to received window is full and the sample is out-of-order.*

- long **received_fragment_count**

  *The number of DATA_FRAG messages that have been received by this **DataReader** (p. 450).*

- long **dropped_fragment_count**

  *The number of DATA_FRAG messages that have been dropped by a **DataReader** (p. 450).*

- long **reassembled_sample_count**

  *The number of fragmented samples that have been reassembled by a **DataReader** (p. 450).*

- long **sent_nack_fragment_count**

  *The number of NACK fragments that have been sent from a **DataReader** (p. 450) to a DataWriter.*

- long **sent_nack_fragment_bytes**

  *The number of NACK fragment bytes that have been sent from a **DataReader** (p. 450) to a DataWriter.*

8.51.1 Detailed Description

<<***extension***>> *(p. 155)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

8.51.2 Member Data Documentation

8.51.2.1 received_sample_count

```
long received_sample_count
```

The number of samples received by a **DataReader** (p. 450).

Depending on how the **com.rti.dds.subscription.DataReaderProtocolStatus** (p. 505) was obtained this may count samples coming from a specific DataWriter or from all the DataWriters that are matched with the **DataReader** (p. 450).

If the **com.rti.dds.subscription.DataReaderProtocolStatus** (p. 505) is obtained using the **com.rti.dds.subscription.DataReader.get_datareader_protocol_status** (p. 463) operation then it will count samples from any DataWriter. If the **DataReaderProtocolStatus** (p. 505) is obtained using the **com.rti.dds.subscription.DataReader.get_matched_publication_datareader_protocol_status** (p. 465) then it will count the samples for the DataWriter specified as a parameter to the function.

Duplicate samples arriving from the DataWriter(s) (e.g. via multiple network paths) are detected prior to increasing this counter. The duplicate samples are counted by **com.rti.dds.subscription.DataReaderProtocolStatus.duplicate_sample_count** (p. 509).

If the **DataReader** (p. 450) has specified a ContentFilter the received samples that do not pass the filter are part of this counter. The filtered samples are counted by **com.rti.dds.subscription.DataReaderProtocolStatus.filtered_sample_count** (p. 510).

Samples rejected because they do not fit on the **DataReader** (p. 450) Queue are also part of this counter.

Note the received_sample_count counts samples received from all DataWriters and it does not necessarily match the number of samples accepted into the **DataReader** (p. 450) Queue. This is because:

- Samples can also be inserted into the **DataReader** (p. 450) Queue by lifecycle events that are locally detected like an instance becoming not alive as a result of DataWriters leaving the network.
- Samples can be filtered out due to ContentFilter or TimeFilter
- Samples can be rejected because there is no space in **DataReader** (p. 450) Queue

Note that when data is fragmented, this statistic is updated when all of the fragments required to reassemble a sample are received, not when individual fragments are received.

8.51.2.2 received_sample_count_change

```
long received_sample_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_count** (p. 507) since the last time the status was read.

See also

com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_count (p. 507)

Note that when data is fragmented, this statistic is updated when all of the fragments required to reassemble a sample are received, not when individual fragments are received.

8.51.2.3 received_sample_bytes

```
long received_sample_bytes
```

The number of bytes received by a **DataReader** (p. 450).

See also

com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_count (p. 507)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

8.51.2.4 received_sample_bytes_change

```
long received_sample_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_bytes** (p. 508) since the last time the status was read.

See also

com.rti.dds.subscription.DataReaderProtocolStatus.received_sample_count_change (p. 508)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

8.51.2.5 duplicate_sample_count

```
long duplicate_sample_count
```

The number of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 450).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.51.2.6 duplicate_sample_count_change

```
long duplicate_sample_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.duplicate_sample_count** (p. 509) since the last time the status was read.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.51.2.7 duplicate_sample_bytes

```
long duplicate_sample_bytes
```

The number of bytes of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 450).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.51.2.8 duplicate_sample_bytes_change

```
long duplicate_sample_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.duplicate_sample_bytes** (p. 509) since the last time the status was read.

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.51.2.9 filtered_sample_count

```
long filtered_sample_count
```

[DEPRECATED]. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)

8.51.2.10 filtered_sample_count_change

```
long filtered_sample_count_change
```

[DEPRECATED]. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)

8.51.2.11 filtered_sample_bytes

```
long filtered_sample_bytes
```

[DEPRECATED]. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)

8.51.2.12 filtered_sample_bytes_change

```
long filtered_sample_bytes_change
```

[DEPRECATED]. See: **com.rti.dds.subscription.DataReaderCacheStatus.time_based_filter_dropped_sample_count** (p. 491) **com.rti.dds.subscription.DataReaderCacheStatus.content_filter_dropped_sample_count** (p. 490)

8.51.2.13 received_heartbeat_count

```
long received_heartbeat_count
```

The number of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 450).

8.51.2.14 received_heartbeat_count_change

```
long received_heartbeat_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_heartbeat_count** (p. 511) since the last time the status was read.

8.51.2.15 received_heartbeat_bytes

```
long received_heartbeat_bytes
```

The number of bytes of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 450).

8.51.2.16 received_heartbeat_bytes_change

```
long received_heartbeat_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_heartbeat_bytes** (p. 511) since the last time the status was read.

8.51.2.17 sent_ack_count

```
long sent_ack_count
```

The number of ACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.

8.51.2.18 sent_ack_count_change

```
long sent_ack_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_ack_count** (p. 511) since the last time the status was read.

8.51.2.19 sent_ack_bytes

```
long sent_ack_bytes
```

The number of bytes of ACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.

8.51.2.20 sent_ack_bytes_change

```
long sent_ack_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_ack_bytes** (p. 512) since the last time the status was read.

8.51.2.21 sent_nack_count

```
long sent_nack_count
```

The number of NACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.

8.51.2.22 sent_nack_count_change

```
long sent_nack_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_nack_count** (p. 512) since the last time the status was read.

8.51.2.23 sent_nack_bytes

```
long sent_nack_bytes
```

The number of bytes of NACKs sent from a local **DataReader** (p. 450) to a matching remote DataWriter.

8.51.2.24 sent_nack_bytes_change

```
long sent_nack_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.sent_nack_bytes** (p. 513) since the last time the status was read.

8.51.2.25 received_gap_count

```
long received_gap_count
```

The number of GAPs received from remote DataWriter to this **DataReader** (p. 450).

8.51.2.26 received_gap_count_change

```
long received_gap_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_gap_count** (p. 513) since the last time the status was read.

8.51.2.27 received_gap_bytes

```
long received_gap_bytes
```

The number of bytes of GAPs received from remote DataWriter to this **DataReader** (p. 450).

8.51.2.28 received_gap_bytes_change

```
long received_gap_bytes_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.received_gap_bytes** (p. 513) since the last time the status was read.

8.51.2.29 rejected_sample_count

```
long rejected_sample_count
```

The number of times a sample is rejected because it cannot be accepted by a reliable **DataReader** (p. 450).

Samples rejected by a reliable **DataReader** (p. 450) will be NACKed, and they will have to be resent by the DataWriter if they are still available in the DataWriter queue.

This counter will always be 0 when using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_↩ RELIABILITY_QOS** (p. 1532).

8.51.2.30 rejected_sample_count_change

```
long rejected_sample_count_change
```

The change in **com.rti.dds.subscription.DataReaderProtocolStatus.rejected_sample_count** (p. 514) since the last time the status was read.

This counter will always be 0 when using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_↩ RELIABILITY_QOS** (p. 1532).

8.51.2.31 first_available_sample_sequence_number

```
SequenceNumber_t first_available_sample_sequence_number
```

Sequence number of the first available sample in a matched DataWriters reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

8.51.2.32 last_available_sample_sequence_number

```
SequenceNumber_t last_available_sample_sequence_number
```

Sequence number of the last available sample in a matched Datawriter's reliability queue.

Applicable only for reliable DataReaders, and when retrieving matched DataWriter statuses.

Updated upon receiving Heartbeat submessages from a matched reliable DataWriter.

8.51.2.33 last_committed_sample_sequence_number

```
SequenceNumber_t last_committed_sample_sequence_number
```

Sequence number of the newest sample received from the matched DataWriter committed to the **DataReader** (p. 450)'s queue.

Applicable only when retrieving matched DataWriter statuses.

For best-effort DataReaders, this is the sequence number of the latest sample received.

For reliable DataReaders, this is the sequence number of the latest sample that is available to be read or taken from the **DataReader** (p. 450)'s queue.

8.51.2.34 uncommitted_sample_count

```
int uncommitted_sample_count
```

Number of received samples that are not yet available to be read or taken, due to being received out of order.

Applicable only when retrieving matched DataWriter statuses.

8.51.2.35 out_of_range_rejected_sample_count

```
long out_of_range_rejected_sample_count
```

The number of samples dropped by the **DataReader** (p. 450) due to received window is full and the sample is out-of-order.

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532); if the **DataReader** (p. 450) received samples out-of-order, they are stored internally until the missing samples are received. The number of out-of-order samples that the **DataReader** (p. 450) can keep is set by **com.rti.dds.infrastructure.↩ RtpsReliableReaderProtocol_t.receive_window_size** (p. 1603). When the received window is full any out-of-order sample received will be dropped.

8.51.2.36 received_fragment_count

```
long received_fragment_count
```

The number of DATA_FRAG messages that have been received by this **DataReader** (p. 450).

This statistic is incremented upon the receipt of each DATA_FRAG message. Fragments from duplicate samples do not count towards this statistic. Applicable only when data is fragmented.

8.51.2.37 dropped_fragment_count

```
long dropped_fragment_count
```

The number of DATA_FRAG messages that have been dropped by a **DataReader** (p. 450).

This statistic does not include malformed fragments. Applicable only when data is fragmented.

8.51.2.38 reassembled_sample_count

```
long reassembled_sample_count
```

The number of fragmented samples that have been reassembled by a **DataReader** (p. 450).

This statistic is incremented when all of the fragments which are required to reassemble an entire sample have been received. Applicable only when data is fragmented.

8.51.2.39 sent_nack_fragment_count

```
long sent_nack_fragment_count
```

The number of NACK fragments that have been sent from a **DataReader** (p. 450) to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.51.2.40 sent_nack_fragment_bytes

```
long sent_nack_fragment_bytes
```

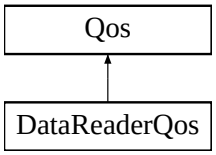
The number of NACK fragment bytes that have been sent from a **DataReader** (p. 450) to a DataWriter.

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.52 DataReaderQos Class Reference

QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 450) entity.

Inheritance diagram for DataReaderQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**DataReaderQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **DataReaderQos*** (p. *517) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **DataReaderQos*** (p. *517) object.*
- String **toString** (**DataReaderQos** baseQos)

  *Obtains a string representation of a **DataReaderQos*** (p. *517) object.*

Public Attributes

- final **DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY*** (p. *230).*
- final **DeadlineQosPolicy deadline**

  *Deadline policy, **DEADLINE*** (p. *217).*
- final **LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET*** (p. *238).*
- final **LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS*** (p. *239).*
- final **ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY*** (p. *258).*
- final **DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER*** (p. *218).*
- final **HistoryQosPolicy history**

  *History policy, **HISTORY*** (p. *237).*
- final **ResourceLimitsQosPolicy resource_limits**

  *Resource limits policy, **RESOURCE_LIMITS*** (p. *259).*
- final **UserDataQosPolicy user_data**

  *User data policy, **USER_DATA*** (p. *278).*
- final **OwnershipQosPolicy ownership**

  *Ownership policy, **OWNERSHIP*** (p. *244).*
- final **TimeBasedFilterQosPolicy time_based_filter**

*Time-based filter policy, **TIME_BASED_FILTER** (p. 267).*

- final **ReaderDataLifecycleQosPolicy reader_data_lifecycle**

  *Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 256).*

- final **DataReaderResourceLimitsQosPolicy reader_resource_limits**

  *<<**extension**>> (p. 155) **com.rti.dds.subscription.DataReader** (p. 450) resource limits policy, **DATA_READER_↩ RESOURCE_LIMITS** (p. 211). This policy is an extension to the DDS standard.*

- final **DataReaderProtocolQosPolicy protocol**

  *<<**extension**>> (p. 155) **com.rti.dds.subscription.DataReader** (p. 450) protocol policy, **DATA_READER_↩ PROTOCOL** (p. 210)*

- final **TransportSelectionQosPolicy transport_selection**

  *<<**extension**>> (p. 155) Transport selection policy, **TRANSPORT_SELECTION** (p. 275).*

- final **TransportUnicastQosPolicy unicast**

  *<<**extension**>> (p. 155) Unicast transport policy, **TRANSPORT_UNICAST** (p. 276).*

- final **TransportMulticastQosPolicy multicast**

  *<<**extension**>> (p. 155) Multicast transport policy, **TRANSPORT_MULTICAST** (p. 272).*

- final **PropertyQosPolicy property**

  *<<**extension**>> (p. 155) Property policy, **PROPERTY** (p. 248). See also `Property Reference Guide`.*

- final **DataTagQosPolicy data_tags**

  *DataTag policy, **DATA_TAG** (p. 214).*

- final **ServiceQosPolicy service**

  *<<**extension**>> (p. 155) Service policy, **SERVICE** (p. 261).*

- final **AvailabilityQosPolicy availability**

  *<<**extension**>> (p. 155) Availability policy, **AVAILABILITY** (p. 169).*

- final **EntityNameQosPolicy subscription_name**

  *<<**extension**>> (p. 155) EntityName policy, **ENTITY_NAME** (p. 234).*

- final **TransportPriorityQosPolicy transport_priority**

  *Transport priority policy, **TRANSPORT_PRIORITY** (p. 274).*

- final **TypeConsistencyEnforcementQosPolicy type_consistency**

  *Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 276).*

- final **DataRepresentationQosPolicy representation**

  *Data representation policy, **DATA_REPRESENTATION** (p. 212).*

- final **TypeSupportQosPolicy type_support**

  *<<**extension**>> (p. 155) type support data, **TYPESUPPORT** (p. 277).*

8.52.1 Detailed Description

QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 450) entity.

You must set certain members in a consistent manner:

com.rti.dds.subscription.DataReaderQos.deadline (p. 521) .period >= **com.rti.dds.subscription.DataReader↩ Qos.time_based_filter** (p. 523) .minimum_separation

com.rti.dds.subscription.DataReaderQos.history (p. 522) .depth <= **com.rti.dds.subscription.DataReaderQos.↩ resource_limits** (p. 522) .max_samples_per_instance

com.rti.dds.subscription.DataReaderQos.resource_limits (p. 522) .max_samples_per_instance <= **com.rti.dds.↩ subscription.DataReaderQos.resource_limits** (p. 522) .max_samples **com.rti.dds.subscription.DataReader↩ Qos.resource_limits** (p. 522) .initial_samples <= **com.rti.dds.subscription.DataReaderQos.resource_limits** (p. 522) .max_samples

com.rti.dds.subscription.DataReaderQos.resource_limits (p. 522) .initial_instances <= **com.rti.dds.subscription.↩ DataReaderQos.resource_limits** (p. 522) .max_instances

com.rti.dds.subscription.DataReaderQos.reader_resource_limits (p. 523) .initial_remote_writers_per_instance <= **com.rti.dds.subscription.DataReaderQos.reader_resource_limits** (p. 523) .max_remote_writers_per_instance

com.rti.dds.subscription.DataReaderQos.reader_resource_limits (p. 523) .initial_infos <= **com.rti.dds.↩ subscription.DataReaderQos.reader_resource_limits** (p. 523) .max_infos

com.rti.dds.subscription.DataReaderQos.reader_resource_limits (p. 523) .max_remote_writers_per_instance <= **com.rti.dds.subscription.DataReaderQos.reader_resource_limits** (p. 523) .max_remote_writers

com.rti.dds.subscription.DataReaderQos.reader_resource_limits (p. 523) .max_samples_per_remote_writer <= **com.rti.dds.subscription.DataReaderQos.resource_limits** (p. 522) .max_samples

length of **com.rti.dds.subscription.DataReaderQos.user_data** (p. 523) .value <= **com.rti.dds.domain.Domain↩ ParticipantQos.resource_limits** (p. 800) .reader_user_data_max_length

If any of the above are not true, **com.rti.dds.subscription.DataReader.set_qos** (p. 457) and **com.rti.dds.↩ subscription.DataReader.set_qos_with_profile** (p. 458) will fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596) and **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735) will return NULL.

8.52.2 Member Function Documentation

8.52.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 519) overloads allow formatting the output and printing only the differences with respect to another **DataReaderQos** (p. 517) object.

```
DataReaderQos qos = new DataReaderQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
DataReaderQos baseQos = new DataReaderQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value DATAREADER_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(DATAREADER_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **DataReaderQos** (p. 517) and the documented default.

Returns

The string representation of the Qos.

References **DataReaderQos.toString()**.

Referenced by **DataReaderQos.toString()**.

8.52.2.2 toString() [2/4]

```
String toString (
              DataReaderQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **DataReaderQos** (p. 517) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_PRINT_ALL** (p. 81) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **Subscriber.DATAREADER_QOS_PRINT_ALL**.

8.52.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **DataReaderQos** (p. 517) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DataReaderQos.toString()**.

8.52.2.4 toString() [4/4]

```
String toString (
              DataReaderQos baseQos )
```

Obtains a string representation of a **DataReaderQos** (p. 517) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_PRINT_ALL** (p. 81) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DataReaderQos.toString()**.

8.52.3 Member Data Documentation

8.52.3.1 durability

```
final DurabilityQosPolicy durability
```

Durability policy, **DURABILITY** (p. 230).

8.52.3.2 deadline

```
final DeadlineQosPolicy deadline
```

Deadline policy, **DEADLINE** (p. 217).

8.52.3.3 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 238).

8.52.3.4 liveliness

```
final LivelinessQosPolicy liveliness
```

Liveliness policy, **LIVELINESS** (p. 239).

8.52.3.5 reliability

```
final ReliabilityQosPolicy reliability
```

Reliability policy, **RELIABILITY** (p. 258).

8.52.3.6 destination_order

```
final DestinationOrderQosPolicy destination_order
```

Destination order policy, **DESTINATION_ORDER** (p. 218).

8.52.3.7 history

```
final HistoryQosPolicy history
```

History policy, **HISTORY** (p. 237).

8.52.3.8 resource_limits

```
final ResourceLimitsQosPolicy resource_limits
```

Resource limits policy, **RESOURCE_LIMITS** (p. 259).

8.52.3.9 user_data

```
final UserDataQosPolicy user_data
```

User data policy, **USER_DATA** (p. 278).

8.52.3.10 ownership

```
final OwnershipQosPolicy ownership
```

Ownership policy, **OWNERSHIP** (p. 244).

8.52.3.11 time_based_filter

```
final TimeBasedFilterQosPolicy time_based_filter
```

Time-based filter policy, **TIME_BASED_FILTER** (p. 267).

8.52.3.12 reader_data_lifecycle

```
final ReaderDataLifecycleQosPolicy reader_data_lifecycle
```

Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 256).

8.52.3.13 reader_resource_limits

```
final DataReaderResourceLimitsQosPolicy reader_resource_limits
```

<<***extension***>> *(p. 155)* **com.rti.dds.subscription.DataReader** (p. 450) resource limits policy, **DATA_READER_↩ RESOURCE_LIMITS** (p. 211). This policy is an extension to the DDS standard.

8.52.3.14 protocol

```
final DataReaderProtocolQosPolicy protocol
```

<<***extension***>> *(p. 155)* **com.rti.dds.subscription.DataReader** (p. 450) protocol policy, **DATA_READER_↩ PROTOCOL** (p. 210)

8.52.3.15 transport_selection

```
final TransportSelectionQosPolicy transport_selection
```

<<***extension***>> *(p. 155)* Transport selection policy, **TRANSPORT_SELECTION** (p. 275).

Specifies the transports available for use by the **com.rti.dds.subscription.DataReader** (p. 450).

8.52.3.16 unicast

```
final TransportUnicastQosPolicy unicast
```

<<***extension***>> *(p. 155)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 276).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **com.rti.dds.publication.DataWriter** (p. 553) entities in the domain.

8.52.3.17 multicast

```
final TransportMulticastQosPolicy multicast
```

<<***extension***>> *(p. 155)* Multicast transport policy, **TRANSPORT_MULTICAST** (p. 272).

Specifies the multicast group addresses and ports on which **messages** can be received.

The multicast addresses are used to receive messages from **com.rti.dds.publication.DataWriter** (p. 553) entities in the domain.

8.52.3.18 property

```
final PropertyQosPolicy property
```

<<***extension***>> *(p. 155)* Property policy, **PROPERTY** (p. 248). See also `Property Reference Guide`.

8.52.3.19 data_tags

```
final DataTagQosPolicy data_tags
```

DataTag policy, **DATA_TAG** (p. 214).

8.52.3.20 service

```
final ServiceQosPolicy service
```

<<***extension***>> *(p. 155)* Service policy, **SERVICE** (p. 261).

8.52.3.21 availability

```
final AvailabilityQosPolicy availability
```

<<***extension***>> *(p. 155)* Availability policy, **AVAILABILITY** (p. 169).

8.52.3.22 subscription_name

```
final EntityNameQosPolicy subscription_name
```

<<***extension***>> *(p. 155)* EntityName policy, **ENTITY_NAME** (p. 234).

8.52.3.23 transport_priority

```
final TransportPriorityQosPolicy transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 274).

8.52.3.24 type_consistency

```
final TypeConsistencyEnforcementQosPolicy type_consistency
```

Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 276).

8.52.3.25 representation

```
final DataRepresentationQosPolicy representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 212).

8.52.3.26 type_support

```
final TypeSupportQosPolicy type_support
```

<<***extension***>> *(p. 155)* type support data, **TYPESUPPORT** (p. 277).

Optional value that is passed to a type plugin's on_endpoint_attached and deserialization functions.

8.53 DataReaderResourceLimitsInstanceReplacementSettings Class Reference

Instance replacement kind applied to each instance state.

Inherits Struct.

Public Member Functions

- **DataReaderResourceLimitsInstanceReplacementSettings** ()

  Default constructor creates the policy with the default values.
- **DataReaderResourceLimitsInstanceReplacementSettings** (**DataReaderInstanceRemovalKind** alive↩ InstanceRemovalKind, **DataReaderInstanceRemovalKind** disposedInstanceRemovalKind, **DataReader**↩ **InstanceRemovalKind** noWritersInstanceRemovalKind)

  Constructor with the given alive_instance_removal, disposed_instance_removal and no_writers_instance_removal values.

Public Attributes

- **DataReaderInstanceRemovalKind alive_instance_removal**

  *Removal kind applied to alive (**com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161)) instances.*
- **DataReaderInstanceRemovalKind disposed_instance_removal**

  *Removal kind applied to disposed (**com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_**↩ ***INSTANCE_STATE** (p. 1161)) instances.*
- **DataReaderInstanceRemovalKind no_writers_instance_removal**

  *Removal kind applied to fully-unregistered (**com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_**↩ ***WRITERS_INSTANCE_STATE** (p. 1161)) instances.*

8.53.1 Detailed Description

Instance replacement kind applied to each instance state.

[default]

- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.alive_instance_removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.NO_INSTANCE_REMOVAL** (p. 496)
- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.disposed_instance_removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL** (p. 496)
- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.no_writers_instance_removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL** (p. 496)

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.instance_replacement (p. 540)

8.53.2 Constructor & Destructor Documentation

8.53.2.1 DataReaderResourceLimitsInstanceReplacementSettings() [1/2]

```
DataReaderResourceLimitsInstanceReplacementSettings ( )
```

Default constructor creates the policy with the default values.

8.53.2.2 DataReaderResourceLimitsInstanceReplacementSettings() [2/2]

```
DataReaderResourceLimitsInstanceReplacementSettings (
            DataReaderInstanceRemovalKind aliveInstanceRemovalKind,
            DataReaderInstanceRemovalKind disposedInstanceRemovalKind,
            DataReaderInstanceRemovalKind noWritersInstanceRemovalKind )
```

Constructor with the given alive_instance_removal, disposed_instance_removal and no_writers_instance_removal values.

References **DataReaderResourceLimitsInstanceReplacementSettings.alive_instance_removal**, **DataReaderResourceLimitsInstanceReplacementSettings.disposed_instance_removal**, and **DataReaderResourceLimitsInstanceReplacementSettings.no_writers_instance_removal**.

8.53.3 Member Data Documentation

8.53.3.1 alive_instance_removal

```
DataReaderInstanceRemovalKind alive_instance_removal
```

Initial value:

```
= 
        DataReaderInstanceRemovalKind.NO_INSTANCE_REMOVAL
```

Removal kind applied to alive (**com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161)) instances.

[default] com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.NO_INSTANCE_REMOVAL (p. 496)

Referenced by **DataReaderResourceLimitsInstanceReplacementSettings.DataReaderResourceLimitsInstance↩ ReplacementSettings()**.

8.53.3.2 disposed_instance_removal

```
DataReaderInstanceRemovalKind disposed_instance_removal
```

Initial value:

```
= 
        DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL
```

Removal kind applied to disposed (**com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_↩ INSTANCE_STATE** (p. 1161)) instances.

[default] com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL (p. 496)

Referenced by **DataReaderResourceLimitsInstanceReplacementSettings.DataReaderResourceLimitsInstance↩ ReplacementSettings()**.

8.53.3.3 no_writers_instance_removal

```
DataReaderInstanceRemovalKind no_writers_instance_removal
```

Initial value:

```
= 
        DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL
```

Removal kind applied to fully-unregistered (**com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE** (p. 1161)) instances.

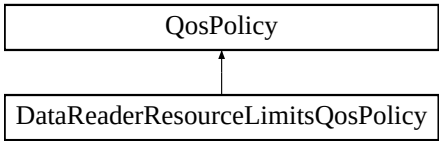
[default] com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_INSTANCE_REMOVAL (p. 496)

Referenced by **DataReaderResourceLimitsInstanceReplacementSettings.DataReaderResourceLimitsInstance↩ ReplacementSettings()**.

8.54 DataReaderResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 450) allocates and uses physical memory for internal resources.

Inheritance diagram for DataReaderResourceLimitsQosPolicy:



Public Attributes

- int **max_remote_writers**

  *The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.*
- int **max_remote_writers_per_instance**

  *The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read a single instance.*
- int **max_samples_per_remote_writer**

  *The maximum number of out-of-order samples from a given remote **com.rti.dds.publication.DataWriter** (p. 553) that a **com.rti.dds.subscription.DataReader** (p. 450) may store when maintaining a reliable connection to the **com.rti.dds.↩ publication.DataWriter** (p. 553).*
- int **max_infos**

  *The maximum number of info units that a **com.rti.dds.subscription.DataReader** (p. 450) can use to store **com.rti.↩ dds.subscription.SampleInfo** (p. 1634).*
- int **initial_remote_writers**

  *The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.*
- int **initial_remote_writers_per_instance**

  *The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read a single instance.*
- int **initial_infos**

  *The initial number of info units that a **com.rti.dds.subscription.DataReader** (p. 450) can have, which are used to store **com.rti.dds.subscription.SampleInfo** (p. 1634).*
- int **initial_outstanding_reads**

  *The initial number of outstanding calls to read/take (or one of their variants) on the same **com.rti.dds.subscription.Data↩ Reader** (p. 450) for which memory has not been returned by calling **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).*
- int **max_outstanding_reads**

  *The maximum number of outstanding read/take calls (or one of their variants) on the same **com.rti.dds.subscription.↩ DataReader** (p. 450) for which memory has not been returned by calling **com.rti.ndds.example.FooDataReader.↩ return_loan** (p. 1094).*
- int **max_samples_per_read**

  *The maximum number of data samples that the application can receive from the middleware in a single call to **com.rti.↩ ndds.example.FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071). If more data exists in the middleware, the application will need to issue multiple read/take calls.*

- boolean **disable_fragmentation_support**

  *Determines whether the **com.rti.dds.subscription.DataReader** (p. 450) can receive fragmented samples.*
- int **max_fragmented_samples**

  *The maximum number of samples for which the **com.rti.dds.subscription.DataReader** (p. 450) may store fragments at a given point in time.*
- int **initial_fragmented_samples**

  *The initial number of samples for which a **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.*
- int **max_fragmented_samples_per_remote_writer**

  *The maximum number of samples per remote writer for which a **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.*
- int **max_fragments_per_sample**

  Maximum number of fragments for a single sample.
- boolean **dynamically_allocate_fragmented_samples**

  *Determines whether the **com.rti.dds.subscription.DataReader** (p. 450) pre-allocates storage for storing fragmented samples.*
- int **max_total_instances**

  Maximum number of instances for which a DataReader will keep state.
- int **max_remote_virtual_writers**

  *The maximum number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.*
- int **initial_remote_virtual_writers**

  *The initial number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.*
- int **max_remote_virtual_writers_per_instance**

  The maximum number of virtual remote writers that can be associated with an instance.
- int **initial_remote_virtual_writers_per_instance**

  The initial number of virtual remote writers per instance.
- int **max_remote_writers_per_sample**

  The maximum number of remote writers allowed to write the same sample.
- int **max_query_condition_filters**

  The maximum number of query condition filters a reader is allowed.
- int **max_app_ack_response_length**

  Maximum length of application-level acknowledgment response data.
- boolean **keep_minimum_state_for_instances**

  *Whether or not keep a minimum instance state for up to **com.rti.dds.infrastructure.DataReaderResourceLimitsQos↩ Policy.max_total_instances** (p. 536).*
- int **initial_topic_queries**

  *The initial number of TopicQueries allocated by a **com.rti.dds.subscription.DataReader** (p. 450).*
- int **max_topic_queries**

  *The maximum number of active TopicQueries that a **com.rti.dds.subscription.DataReader** (p. 450) can create.*
- **DataReaderResourceLimitsInstanceReplacementSettings instance_replacement**

  *Sets the kind of instances allowed to be replaced for each instance state (**com.rti.dds.subscription.InstanceState↩ Kind** (p. 1160)) when a DataReader reaches **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592).*
- **Duration_t autopurge_remote_not_alive_writer_delay**

  *Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding a **com.rti.dds.publication.DataWriter** (p. 553) once the **com.rti.dds.publication.DataWriter** (p. 553) has become not alive.*
- **Duration_t autopurge_remote_virtual_writer_delay**

  *Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding a virtual **com.rti.dds.publication.DataWriter** (p. 553) once the **com.rti.dds.publication.DataWriter** (p. 553) has become detached.*

Static Public Attributes

- static final int **AUTO_MAX_TOTAL_INSTANCES**

  *<<**extension**>> (p. 155) This value is used to make **com.rti.dds.infrastructure.DataReaderResourceLimitsQos↩ Policy.max_total_instances** (p. 536) equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592).*

8.54.1 Detailed Description

Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 450) allocates and uses physical memory for internal resources.

DataReaders must allocate internal structures to handle the maximum number of DataWriters that may connect to it, whether or not a **com.rti.dds.subscription.DataReader** (p. 450) handles data fragmentation and how many data fragments that it may handle (for data samples larger than the MTU of the underlying network transport), how many simultaneous outstanding loans of internal memory holding data samples can be provided to user code, as well as others.

Most of these internal structures start at an initial size and, by default, will grow as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **com.rti.dds.subscription.DataReader** (p. 450). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **com.↩ rti.dds.subscription.DataReader** (p. 450).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.54.2 Member Data Documentation

8.54.2.1 max_remote_writers

```
int max_remote_writers
```

The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)
[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_remote_writers, >= max_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers_per_instance if max_remote_writers_per_instance is not equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

Note: For efficiency, set `max_remote_writers` >= **com.rti.dds.infrastructure.DataReaderResourceLimits↩ QosPolicy.max_remote_writers_per_instance** (p. 531).

8.54.2.2 max_remote_writers_per_instance

```
int max_remote_writers_per_instance
```

The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read a single instance.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1024] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), <= max_remote_writers or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers if it is not **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

Note: For efficiency, set `max_remote_writers_per_instance` <= **com.rti.dds.infrastructure.DataReader↩ ResourceLimitsQosPolicy.max_remote_writers** (p. 531)

8.54.2.3 max_samples_per_remote_writer

```
int max_samples_per_remote_writer
```

The maximum number of out-of-order samples from a given remote **com.rti.dds.publication.DataWriter** (p. 553) that a **com.rti.dds.subscription.DataReader** (p. 450) may store when maintaining a reliable connection to the **com.rti.↩ dds.publication.DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)

8.54.2.4 max_infos

```
int max_infos
```

The maximum number of info units that a **com.rti.dds.subscription.DataReader** (p. 450) can use to store **com.rti.↩ dds.subscription.SampleInfo** (p. 1634).

When read/take is called on a DataReader, the DataReader passes a sequence of data samples and an associated sample info sequence. The sample info sequence contains additional information for each data sample.

max_infos determines the resources allocated for storing sample info. This memory is loaned to the application when passing a sample info sequence.

Note that sample info is a snapshot, generated when read/take is called.

max_infos should not be less than max_samples.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_infos

8.54.2.5 initial_remote_writers

```
int initial_remote_writers
```

The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.

[default] 2

[range] [1, 1 million], <= max_remote_writers

For unkeyed types this value has to be equal to initial_remote_writers_per_instance.

Note: For efficiency, set initial_remote_writers >= **com.rti.dds.infrastructure.DataReaderResourceLimitsQos↩ Policy.initial_remote_writers_per_instance** (p. 533).

8.54.2.6 initial_remote_writers_per_instance

```
int initial_remote_writers_per_instance
```

The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read a single instance.

[default] 2

[range] [1,1024], <= max_remote_writers_per_instance

For unkeyed types this value has to be equal to initial_remote_writers.

Note: For efficiency, set initial_remote_writers_per_instance <= **com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.initial_remote_writers** (p. 532).

8.54.2.7 initial_infos

```
int initial_infos
```

The initial number of info units that a **com.rti.dds.subscription.DataReader** (p. 450) can have, which are used to store **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] 32

[range] [1,1 million], <= max_infos

8.54.2.8 initial_outstanding_reads

```
int initial_outstanding_reads
```

The initial number of outstanding calls to read/take (or one of their variants) on the same **com.rti.dds.subscription.↩ DataReader** (p. 450) for which memory has not been returned by calling **com.rti.ndds.example.FooDataReader.↩ return_loan** (p. 1094).

[default] 2

[range] [1, 65536], <= max_outstanding_reads

8.54.2.9 max_outstanding_reads

```
int max_outstanding_reads
```

The maximum number of outstanding read/take calls (or one of their variants) on the same **com.rti.dds.subscription.↩ DataReader** (p. 450) for which memory has not been returned by calling **com.rti.ndds.example.FooDataReader.↩ return_loan** (p. 1094).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 65536] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_outstanding_reads

8.54.2.10 max_samples_per_read

```
int max_samples_per_read
```

The maximum number of data samples that the application can receive from the middleware in a single call to **com.rti.↩ ndds.example.FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071). If more data exists in the middleware, the application will need to issue multiple read/take calls.

When reading data using listeners, the expected number of samples available for delivery in a single `take` call is typically small: usually just one, in the case of unbatched data, or the number of samples in a single batch, in the case of batched data. (See **com.rti.dds.infrastructure.BatchQosPolicy** (p. 355) for more information about this feature.) When polling for data or using a **com.rti.dds.infrastructure.WaitSet** (p. 1973), however, multiple samples (or batches) could be retrieved at once, depending on the data rate.

A larger value for this parameter makes the API simpler to use at the expense of some additional memory consumption.

[default] 1024

[range] [1,65536]

8.54.2.11 disable_fragmentation_support

```
boolean disable_fragmentation_support
```

Determines whether the **com.rti.dds.subscription.DataReader** (p. 450) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] com.rti.dds.infrastructure.false

8.54.2.12 max_fragmented_samples

```
int max_fragmented_samples
```

The maximum number of samples for which the **com.rti.dds.subscription.DataReader** (p. 450) may store fragments at a given point in time.

At any given time, a **com.rti.dds.subscription.DataReader** (p. 450) may store fragments for up to `max_fragmented_samples` samples while waiting for the remaining fragments. These samples need not have consecutive sequence numbers and may have been sent by different **com.rti.dds.publication.DataWriter** (p. 553) instances.

Once all fragments of a sample have been received, the sample is treated as a regular sample and becomes subject to standard QoS settings such as **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592).

The middleware will drop fragments if the `max_fragmented_samples` limit has been reached. For best-effort communication, the middleware will accept a fragment for a new sample, but drop the oldest fragmented sample from the same remote writer. For reliable communication, the middleware will drop fragments for any new samples until all fragments for at least one older sample from that writer have been received.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

[default] 1024

[range] [1, 1 million]

8.54.2.13 initial_fragmented_samples

```
int initial_fragmented_samples
```

The initial number of samples for which a **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

[default] 4

[range] [1,1024], <= max_fragmented_samples

8.54.2.14 max_fragmented_samples_per_remote_writer

```
int max_fragmented_samples_per_remote_writer
```

The maximum number of samples per remote writer for which a **com.rti.dds.subscription.DataReader** (p. 450) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

8.54.2.15 max_fragments_per_sample

```
int max_fragments_per_sample
```

Maximum number of fragments for a single sample.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.54.2.16 dynamically_allocate_fragmented_samples

```
boolean dynamically_allocate_fragmented_samples
```

Determines whether the **com.rti.dds.subscription.DataReader** (p. 450) pre-allocates storage for storing fragmented samples.

By default, the middleware does not allocate memory upfront, but instead allocates memory from the heap upon receiving the first fragment of a new sample. The amount of memory allocated equals the amount of memory needed to store all fragments in the sample. Once all fragments of a sample have been received, the sample is deserialized and stored in the regular receive queue. At that time, the dynamically allocated memory is freed again.

This QoS setting is useful for large, but variable-sized data types where upfront memory allocation for multiple samples based on the maximum possible sample size may be expensive. The main disadvantage of not pre-allocating memory is that one can no longer guarantee the middleware will have sufficient resources at runtime.

If `dynamically_allocate_fragmented_samples` is set to com.rti.dds.infrastructure.false, the middleware will allocate memory upfront for storing fragments for up to **com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.initial_fragmented_samples** (p. 535) samples. This memory may grow up to **com.rti.dds.↩ infrastructure.DataReaderResourceLimitsQosPolicy.max_fragmented_samples** (p. 535) if needed.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 534) is com.rti.dds.infrastructure.false.

[default] com.rti.dds.infrastructure.true

8.54.2.17 max_total_instances

```
int max_total_instances
```

Maximum number of instances for which a DataReader will keep state.

The maximum number of instances actively managed by a DataReader is determined by **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.max_instances** (p. 1592).

These instances have associated DataWriters or samples in the DataReader's queue and are visible to the user through operations such as **com.rti.ndds.example.FooDataReader.take** (p. 1071), **com.rti.ndds.example.FooDataReader.↩ read** (p. 1069), and **com.rti.ndds.example.FooDataReader.get_key_value** (p. 1095).

The features Durable Reader State, MultiChannel DataWriters and RTI Persistence Service require RTI Connext to keep some internal state even for instances without DataWriters or samples in the DataReader's queue. The additional state is used to filter duplicate samples that could be coming from different DataWriter channels or from multiple executions of RTI Persistence Service.

The total maximum number of instances that will be managed by the middleware, including instances without associated DataWriters or samples, is determined by max_total_instances.

When a new instance is received, RTI Connext will check the resource limit **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.max_instances** (p. 1592). If the limit is exceeded, RTI Connext will drop the sample with the reason LOST_BY_INSTANCES_LIMIT. If the limit is not exceeded, RTI Connext will check max_total_instances. If max_total↩ _instances is exceeded, RTI Connext will replace an existing instance without DataWriters and samples with the new one. The application could receive duplicate samples for the replaced instance if it becomes alive again.

The max_total_instances limit is not used if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.↩ keep_minimum_state_for_instances** (p. 539) is false, and in that case should be left at the default value.

[default] **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.AUTO_MAX_TOTAL_INSTANCES** (p. 212)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) or **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.AUTO_MAX_TOTAL_INSTANCES** (p. 212), >= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)

8.54.2.18 max_remote_virtual_writers

```
int max_remote_virtual_writers
```

The maximum number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is set to com.rti.dds.↩ infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS, this value determines the maximum number of DataWriter groups that can be managed by the **com.rti.dds.↩ subscription.Subscriber** (p. 1730) containing this **com.rti.dds.subscription.DataReader** (p. 450).

Since the **com.rti.dds.subscription.Subscriber** (p. 1730) may contain more than one **com.rti.dds.subscription.↩ DataReader** (p. 450), only the setting of the first applies.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)
[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_remote_virtual_writers, >= max_remote_virtual_writers_per_instance

8.54.2.19 initial_remote_virtual_writers

```
int initial_remote_virtual_writers
```

The initial number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read, including all instances.

[default] 2
[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), <= max_remote_virtual_writers

8.54.2.20 max_remote_virtual_writers_per_instance

```
int max_remote_virtual_writers_per_instance
```

The maximum number of virtual remote writers that can be associated with an instance.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1024] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

The features of Durable Reader State and MultiChannel DataWriters, and RTI Persistence Service require RTI Connext to keep some internal state per virtual writer and instance that is used to filter duplicate samples. These duplicate samples could be coming from different DataWriter channels or from multiple executions of RTI Persistence Service.

Once an association between a remote virtual writer and an instance is established, it is permanent – it will not disappear even if the physical writer incarnating the virtual writer is destroyed.

If max_remote_virtual_writers_per_instance is exceeded for an instance, RTI Connext will not associate this instance with new virtual writers. Duplicates samples from these virtual writers will not be filtered on the reader.

If you are not using Durable Reader State, MultiChannel DataWriters or RTI Persistence Service in your system, you can set this property to 1 to optimize resources.

8.54.2.21 initial_remote_virtual_writers_per_instance

```
int initial_remote_virtual_writers_per_instance
```

The initial number of virtual remote writers per instance.

[default] 2

[range] [1, 1024], <= max_remote_virtual_writers_per_instance

For unkeyed types, this value is ignored.

8.54.2.22 max_remote_writers_per_sample

```
int max_remote_writers_per_sample
```

The maximum number of remote writers allowed to write the same sample.

One scenario in which two DataWriters may write the same sample is Persistence Service. The DataReader may receive the same sample coming from the original DataWriter and from a Persistence Service DataWriter. **[default]** 3

[range] [1, 1024]

8.54.2.23 max_query_condition_filters

```
int max_query_condition_filters
```

The maximum number of query condition filters a reader is allowed.

[default] 4

[range] [0, 32]

This value determines the maximum number of unique query condition content filters that a reader may create.

Each query condition content filter is comprised of both its `query_expression` and `query_parameters`. Two query conditions that have the same `query_expression` will require unique query condition filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition filter.

8.54.2.24 max_app_ack_response_length

```
int max_app_ack_response_length
```

Maximum length of application-level acknowledgment response data.

The maximum length of response data in an application-level acknowledgment.

When set to zero, no response data is sent with application-level acknowledgments.

[default] 1

[range] [0, 32768]

8.54.2.25 keep_minimum_state_for_instances

```
boolean keep_minimum_state_for_instances
```

Whether or not keep a minimum instance state for up to **com.rti.dds.infrastructure.DataReaderResourceLimits↩ QosPolicy.max_total_instances** (p. 536).

The features Durable Reader State, multi-channel DataWriters, and Persistence Service require RTI Connext to keep some minimal internal state even for instances without DataWriters or DDS samples in the DataReader's queue, or that have been purged due to a dispose. The additional state is used to filter duplicate DDS samples that could be coming from different DataWriter channels or from multiple executions of Persistence Service. The total maximum number of instances that will be managed by the middleware, including instances without associated DataWriters or DDS samples or that have been purged due to a dispose, is determined by **com.rti.dds.infrastructure.DataReader↩ ResourceLimitsQosPolicy.max_total_instances** (p. 536).

This additional state will only be kept for up to max_total_instances if this field is set to com.rti.dds.infrastructure.true, otherwise the additional state will not be kept for any instances.

The minimum state includes information such as the source timestamp of the last sample received by the instance and the last sequence number received from a virtual GUID.

[default] com.rti.dds.infrastructure.true

8.54.2.26 initial_topic_queries

```
int initial_topic_queries
```

The initial number of TopicQueries allocated by a **com.rti.dds.subscription.DataReader** (p. 450).

[default] 1

See also

com.rti.dds.subscription.TopicQuery (p. 1830)

8.54.2.27 max_topic_queries

```
int max_topic_queries
```

The maximum number of active TopicQueries that a **com.rti.dds.subscription.DataReader** (p. 450) can create.

Once this limit is reached, a **com.rti.dds.subscription.DataReader** (p. 450) can create more TopicQueries only if it deletes some of the previously created ones.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

See also

com.rti.dds.subscription.TopicQuery (p. 1830)

8.54.2.28 instance_replacement

```
DataReaderResourceLimitsInstanceReplacementSettings instance_replacement
```

Sets the kind of instances allowed to be replaced for each instance state (**com.rti.dds.subscription.InstanceState↩ Kind** (p. 1160)) when a DataReader reaches **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592).

When **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is reached, a **com.rti.dds.↩ subscription.DataReader** (p. 450) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kinds specified by this field.

A DataReader can choose what kinds of instances can be replaced for each **com.rti.dds.subscription.Instance↩ StateKind** (p. 1160) separately. This means, for example, that a DataReader can choose to not allow replacing alive (**com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161)) instances but allow replacement of empty disposed (**com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE↩ _STATE** (p. 1161)) instances.

Only instances whose states match the specified kinds are eligible to be replaced. In addition, there must be no outstanding loans on any of the samples belonging to the instance for it to be considered for replacement.

For all kinds, a **com.rti.dds.subscription.DataReader** (p. 450) will replace the least-recently-updated instance satisfying that kind. An instance is considered 'updated' when a valid sample or dispose sample is received and accepted for that instance. When using **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1348), only samples that are received from the owner of the instance will cause the instance to be considered updated. An instance is not considered updated when an unregister sample is received because the unregister message simply indicates that there is one less writer that has updates for the instance, not that the instance itself was updated.

If no replaceable instance exists, the sample for the new instance will be considered lost with lost reason **com.rti.dds.↩ subscription.SampleLostStatusKind.LOST_BY_INSTANCES_LIMIT** (p. 1652) and the instance will not be asserted into the DataReader queue.

[default]

- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.alive_instance_↩ removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.NO_INSTANCE_REMOVAL** (p. 496)
- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.disposed_instance↩ _removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_INSTANCE_↩ REMOVAL** (p. 496)
- **com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings.no_writers_↩ instance_removal** (p. 528) = **com.rti.dds.infrastructure.DataReaderInstanceRemovalKind.EMPTY_↩ INSTANCE_REMOVAL** (p. 496)

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsInstanceReplacementSettings (p. 526)

8.54.2.29 autopurge_remote_not_alive_writer_delay

```
Duration_t autopurge_remote_not_alive_writer_delay
```

Initial value:

```
= new Duration_t(
        Duration_t.DURATION_AUTO_SEC, Duration_t.DURATION_AUTO_NSEC)
```

Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding a **com.rti.dds.publication.DataWriter** (p. 553) once the **com.rti.dds.publication.DataWriter** (p. 553) has become not alive.

After this time elapses, the **com.rti.dds.subscription.DataReader** (p. 450) will purge all internal information regarding the not alive **com.rti.dds.publication.DataWriter** (p. 553).

See also

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1243) for more information on when a **com.rti.dds.↩ publication.DataWriter** (p. 553) is considered not alive.

When set to com.rti.dds.infrastructure.Duration_t.AUTO, this parameter is set to 10 times the value of **com.rti.dds.↩ infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_lease_duration** (p. 649).

This QoS only applies when the **com.rti.dds.subscription.DataReader** (p. 450) is using **com.rti.dds.infrastructure.↩ InstanceStateConsistencyKind.RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 1159) for the **com.rti.dds.↩ infrastructure.ReliabilityQosPolicy.instance_state_consistency_kind** (p. 1529) QoS. When using **com.rti.dds.↩ infrastructure.InstanceStateConsistencyKind.RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 1159), a **com.↩ rti.dds.subscription.DataReader** (p. 450) keeps state about all **com.rti.dds.publication.DataWriter** (p. 553) entities and the instances they were writing in order to be able to transition those instances back to their correct state if liveliness with the **com.rti.dds.publication.DataWriter** (p. 553) is recovered. This can cause unbounded memory growth if that state is never purged and **com.rti.dds.publication.DataWriter** (p. 553) entities continuously come and go in a system. This QoS avoids that unbounded memory growth by setting a time at which that state will be purged.

This QoS should be set such that it is longer than the longest period of time for which a **com.rti.dds.publication.↩ DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) are expected to be disconnected and then reconnected in your system.

An alternative to using this QoS to purge the state is to set the **com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.max_remote_writers** (p. 531) QoS to a finite value. If that QoS is set to a finite value and the number of alive + not alive **com.rti.dds.publication.DataWriter** (p. 553) entities reaches the limit when a new **com.rti.dds.↩ publication.DataWriter** (p. 553) is discovered, the oldest not alive **com.rti.dds.publication.DataWriter** (p. 553) will be replaced.

[default] com.rti.dds.infrastructure.Duration_t.AUTO

[range] $>$ 0 or com.rti.dds.infrastructure.Duration_t.AUTO

8.54.2.30 autopurge_remote_virtual_writer_delay

```
Duration_t autopurge_remote_virtual_writer_delay
```

Initial value:

```
= new Duration_t(
        Duration_t.DURATION_INFINITE_SEC, Duration_t.DURATION_INFINITE_NSEC)
```

Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding a virtual **com.rti.dds.publication.DataWriter** (p. 553) once the **com.rti.dds.publication.DataWriter** (p. 553) has become detached.

Determines how long the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding a virtual **com.rti.dds.publication.DataWriter** (p. 553) that has has become detached.

After this time elapses, the **com.rti.dds.subscription.DataReader** (p. 450) will purge all internal information regarding the **com.rti.dds.publication.DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] >= 0 or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.55 DataReaderSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.DataReader** (p. 450) > .

Inherits AbstractNativeSequence.

Public Member Functions

- **DataReaderSeq** (Collection readers)
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.55.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.DataReader** (p. 450) > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.55.2 Constructor & Destructor Documentation

8.55.2.1 DataReaderSeq()

```
DataReaderSeq (
            Collection readers )
```

Exceptions

NullPointerException	if the given collection is null

8.55.3 Member Function Documentation

8.55.3.1 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 329), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

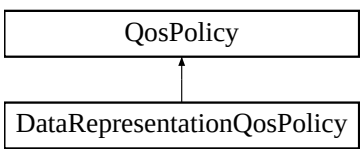
com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

8.56 DataRepresentationQosPolicy Class Reference

This QoS policy contains a list of representation identifiers and compression settings used by **com.rti.dds.↩ publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities to negotiate which data representation and compression settings to use.

Inheritance diagram for DataRepresentationQosPolicy:



Public Attributes

- final **ShortSeq value** = new **ShortSeq**()

  Sequence of representation identifiers.
- **CompressionSettings_t compression_settings** = new **CompressionSettings_t**()

  <<***extension***>> *(p. 155) Structure that contains the compression settings.*

Static Public Attributes

- static final short **XCDR_DATA_REPRESENTATION** = 0

  Corresponds to the Extended Common Data Representation encoding version 1.
- static final short **XML_DATA_REPRESENTATION** = 1

  Corresponds to the XML Data Representation (unsupported).
- static final short **XCDR2_DATA_REPRESENTATION** = 2

  Corresponds to the Extended Common Data Representation encoding version 2.
- static final short **AUTO_DATA_REPRESENTATION** = -1

  Representation is automatically chosen based on the type.

8.56.1 Detailed Description

This QoS policy contains a list of representation identifiers and compression settings used by **com.rti.dds.↩ publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities to negotiate which data representation and compression settings to use.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

See also

DATA_REPRESENTATION (p. 212)

This policy has request-offer semantics for both representation and compression. For representation, a **com.rti.dds.publication.DataWriter** (p. 553) may only offer a single representation. Attempting to put multiple representations in a **com.rti.dds.publication.DataWriter** (p. 553) will result in **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596). A **com.rti.dds.publication.DataWriter** (p. 553) will use its offered policy to communicate with its matched **com.rti.dds.subscription.DataReader** (p. 450) entities. A **com.rti.dds.subscription.DataReader** (p. 450) requests one or more representations. If a **com.rti.dds.publication.DataWriter** (p. 553) offers a representation that is contained within the sequence of the **com.rti.dds.subscription.DataReader** (p. 450), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When representations are specified in the **com.rti.dds.topic.TopicQos** (p. 1824), **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1484) copies the first element of the sequence, and **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1751) copies the whole sequence.

For Compression, only the **com.rti.dds.infrastructure.CompressionSettings_t.compression_ids** (p. 427) is propagated and a **com.rti.dds.publication.DataWriter** (p. 553) may only offer a single compression id. Attempting to put multiple compression_ids in a **com.rti.dds.publication.DataWriter** (p. 553) will result in **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

A **com.rti.dds.publication.DataWriter** (p. 553) will use its offered compression algorithm to compress data that it sends to its matched **com.rti.dds.subscription.DataReader** (p. 450) entities. A **com.rti.dds.subscription.DataReader** (p. 450) requests zero or more compression algorithms. If a **com.rti.dds.publication.DataWriter** (p. 553) offers a representation that is contained within the algorithms requested by the **com.rti.dds.subscription.DataReader** (p. 450), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When compression IDs are specified in the **com.rti.dds.topic.TopicQos** (p. 1824), **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1484) copies a single compression ID (see **com.rti.dds.infrastructure.CompressionSettings_t.compression_ids** (p. 427)), and **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1751) copies all of the compression_ids.

8.56.2 Member Data Documentation

8.56.2.1 value

```
final ShortSeq value = new ShortSeq()
```

Sequence of representation identifiers.

[default] For **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.DataWriter** (p. 553), and **com.rti.dds.subscription.DataReader** (p. 450), a list with one element: **com.rti.dds.infrastructure.DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION** (p. 214).

Note

An empty sequence is equivalent to a list with one element: **com.rti.dds.infrastructure.DataRepresentationQosPolicy.XCDR_DATA_REPRESENTATION** (p. 213).

8.56.2.2 compression_settings

```
CompressionSettings_t compression_settings = new CompressionSettings_t()
```

<<***extension***>> *(p. 155)* Structure that contains the compression settings.

compression_ids:
[default] The value depends on the entity
com.rti.dds.publication.DataWriter (p. 553) and **com.rti.dds.topic.Topic** (p. 1807) are set to com.rti.dds.↩ infrastructure.CompressionIdMaskBits.MASK_NONE
com.rti.dds.subscription.DataReader (p. 450) com.rti.dds.infrastructure.CompressionIdMaskBits.MASK_ALL

writer_compression_level:
[default] The value depends on the entity
com.rti.dds.publication.DataWriter (p. 553) and **com.rti.dds.topic.Topic** (p. 1807) are set to com.rti.dds.↩ infrastructure.COMPRESSION_LEVEL_DEFAULT
com.rti.dds.subscription.DataReader (p. 450) NOT SUPPORTED

writer_compression_threshold:
[default] The value depends on the entity
com.rti.dds.publication.DataWriter (p. 553) and **com.rti.dds.topic.Topic** (p. 1807) are set to com.rti.dds.↩ infrastructure.COMPRESSION_THRESHOLD_DEFAULT
com.rti.dds.subscription.DataReader (p. 450) NOT SUPPORTED

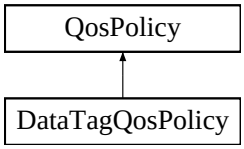
See also

com.rti.dds.infrastructure.CompressionSettings_t (p. 425)

8.57 DataTagQosPolicy Class Reference

Stores (name, value) pairs that can be used to determine access permissions.

Inheritance diagram for DataTagQosPolicy:



Public Attributes

- final **TagSeq tags** = new **TagSeq**()
  Sequence of data tags.

8.57.1 Detailed Description

Stores (name, value) pairs that can be used to determine access permissions.

Entity:

com.rti.dds.subscription.DataReader (p. 450) **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = N/A;
Changeable (p. 256) = **NO** (p. 256)

8.57.2 Usage

The DATA_TAG QoS policy can be used to associate a set of tags in the form of (name, value) pairs with a **com.rti.dds.subscription.DataReader** (p. 450) or **com.rti.dds.publication.DataWriter** (p. 553). This is similar to the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438), except for the following differences:

- Data tags are always propagated. You cannot select whether or not a particular pair should be propagated.
- Data tags are not exposed to API functions, such as **com.rti.dds.publication.DataWriter.get_matched_subscription_data** (p. 568), that receive **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) or **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) as a parameter.
- Connext passes data tags to the Access Control Security Plugin, which may use them to decide whether to allow or deny the corresponding entities.

There are helper functions to facilitate working with data tags. See the **DATA_TAG** (p. 214) page.

8.57.3 Member Data Documentation

8.57.3.1 tags

```
final TagSeq tags = new TagSeq()
```

Sequence of data tags.

[default] An empty list.

Referenced by **DataTagQosPolicyHelper.add_tag()**, **DataTagQosPolicyHelper.assert_tag()**, **DataTagQosPolicyHelper.get_number_of_tags()**, **DataTagQosPolicyHelper.lookup_tag()**, and **DataTagQosPolicyHelper.remove_tag()**.

8.58 DataTagQosPolicyHelper Class Reference

Policy helpers that facilitate management of the data tags in the input policy.

Static Public Member Functions

- static int **get_number_of_tags** (**DataTagQosPolicy** policy)

  Gets the number of data tags in the input policy.
- static void **assert_tag** (**DataTagQosPolicy** policy, String name, String value)

  Asserts the tag identified by name in the input policy.
- static void **add_tag** (**DataTagQosPolicy** policy, String name, String value)

  Adds a new tag to the input policy.
- static **Tag** **lookup_tag** (**DataTagQosPolicy** policy, String name)

  Searches by tag name for a tag in the input policy.
- static void **remove_tag** (**DataTagQosPolicy** policy, String name)

  Removes a tag from the input policy.

8.58.1 Detailed Description

Policy helpers that facilitate management of the data tags in the input policy.

8.58.2 Member Function Documentation

8.58.2.1 get_number_of_tags()

```
static int get_number_of_tags (
              DataTagQosPolicy policy )  [static]
```

Gets the number of data tags in the input policy.

Precondition

policy cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.

Returns

Number of data tags.

References **DataTagQosPolicy.tags**.

8.58.2.2 assert_tag()

```
static void assert_tag (
              DataTagQosPolicy policy,
             String name,
             String value )  [static]
```

Asserts the tag identified by name in the input policy.

If the tag already exists, this function replaces its current value with the new one.

If the tag identified by name does not exist, this function adds it to the tag set.

This function increases the maximum number of elements of the policy sequence by 10 when this number is not enough to store the new tag.

Precondition

policy, name and value cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* **Tag** (p. 1783) name.
value	<<***in***>> *(*p. *156)* **Tag** (p. 1783) value.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

References **DataTagQosPolicyHelper.lookup_tag()**, **DataTagQosPolicy.tags**, and **Tag.value**.

8.58.2.3 add_tag()

```
static void add_tag (
              DataTagQosPolicy policy,
```

```
            String name,
            String value )  [static]
```

Adds a new tag to the input policy.

This function will allocate memory to store the (name, value) pair. The memory allocated is owned by RTI Connext.

If the maximum number of elements of the policy sequence is not enough to store the new tag, this function will increase it by 10.

If the tag already exists, the function will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Precondition

policy, name and value cannot be null.

The tag is not in the policy.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* **Tag** (p. 1783) name.
value	<<***in***>> *(*p. *156)* **Tag** (p. 1783) value.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

References **DataTagQosPolicyHelper.lookup_tag()**, and **DataTagQosPolicy.tags**.

8.58.2.4 lookup_tag()

```
static  Tag lookup_tag (
             DataTagQosPolicy policy,
            String name )  [static]
```

Searches by tag name for a tag in the input policy.

Precondition

policy, name and value cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* **Tag** (p. 1783) name.

Returns

The function returns the first tag with the given name. If such a tag does not exist, the function returns NULL.

References **Tag.name**, and **DataTagQosPolicy.tags**.

Referenced by **DataTagQosPolicyHelper.add_tag()**, **DataTagQosPolicyHelper.assert_tag()**, and **DataTagQos↩ PolicyHelper.remove_tag()**.

8.58.2.5 remove_tag()

```
static void remove_tag (
              DataTagQosPolicy policy,
             String name )  [static]
```

Removes a tag from the input policy.

If the tag does not exist, the function fails with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Precondition

policy and name cannot be null.

The tag is in the policy.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* **Tag** (p. 1783) name.

Exceptions

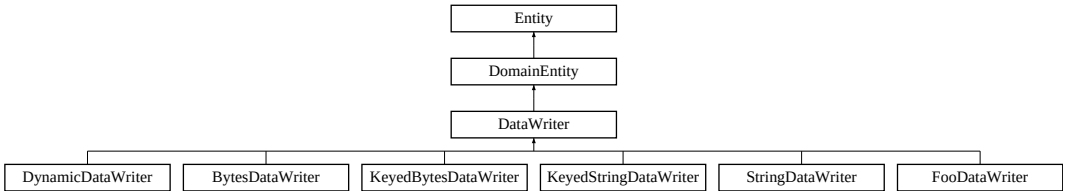
One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

References **DataTagQosPolicyHelper.lookup_tag()**, and **DataTagQosPolicy.tags**.

8.59 DataWriter Interface Reference

<<***interface***>> *(*p. *156)* Allows an application to set the value of the data to be published under a given **com.rti.dds.topic.Topic** (p. 1807).

Inheritance diagram for DataWriter:



Public Member Functions

- void **set_qos** (**DataWriterQos** qos)

  Sets the writer QoS.
- void **set_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Change the QoS of this writer using the input XML QoS profile.*
- void **get_qos** (**DataWriterQos** qos)

  Gets the writer QoS.
- void **set_listener** (**DataWriterListener** l, int mask)

  Sets the writer listener.
- **DataWriterListener get_listener** ()

  Get the writer listener.
- void **get_liveliness_lost_status** (**LivelinessLostStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS communication status.
- void **get_offered_deadline_missed_status** (**OfferedDeadlineMissedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS communication status.
- void **get_offered_incompatible_qos_status** (**OfferedIncompatibleQosStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS communication status.
- void **get_publication_matched_status** (**PublicationMatchedStatus** status)

  Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS communication status.
- void **get_reliable_writer_cache_changed_status** (**ReliableWriterCacheChangedStatus** status)

  <<***extension***>> *(*p. *155) Get the reliable cache status for this writer.*
- void **get_reliable_reader_activity_changed_status** (**ReliableReaderActivityChangedStatus** status)

  <<***extension***>> *(*p. *155) Get the reliable reader activity changed status for this writer.*
- void **get_datawriter_cache_status** (**DataWriterCacheStatus** status)

  <<***extension***>> *(*p. *155) Get the datawriter cache status for this writer.*
- void **get_datawriter_protocol_status** (**DataWriterProtocolStatus** status)

  <<***extension***>> *(*p. *155) Get the datawriter protocol status for this writer.*
- void **get_matched_subscription_datawriter_protocol_status** (**DataWriterProtocolStatus** status, **InstanceHandle_t** subscription_handle)

*<<**extension**>> (p. 155) Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.*

- void **get_matched_subscription_datawriter_protocol_status_by_locator** (**DataWriterProtocolStatus** status, **Locator_t** locator)

  *<<**extension**>> (p. 155) Get the datawriter protocol status for this writer, per matched subscription identified by the locator.*

- void **get_service_request_accepted_status** (**ServiceRequestAcceptedStatus** status)

  *Accesses the **com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1709) communication status.*

- void **get_matched_subscription_locators** (**LocatorSeq** locators)

  *<<**extension**>> (p. 155) Retrieve the list of locators for subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 553).*

- void **get_matched_subscriptions** (**InstanceHandleSeq** subscription_handles)

  *Retrieve the list of subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 553).*

- boolean **is_matched_subscription_active** (**InstanceHandle_t** subscription_handle)

  *Check if a subscription currently matched with a **DataWriter** (p. 553) is active.*

- void **get_matched_subscription_data** (**SubscriptionBuiltinTopicData** subscription_data, **InstanceHandle_t** subscription_handle)

  *This operation retrieves the information on a subscription that is currently "associated" with the **com.rti.dds.publication.DataWriter** (p. 553).*

- void **get_matched_subscription_participant_data** (**ParticipantBuiltinTopicData** participant_data, **InstanceHandle_t** subscription_handle)

  *This operation retrieves the information on the discovered **com.rti.dds.domain.DomainParticipant** (p. 670) associated with the subscription that is currently matching with the **com.rti.dds.publication.DataWriter** (p. 553).*

- **Topic get_topic** ()

  *This operation returns the **com.rti.dds.topic.Topic** (p. 1807) associated with the **com.rti.dds.publication.DataWriter** (p. 553).*

- **Publisher get_publisher** ()

  *This operation returns the **com.rti.dds.publication.Publisher** (p. 1466) to which the **com.rti.dds.publication.DataWriter** (p. 553) belongs.*

- void **wait_for_acknowledgments** (**Duration_t** max_wait)

  *Blocks the calling thread until all data written by reliable **com.rti.dds.publication.DataWriter** (p. 553) entity is acknowledged, or until timeout expires.*

- boolean **is_sample_app_acknowledged** (**SampleIdentity_t** identity)

  This method can be used to see if a sample has been application acknowledged.

- void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

  *<<**extension**>> (p. 155) Blocks the calling thread until asynchronous sending is complete.*

- void **assert_liveliness** ()

  *This operation manually asserts the liveliness of this **com.rti.dds.publication.DataWriter** (p. 553).*

- void **flush** ()

  *<<**extension**>> (p. 155) Flushes the batch in progress in the context of the calling thread.*

- **InstanceHandle_t register_instance_untyped** (Object instance_data)

  Register a new instance with this writer.

- **InstanceHandle_t register_instance_w_timestamp_untyped** (Object instance_data, **Time_t** source_timestamp)

  Register a new instance with this writer using the given time instead of the current time.

- void **unregister_instance_untyped** (Object instance_data, **InstanceHandle_t** handle)

  Unregister a new instance from this writer.

- void **unregister_instance_w_timestamp_untyped** (Object instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

Unregister a new instance from this writer using the given time instead of the current time.

- void **write_untyped** (Object instance_data, **InstanceHandle_t** handle)

  Publish a data sample.
- void **write_w_timestamp_untyped** (Object instance_data, **InstanceHandle_t** handle, **Time_t** source_↩ timestamp)

  Publish a data sample using the given time instead of the current time.
- void **dispose_untyped** (Object instance_data, **InstanceHandle_t** handle)

  Dispose a data sample.
- void **dispose_w_timestamp_untyped** (Object instance_data, **InstanceHandle_t** handle, **Time_t** source_↩ timestamp)

  Dispose a data sample using the given time instead of the current time.
- void **get_key_value_untyped** (Object key_holder, **InstanceHandle_t** handle)

  Fill in the key fields of the given data sample.
- **InstanceHandle_t lookup_instance_untyped** (Object key_value)

  Given a sample with the given key field values, return the handle corresponding to its instance.
- void **take_discovery_snapshot** ()

  Take a snapshot of the compatible and incompatible remote readers matched by a local writer.
- void **take_discovery_snapshot** (String file_name)

  Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

8.59.1 Detailed Description

<<***interface***>> *(*p. *156)* Allows an application to set the value of the data to be published under a given **com.rti.↩ dds.topic.Topic** (p. 1807).

QoS:

com.rti.dds.publication.DataWriterQos (p. 612)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.↩ LivelinessLostStatus** (p. 1242);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.↩ publication.OfferedDeadlineMissedStatus** (p. 1339);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.↩ publication.OfferedIncompatibleQosStatus** (p. 1340);
com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS, **com.rti.dds.publication.↩ PublicationMatchedStatus** (p. 1463);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **com.↩ rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1532);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS, **com.rti.↩ dds.publication.ReliableWriterCacheChangedStatus** (p. 1535).

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

A **com.rti.dds.publication.DataWriter** (p. 553) is attached to exactly one **com.rti.dds.publication.Publisher** (p. 1466), that acts as a factory for it.

A **com.rti.dds.publication.DataWriter** (p. 553) is bound to exactly one **com.rti.dds.topic.Topic** (p. 1807) and therefore to exactly one data type. The **com.rti.dds.topic.Topic** (p. 1807) must exist prior to the **com.rti.dds.publication.DataWriter** (p. 553)'s creation.

com.rti.dds.publication.DataWriter (p. 553) is an abstract class. It must be specialized for each particular application data-type (see **USER_DATA** (p. 278)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **com.rti.ndds.example.Foo** (p. 1066) are specified in the example type **com.rti.dds.publication.DataWriter** (p. 553).

The following operations may be called even if the **com.rti.dds.publication.DataWriter** (p. 553) is not enabled. Other operations will fail with **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) if called on a disabled **com.rti.dds.publication.DataWriter** (p. 553):

- The base-class operations **com.rti.dds.publication.DataWriter.set_qos** (p. 556), **com.rti.dds.publication.DataWriter.get_qos** (p. 558), **com.rti.dds.publication.DataWriter.set_listener** (p. 558), **com.rti.dds.publication.DataWriter.get_listener** (p. 559), **com.rti.dds.infrastructure.Entity.enable** (p. 1032), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034) and **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034)
- **com.rti.dds.publication.DataWriter.get_liveliness_lost_status** (p. 559), **com.rti.dds.publication.DataWriter.get_offered_deadline_missed_status** (p. 560), **com.rti.dds.publication.DataWriter.get_offered_incompatible_qos_status** (p. 560), **com.rti.dds.publication.DataWriter.get_publication_matched_status** (p. 561), **com.rti.dds.publication.DataWriter.get_reliable_writer_cache_changed_status** (p. 561), **com.rti.dds.publication.DataWriter.get_reliable_reader_activity_changed_status** (p. 561) **com.rti.dds.publication.DataWriter.get_service_request_accepted_status** (p. 565)

Several **com.rti.dds.publication.DataWriter** (p. 553) may operate in different threads. If they share the same **com.rti.dds.publication.Publisher** (p. 1466), the middleware guarantees that its operations are thread-safe.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

Operations Allowed in Listener Callbacks (p. 1238)

8.59.2 Member Function Documentation

8.59.2.1 set_qos()

```
void set_qos (
                DataWriterQos qos )
```

Sets the writer QoS.

This operation modifies the QoS of the **com.rti.dds.publication.DataWriter** (p. 553).

The **com.rti.dds.publication.DataWriterQos.user_data** (p. 619), **com.rti.dds.publication.DataWriterQos.deadline** (p. 617), **com.rti.dds.publication.DataWriterQos.latency_budget** (p. 618), **com.rti.dds.publication.DataWriterQos.ownership_strength** (p. 619), **com.rti.dds.publication.DataWriterQos.transport_priority** (p. 619), **com.rti.dds.publication.DataWriterQos.lifespan** (p. 619) and **com.rti.dds.publication.DataWriterQos.writer_data_lifecycle** (p. 619) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(*p. *156)* The **com.rti.dds.publication.DataWriterQos** (p. 612) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.publication.DataWriter** (p. 553) is enabled. The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) can be used to indicate that the QoS of the **com.rti.dds.publication.DataWriter** (p. 553) should be changed to match the current default **com.rti.dds.publication.DataWriterQos** (p. 612) set in the **com.rti.dds.publication.Publisher** (p. 1466). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

set_qos (abstract) (p. 1030)

Operations Allowed in Listener Callbacks (p. 1238)

8.59.2.2 set_qos_with_profile()

```
void set_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(*p. *155)* Change the QoS of this writer using the input XML QoS profile.

This operation modifies the QoS of the **com.rti.dds.publication.DataWriter** (p. 553).

The **com.rti.dds.publication.DataWriterQos.user_data** (p. 619), **com.rti.dds.publication.DataWriterQos.deadline** (p. 617), **com.rti.dds.publication.DataWriterQos.latency_budget** (p. 618), **com.rti.dds.publication.DataWriterQos.ownership_strength** (p. 619), **com.rti.dds.publication.DataWriterQos.transport_priority** (p. 619), **com.rti.dds.publication.DataWriterQos.lifespan** (p. 619) and **com.rti.dds.publication.DataWriterQos.writer_data_lifecycle** (p. 619) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.publication.Publisher.set_default_library** (p. 1478)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1238)

8.59.2.3 get_qos()

```
void get_qos (
              DataWriterQos qos )
```

Gets the writer QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 156)* The **com.rti.dds.publication.DataWriterQos** (p. 612) to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.59.2.4 set_listener()

```
void set_listener (
              DataWriterListener l,
             int mask )
```

Sets the writer listener.

Parameters

l	*<<**in**>> (p. 156)* **com.rti.dds.publication.DataWriterListener** (p. 589) to set to
mask	*<<**in**>> (p. 156)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.publication.DataWriterListener** (p. 589).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.59.2.5 get_listener()

```
DataWriterListener get_listener ( )
```

Get the writer listener.

Returns

com.rti.dds.publication.DataWriterListener (p. 589) of the **com.rti.dds.publication.DataWriter** (p. 553).

See also

get_listener (abstract) (p. 1032)

8.59.2.6 get_liveliness_lost_status()

```
void get_liveliness_lost_status (
                LivelinessLostStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	*<<**inout**>> (p. 156)* **com.rti.dds.publication.LivelinessLostStatus** (p. 1242) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.7 get_offered_deadline_missed_status()

```
void get_offered_deadline_missed_status (
            OfferedDeadlineMissedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	*<<**inout**>> (p. 156)* **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1339) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.8 get_offered_incompatible_qos_status()

```
void get_offered_incompatible_qos_status (
            OfferedIncompatibleQosStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	*<<**inout**>> (p. 156)* **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1340) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.9 get_publication_matched_status()

```
void get_publication_matched_status (
            PublicationMatchedStatus status )
```

Accesses the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(*p. *156)* **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.10 get_reliable_writer_cache_changed_status()

```
void get_reliable_writer_cache_changed_status (
            ReliableWriterCacheChangedStatus status )
```

<<***extension***>> *(*p. *155)* Get the reliable cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(*p. *156)* **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1535) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.11 get_reliable_reader_activity_changed_status()

```
void get_reliable_reader_activity_changed_status (
            ReliableReaderActivityChangedStatus status )
```

<<***extension***>> *(p. 155)* Get the reliable reader activity changed status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1532) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.12 get_datawriter_cache_status()

```
void get_datawriter_cache_status (
            DataWriterCacheStatus status )
```

<<***extension***>> *(p. 155)* Get the datawriter cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.DataWriterCacheStatus** (p. 587) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.59.2.13 get_datawriter_protocol_status()

```
void get_datawriter_protocol_status (
            DataWriterProtocolStatus status )
```

<<***extension***>> *(*p. *155)* Get the datawriter protocol status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 601) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.59.2.14 get_matched_subscription_datawriter_protocol_status()

```
void get_matched_subscription_datawriter_protocol_status (
              DataWriterProtocolStatus status,
              InstanceHandle_t subscription_handle )
```

<<***extension***>> *(p. 155)* Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 601) to be filled in. Cannot be NULL.
subscription_handle	<<***in***>> *(p. 156)* Handle to a specific subscription associated with the **com.rti.dds.subscription.DataReader** (p. 450). Cannot be NULL. Must correspond to a subscription currently associated with the **com.rti.dds.publication.DataWriter** (p. 553).

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.59.2.15 get_matched_subscription_datawriter_protocol_status_by_locator()

```
void get_matched_subscription_datawriter_protocol_status_by_locator (
              DataWriterProtocolStatus status,
              Locator_t locator )
```

<<***extension***>> *(p. 155)* Get the datawriter protocol status for this writer, per matched subscription identified by the locator.

This also resets the status so that it is no longer considered changed.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 601) to be filled in Cannot be NULL.
locator	<<***in***>> *(p. 156)* Locator to a specific locator associated with the **com.rti.dds.subscription.DataReader** (p. 450). Cannot be NULL. Must correspond to a locator of one or more subscriptions currently associated with the **com.rti.dds.publication.DataWriter** (p. 553).

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.59.2.16 get_service_request_accepted_status()

```
void get_service_request_accepted_status (
              ServiceRequestAcceptedStatus status )
```

Accesses the **com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1709) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.publication.ServiceRequestAcceptedStatus** (p. 1677) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.17 get_matched_subscription_locators()

```
void get_matched_subscription_locators (
              LocatorSeq locators )
```

<<***extension***>> *(p. 155)* Retrieve the list of locators for subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 553).

The locators returned in the `locators` list are the ones that are used by the DDS implementation to communicate with the corresponding matched **com.rti.dds.subscription.DataReader** (p. 450) entities.

Parameters

locators	<<***inout***>> *(p. 156)*. Handles of all the matched subscription locators.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all the matches and the system can not resize the sequence, this method will fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Cannot be NULL. .

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) if the sequence is too small and the system can not resize it, or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.59.2.18 get_matched_subscriptions()

```
void get_matched_subscriptions (
               InstanceHandleSeq subscription_handles )
```

Retrieve the list of subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 553).

A subscription is considered to be matching if all of the following criteria are true:

- The subscription is within the same domain as this publication.
- The subscription has a matching **com.rti.dds.topic.Topic** (p. 1807).
- The subscription has compatible QoS.
- If the applications are using partitions, the subscription shares a common partition with this publication.
- The **com.rti.dds.domain.DomainParticipant** (p. 670) has not indicated that the subscription's **com.rti.dds.domain.DomainParticipant** (p. 670) should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 725) API.
- If the publication is using the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) and the subscription is using a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), there is an intersection between at least one of the associated filters.
- If the endpoints need to exchange key material to communicate (i.e., they are securing their communications), the writer has completed the key exchange with reader.

The handles returned in the `subscription_handles` list are the ones that RTI Connext uses to locally identify the corresponding matched **com.rti.dds.subscription.DataReader** (p. 450) entities. These handles match the ones that appear in the **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1640) field of the **com.rti.dds.↩ subscription.SampleInfo** (p. 1634) when reading the **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopic↩ DataTypeSupport.SUBSCRIPTION_TOPIC_NAME** (p. 164) builtin topic.

This API may return the subscription handles of subscriptions that are inactive. **com.rti.dds.publication.DataWriter.↩ is_matched_subscription_active** (p. 567) can be used to check this.

Parameters

subscription_handles	<<***inout***>> *(p. 156)*. The handles of all the matched subscriptions.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all of the matches and the system cannot resize the sequence, this method will fail with **com.rti.dds.infrastructure.RETCODE_OUT_↩ OF_RESOURCES** (p. 1598).

The maximum number of matches possible is configured with **com.rti.dds.infrastructure.DomainParticipant↩ ResourceLimitsQosPolicy** (p. 803). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL. .

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) if the sequence is too small and the system cannot resize it, or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.59.2.19 is_matched_subscription_active()

```
boolean is_matched_subscription_active (
                InstanceHandle_t subscription_handle )
```

Check if a subscription currently matched with a **DataWriter** (p. 553) is active.

This API is used for querying the endpoint liveliness of a matched subscription. A matched subscription will be marked as inactive when it becomes nonprogessing (e.g., not responding to a **DataWriter** (p. 553)'s heartbeats, or letting its internal queue fill up without taking the available data). Note that if the participant associated with the matched subscription loses liveliness, the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) will become invalid and this function will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Parameters

subscription_handle	<<***in***>> *(p. 156)* The **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of the matched subscription. See **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566) for a description of what is considered a matched subscription.

Returns

A boolean indicating whether or not the matched subscription is active.

8.59.2.20 get_matched_subscription_data()

```
void get_matched_subscription_data (
                SubscriptionBuiltinTopicData subscription_data,
                InstanceHandle_t subscription_handle )
```

This operation retrieves the information on a subscription that is currently "associated" with the **com.rti.dds.↩ publication.DataWriter** (p. 553).

The `subscription_handle` must correspond to a subscription currently associated with the **com.rti.dds.↩ publication.DataWriter** (p. 553). Otherwise, the operation will fail and fail with **com.rti.dds.infrastructure.↩ RETCODE_BAD_PARAMETER** (p. 1594). Use **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566) to find the subscriptions that are currently matched with the **com.rti.dds.publication.DataWriter** (p. 553).

The above information is also available through **com.rti.dds.subscription.DataReaderListener::on_data_available()** (p. 500) (if a reader listener is installed on the builtin.SubscriptionBuiltinTopicDataDataReader).

When the subscription data is updated, for example when the content filter property changes, there is a small window of time in between when the **DataWriter** (p. 553) is made aware of these changes and when they actually take effect. Taking effect in this example means that the **DataWriter** (p. 553) will perform writer-side filtering using the new filter property values (filter expression and/or parameters).

When the **DataWriter** (p. 553) is made aware of the changes they will first be seen in the **com.rti.dds.subscription.↩ DataReaderListener::on_data_available()** (p. 500) of the builtin.SubscriptionBuiltinTopicDataDataReader. When these changes are applied, they will be seen in the output of this API because this API blocks until the most recent changes known to the **DataWriter** (p. 553) have taken effect. This API will only block when called outside of a listener callback, in order to not block the internal threads from making progress.

If application behavior depends on being made aware of information about a subscription only after it has taken effect on the **DataWriter** (p. 553), the recommended pattern for usage of this API is to wait for subscription data to be received either through polling this API or by installing a listener on the builtin.SubscriptionBuiltinTopicDataDataReader. When a new sample is received by the builtin DataReader, this API may be called in a separate thread and will return the expected matched subscription data once it has been applied to the **DataWriter** (p. 553).

Because this API blocks, it is possible for this API to time out while waiting for the changes to be applied. A timeout may happen if the DataReader's subscription data is changing rapidly, preventing the **DataWriter** (p. 553) from returning valid information before newer data has been received, or if an application is performing a task in a listener callback, thereby preventing the middleware's threads from executing events in a timely manner.

Note: This operation does not retrieve the builtin.SubscriptionBuiltinTopicData.type_code. This information is available through **com.rti.dds.subscription.DataReaderListener::on_data_available()** (p. 500) (if a reader listener is installed on the builtin.SubscriptionBuiltinTopicDataDataReader).

Parameters

subscription_data	*<<**inout**>> (p. 156).* The information to be filled in on the associated subscription. Cannot be NULL.
subscription_handle	*<<**in**>> (p. 156).* Handle to a specific subscription associated with the **com.rti.dds.subscription.DataReader** (p. 450). Must correspond to a subscription currently associated with the **com.rti.dds.publication.DataWriter** (p. 553). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597), or **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)

8.59.2.21 get_matched_subscription_participant_data()

```
void get_matched_subscription_participant_data (
               ParticipantBuiltinTopicData participant_data,
               InstanceHandle_t subscription_handle )
```

This operation retrieves the information on the discovered **com.rti.dds.domain.DomainParticipant** (p. 670) associated with the subscription that is currently matching with the **com.rti.dds.publication.DataWriter** (p. 553).

The `subscription_handle` must correspond to a subscription currently associated with the **com.rti.dds.↩ publication.DataWriter** (p. 553). Otherwise, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD↩ _PARAMETER** (p. 1594). The operation may also fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION↩ _NOT_MET** (p. 1598) if the subscription corresponds to the same **com.rti.dds.domain.DomainParticipant** (p. 670) that the **DataWriter** (p. 553) belongs to. Use **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566) to find the subscriptions that are currently matched with the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

participant_data	*<<**inout**>> (p. 156).* The information to be filled in on the associated participant Cannot be NULL.
subscription_handle	*<<**in**>> (p. 156).* Handle to a specific subscription associated with the **com.rti.dds.subscription.DataReader** (p. 450). Must correspond to a subscription currently associated with the **com.rti.dds.publication.DataWriter** (p. 553).

Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.59.2.22 get_topic()

```
Topic get_topic ( )
```

This operation returns the **com.rti.dds.topic.Topic** (p. 1807) associated with the **com.rti.dds.publication.DataWriter** (p. 553).

This is the same **com.rti.dds.topic.Topic** (p. 1807) that was used to create the **com.rti.dds.publication.DataWriter** (p. 553).

Returns

com.rti.dds.topic.Topic (p. 1807) that was used to create the **com.rti.dds.publication.DataWriter** (p. 553).

8.59.2.23 get_publisher()

```
Publisher get_publisher ( )
```

This operation returns the **com.rti.dds.publication.Publisher** (p. 1466) to which the **com.rti.dds.publication.Data↩ Writer** (p. 553) belongs.

Returns

com.rti.dds.publication.Publisher (p. 1466) to which the **com.rti.dds.publication.DataWriter** (p. 553) belongs.

8.59.2.24 wait_for_acknowledgments()

```
void wait_for_acknowledgments (
              Duration_t max_wait )
```

Blocks the calling thread until all data written by reliable **com.rti.dds.publication.DataWriter** (p. 553) entity is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **com.rti.dds.publication.Data↩ Writer** (p. 553) entity is acknowledged by (a) all reliable **com.rti.dds.subscription.DataReader** (p. 450) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged by all reliable matched data readers and by all required subscriptions; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to wait_for_acknowledgments on a **DataWriter** (p. 553) and a different thread writes new samples on the same **DataWriter** (p. 553), the new samples must be acknowledged before unblocking the thread waiting on wait_for_acknowledgments.

If the **com.rti.dds.publication.DataWriter** (p. 553) does not have **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) kind set to RELIABLE, this operation will succeed immediately.

Parameters

max_wait	*<<**in**>> (p. 156)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 840) .

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)

8.59.2.25 is_sample_app_acknowledged()

```
boolean is_sample_app_acknowledged (
              SampleIdentity_t identity )
```

This method can be used to see if a sample has been application acknowledged.

This method can be used to see if a sample has been application acknowledged by all the matching DataReaders that were alive when the sample was written.

If a DataReader does not enable application acknowledgment (by setting **com.rti.dds.infrastructure.Reliability↩ QosPolicy.acknowledgment_kind** (p. 1529) to a value other than **com.rti.dds.infrastructure.ReliabilityQosPolicy↩ AcknowledgmentModeKind.PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1530)), the sample is considered application acknowledged for that DataReader.

Parameters

identity	*<<**in**>> (p. 156)* Sample identity.

Returns

true when the sample has been application acknowledged. Otherwise, false.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.59.2.26 wait_for_asynchronous_publishing()

```
void wait_for_asynchronous_publishing (
              Duration_t max_wait )
```

<<***extension***>> *(*p. *155)* Blocks the calling thread until asynchronous sending is complete.

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **com.rti.dds.publication.DataWriter** (p. 553) is sent and acknowledged (if reliable) by all matched **com.rti.dds.subscription.DataReader** (p. 450) entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **com.rti.dds.subscription.DataReader** (p. 450) is complete in addition to what **com.rti.dds.publication.DataWriter.wait_for_acknowledgments** (p. 570) provides.

If the **com.rti.dds.publication.DataWriter** (p. 553) does not have **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1496) kind set to com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS the operation will complete immediately with com.rti.dds.infrastructure.RETCODE_OK.

Parameters

max_wait	<<***in***>> *(*p. *156)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 840) .

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)

8.59.2.27 assert_liveliness()

```
void assert_liveliness ( )
```

This operation manually asserts the liveliness of this **com.rti.dds.publication.DataWriter** (p. 553).

This is used in combination with the **LIVELINESS** (p. 239) policy to indicate to RTI Connext that the **com.rti.dds.publication.DataWriter** (p. 553) remains active.

You only need to use this operation if the **LIVELINESS** (p. 239) setting is either com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS or com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS. Otherwise, it has no effect.

Note: writing data via the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) or **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 553) itself, and its **com.rti.dds.domain.DomainParticipant** (p. 670). Consequently the use of **assert_liveliness()** (p. 572) is only needed if the application is not writing data regularly.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1243)

8.59.2.28 flush()

```
void flush ( )
```

<<***extension***>> *(p. 155)* Flushes the batch in progress in the context of the calling thread.

After being flushed, the batch is available to be sent on the network.

If the **com.rti.dds.publication.DataWriter** (p. 553) does not have **com.rti.dds.infrastructure.PublishModeQos↩ Policy** (p. 1496) kind set to com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.↩ ASYNCHRONOUS_PUBLISH_MODE_QOS, the batch will be sent on the network immediately (in the context of the calling thread).

If the **com.rti.dds.publication.DataWriter** (p. 553) does have **com.rti.dds.infrastructure.PublishModeQos↩ Policy** (p. 1496) kind set to com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.↩ ASYNCHRONOUS_PUBLISH_MODE_QOS, the batch will be sent in the context of the asynchronous publishing thread.

This operation may block in the same conditions as **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If this operation does block, the RELIABILITY max_blocking_time configures the maximum time the write operation may block (waiting for space to become available). If max_blocking_time elapses before the DDS_DataWriter is able to store the modification without exceeding the limits, the operation will fail with DDS_RETCODE_TIMEOUT.

MT Safety:

flush() (p. 573) is only thread-safe with batching if **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_↩ write** (p. 359) is TRUE.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.59.2.29 register_instance_untyped()

```
InstanceHandle_t register_instance_untyped (
            Object instance_data )
```

Register a new instance with this writer.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.unregister_instance_untyped (p. 574)

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

Implemented in **DynamicDataWriter** (p. 1005).

Referenced by **FooDataWriter.register_instance()**, **KeyedBytesDataWriter.register_instance()**, and **KeyedStringDataWriter.register_instance()**.

8.59.2.30 register_instance_w_timestamp_untyped()

```
InstanceHandle_t register_instance_w_timestamp_untyped (
            Object instance_data,
             Time_t source_timestamp )
```

Register a new instance with this writer using the given time instead of the current time.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp** (p. 1099) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.unregister_instance_w_timestamp_untyped (p. 575)

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

Referenced by **FooDataWriter.register_instance_w_timestamp()**, **KeyedBytesDataWriter.register_instance_w_timestamp()**, and **KeyedStringDataWriter.register_instance_w_timestamp()**.

8.59.2.31 unregister_instance_untyped()

```
void unregister_instance_untyped (
             Object instance_data,
              InstanceHandle_t handle )
```

Unregister a new instance from this writer.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.register_instance_untyped (p. 573)

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

Implemented in **DynamicDataWriter** (p. 1009).

Referenced by **FooDataWriter.unregister_instance()**, **KeyedBytesDataWriter.unregister_instance()**, and **KeyedStringDataWriter.unregister_instance()**.

8.59.2.32 unregister_instance_w_timestamp_untyped()

```
void unregister_instance_w_timestamp_untyped (
             Object instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Unregister a new instance from this writer using the given time instead of the current time.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_↩ instance_w_timestamp method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.register_instance_w_timestamp_untyped (p. 574)

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

Referenced by **FooDataWriter.unregister_instance_w_timestamp()**, **KeyedBytesDataWriter.unregister_↩ instance_w_timestamp()**, and **KeyedStringDataWriter.unregister_instance_w_timestamp()**.

8.59.2.33 write_untyped()

```
void write_untyped (
             Object instance_data,
              InstanceHandle_t handle )
```

Publish a data sample.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.write** (p. 1105) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.write_w_timestamp_untyped (p. 576)

com.rti.ndds.example.FooDataWriter.write (p. 1105)

Implemented in **DynamicDataWriter** (p. 1014).

Referenced by **BytesDataWriter.write()**, **FooDataWriter.write()**, **KeyedBytesDataWriter.write()**, **KeyedString↩ DataWriter.write()**, and **StringDataWriter.write()**.

8.59.2.34 write_w_timestamp_untyped()

```
void write_w_timestamp_untyped (
             Object instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Publish a data sample using the given time instead of the current time.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.write_untyped (p. 575)

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

Referenced by **BytesDataWriter.write_w_timestamp()**, **FooDataWriter.write_w_timestamp()**, **KeyedBytesData↩ Writer.write_w_timestamp()**, **KeyedStringDataWriter.write_w_timestamp()**, and **StringDataWriter.write_w_↩ timestamp()**.

8.59.2.35 dispose_untyped()

```
void dispose_untyped (
             Object instance_data,
              InstanceHandle_t handle )
```

Dispose a data sample.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.dispose_w_timestamp_untyped (p. 577)

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

Referenced by **FooDataWriter.dispose()**, **KeyedBytesDataWriter.dispose()**, and **KeyedStringDataWriter.dispose()**.

8.59.2.36 dispose_w_timestamp_untyped()

```
void dispose_w_timestamp_untyped (
             Object instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Dispose a data sample using the given time instead of the current time.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.dispose_w_timestamp** (p. 1113) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.dispose_untyped (p. 576)

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

Implemented in **DynamicDataWriter** (p. 1019).

Referenced by **FooDataWriter.dispose_w_timestamp()**, **KeyedBytesDataWriter.dispose_w_timestamp()**, and **KeyedStringDataWriter.dispose_w_timestamp()**.

8.59.2.37 get_key_value_untyped()

```
void get_key_value_untyped (
             Object key_holder,
              InstanceHandle_t handle )
```

Fill in the key fields of the given data sample.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114) method instead of this one. See that method for detailed documentation.

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

Referenced by **FooDataWriter.get_key_value()**, **KeyedBytesDataWriter.get_key_value()**, and **KeyedStringDataWriter.get_key_value()**.

8.59.2.38 lookup_instance_untyped()

```
InstanceHandle_t lookup_instance_untyped (
             Object key_value )
```

Given a sample with the given key field values, return the handle corresponding to its instance.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.lookup_instance** (p. 1115) method instead of this one. See that method for detailed documentation.

See also

com.rti.ndds.example.FooDataWriter.lookup_instance (p. 1115)

Referenced by **FooDataWriter.lookup_instance()**, **KeyedBytesDataWriter.lookup_instance()**, and **KeyedStringDataWriter.lookup_instance()**.

8.59.2.39 take_discovery_snapshot() [1/2]

```
void take_discovery_snapshot ( )
```

Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

The snapshot will be printed through the **com.rti.ndds.config.Logger** (p. 1267). A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udpv4://192.168.1.170:7411"

reason="Inconsistent QoS"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 261).

8.59.2.40 take_discovery_snapshot() [2/2]

```
void take_discovery_snapshot (
             String file_name )
```

Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote readers that match the local writer domain=0 name="writerTestName"

guid="0x0101D8D1,0x20B83C0D,0x4495246E:0x80000003"

topic="FooTopic" type="FooType"

------------------------------------------------------------------------

Compatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000004 name="reader1TestName"
```

```
kind="unkeyed user datareader"
unicastLocators="udpv4://192.168.1.170:7411"
liveliness="ALIVE"
Incompatible readers:
1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"
kind="unkeyed user datareader"
unicastLocators="udpv4://192.168.1.170:7411"
reason="Inconsistent QoS"
------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(*p. *156)* Name of the file where snapshot should be printed.

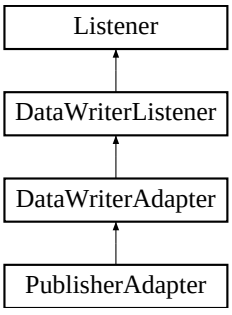
Exceptions

One	of the **Standard Return Codes** (p. 261).

8.60 DataWriterAdapter Class Reference

<<***extension***>> *(*p. *155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods or functions.)

Inheritance diagram for DataWriterAdapter:



Public Member Functions

- void **on_offered_deadline_missed** (**DataWriter** writer, **OfferedDeadlineMissedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.
- void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.
- void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.

- void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.
- void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  *<<**extension**>> (p. 155) A change has occurred in the writer's cache of unacknowledged samples.*
- void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  *<<**extension**>> (p. 155) A matched reliable reader has become active or become inactive.*
- void **on_sample_removed** (**DataWriter** writer, **Cookie_t** cookie)

  *<<**extension**>> (p. 155) Called when a sample is removed from the **DataWriter** (p. 553) queue.*
- void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Notifies when an instance is replaced in **DataWriter** (p. 553) queue.*
- void **on_application_acknowledgment** (**DataWriter** writer, **AcknowledgmentInfo** ackInfo)

  *<<**extension**>> (p. 155) Called when a sample is application-acknowledged*
- void **on_service_request_accepted** (**DataWriter** writer, **ServiceRequestAcceptedStatus** status)

  *<<**extension**>> (p. 155) Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.↩ subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.*

8.60.1 Detailed Description

*<<**extension**>> (p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods or functions.)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.60.2 Member Function Documentation

8.60.2.1 on_offered_deadline_missed()

```
void on_offered_deadline_missed (
              DataWriter writer,
              OfferedDeadlineMissedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **DEADLINE** (p. 217) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **com.rti.dds.publication.DataWriter** (p. 553) failed to provide data for an instance.

Parameters

writer	*<<**out**>> (p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	*<<**out**>> (p. 156)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 591).

8.60.2.2 on_offered_incompatible_qos()

```
void on_offered_incompatible_qos (
            DataWriter writer,
            OfferedIncompatibleQosStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 612) of the **com.rti.dds.publication.↩ DataWriter** (p. 553) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 450). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 553) has discovered a **com.rti.dds.↩ subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 591).

8.60.2.3 on_liveliness_lost()

```
void on_liveliness_lost (
            DataWriter writer,
            LivelinessLostStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **LIVELINESS** (p. 239) qos policy was not respected; this **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 553) simply remains not alive for another liveliness period.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 592).

8.60.2.4 on_publication_matched()

```
void on_publication_matched (
              DataWriter writer,
              PublicationMatchedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 553) has found a **com.rti.dds.subscription.↩ DataReader** (p. 450) that matches the **com.rti.dds.topic.Topic** (p. 1807), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 450) that was previously considered to be matched.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current publication match status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 592).

8.60.2.5 on_reliable_writer_cache_changed()

```
void on_reliable_writer_cache_changed (
              DataWriter writer,
              ReliableWriterCacheChangedStatus status )
```

<<***extension***>> *(p. 155)* A change has occurred in the writer's cache of unacknowledged samples.

This listener callback is triggered when:

- The cache is empty (contains no unacknowledged samples).
- The cache is full (the number of unacknowledged samples has reached the value specified in **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)).
- The number of unacknowledged samples has reached **com.rti.dds.infrastructure.RtpsReliableWriter↩ Protocol_t.high_watermark** (p. 1607) or **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_↩ watermark** (p. 1607).

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 593).

8.60.2.6 on_reliable_reader_activity_changed()

```
void on_reliable_reader_activity_changed (
            DataWriter writer,
            ReliableReaderActivityChangedStatus status )
```

<<***extension***>> *(*p. *155)* A matched reliable reader has become active or become inactive.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 593).

8.60.2.7 on_sample_removed()

```
void on_sample_removed (
            DataWriter writer,
            Cookie_t cookie )
```

<<***extension***>> *(*p. *155)* Called when a sample is removed from the **DataWriter** (p. 553) queue.

This callback is called only if the sample was written with a **com.rti.dds.infrastructure.Cookie_t** (p. 442) with **com.↩ rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), or if this writer uses **Zero Copy** (p. 56) transfer over shared memory" or **FlatData Topic-Types** (p. 55) "FlatData language binding".

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback

Parameters

cookie	<<***out***>> *(*p. *156)* • If this sample was written with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), this field contains a copy of the cookie set in **com.rti.dds.infrastructure.WriteParams_t** (p. 1994). • If this writer uses **Zero Copy transfer over shared memory** (p. 56) or **FlatData language binding** (p. 55), this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using com.rti.dds.infrastructure.Cookie_t.to_pointer

See also

com.rti.ndds.example.FooDataWriter.get_loan

Implements **DataWriterListener** (p. 594).

8.60.2.8 on_instance_replaced()

```
void on_instance_replaced (
              DataWriter writer,
              InstanceHandle_t handle )
```

<<***extension***>> *(*p. *155)* Notifies when an instance is replaced in **DataWriter** (p. 553) queue.

This callback is called when an instance is replaced by the **com.rti.dds.publication.DataWriter** (p. 553) due to instance resource limits being reached. This callback returns to the user the handle of the replaced instance, which can be used to get the key of the replaced instance using the **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114) API.

Because this callback can be called within the context of an in-progress write, dispose, or unregister call, most APIs on the **DataWriter** (p. 553) must not be used. The only **DataWriter** (p. 553) APIs that are safe to call within this callback are:

- **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114)
- com.rti.ndds.example.FooDataWriter.create_data
- com.rti.ndds.example.FooDataWriter.delete_data
- **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566)
- **com.rti.dds.publication.DataWriter.is_matched_subscription_active** (p. 567)
- **com.rti.dds.publication.DataWriter.get_matched_subscription_participant_data** (p. 569)
- **com.rti.dds.publication.DataWriter.get_topic** (p. 569)
- **com.rti.dds.publication.DataWriter.get_publisher** (p. 570)
- **com.rti.dds.publication.DataWriter.is_sample_app_acknowledged** (p. 571)

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
handle	<<***out***>> *(*p. *156)* Handle of the replaced instance

Implements **DataWriterListener** (p. 594).

8.60.2.9 on_application_acknowledgment()

```
void on_application_acknowledgment (
            DataWriter writer,
            AcknowledgmentInfo info )
```

<<***extension***>> *(*p. *155)* Called when a sample is application-acknowledged

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **com.rti.dds.subscription.DataReader** (p. 450). Also provides user-specified response data sent from the **com.rti.dds.subscription.DataReader** (p. 450) by the acknowledgment message.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
info	<<***out***>> *(*p. *156)* **com.rti.dds.publication.AcknowledgmentInfo** (p. 330) of the acknowledged sample

Implements **DataWriterListener** (p. 595).

8.60.2.10 on_service_request_accepted()

```
void on_service_request_accepted (
            DataWriter writer,
            ServiceRequestAcceptedStatus status )
```

<<***extension***>> *(*p. *155)* Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current service request accepted status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

See also

Topic Queries (p. 84)

Implements **DataWriterListener** (p. 596).

8.61 DataWriterCacheStatus Class Reference

<<***extension***>> *(p. 155)* The status of the **DataWriter** (p. 553)'s cache. Provides information on cache related metrics such as the number of samples and instances in the **DataWriter** (p. 553) queue.

Inherits Status.

Public Attributes

- long **sample_count_peak**

  *The highest value of **com.rti.dds.publication.DataWriterCacheStatus.sample_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).*
- long **sample_count**

  *The number of samples in the **DataWriter** (p. 553)'s queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.*
- long **alive_instance_count**

  *The number of instances currently in the **DataWriter** (p. 553)'s queue that have an instance_state equal to **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161).*
- long **alive_instance_count_peak**

  *The highest value of **com.rti.dds.publication.DataWriterCacheStatus.alive_instance_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).*
- long **disposed_instance_count**

  *The number of instances currently in the **DataWriter** (p. 553)'s queue that have an instance_state equal to **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) (due to, for example, being disposed via the **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) operation).*
- long **disposed_instance_count_peak**

  *The highest value of **com.rti.dds.publication.DataWriterCacheStatus.disposed_instance_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).*
- long **unregistered_instance_count**

  *The number of instances currently in the **DataWriter** (p. 553)'s queue that the **DataWriter** (p. 553) has unregistered from via the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.*
- long **unregistered_instance_count_peak**

  *The highest value of **com.rti.dds.publication.DataWriterCacheStatus.unregistered_instance_count** (p. 589) over the lifetime of the **DataWriter** (p. 553).*

8.61.1 Detailed Description

<<***extension***>> *(*p. *155)* The status of the **DataWriter** (p. 553)'s cache. Provides information on cache related metrics such as the number of samples and instances in the **DataWriter** (p. 553) queue.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

8.61.2 Member Data Documentation

8.61.2.1 sample_count_peak

```
long sample_count_peak
```

The highest value of **com.rti.dds.publication.DataWriterCacheStatus.sample_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).

8.61.2.2 sample_count

```
long sample_count
```

The number of samples in the **DataWriter** (p. 553)'s queue. This statistic includes meta-samples that represent the unregistration or disposal of an instance.

8.61.2.3 alive_instance_count

```
long alive_instance_count
```

The number of instances currently in the **DataWriter** (p. 553)'s queue that have an instance_state equal to **com.rti.↩ dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161).

8.61.2.4 alive_instance_count_peak

```
long alive_instance_count_peak
```

The highest value of **com.rti.dds.publication.DataWriterCacheStatus.alive_instance_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).

8.61.2.5 disposed_instance_count

```
long disposed_instance_count
```

The number of instances currently in the **DataWriter** (p. 553)'s queue that have an instance_state equal to **com.↩ rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) (due to, for example, being disposed via the **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) operation).

8.61.2.6 disposed_instance_count_peak

```
long disposed_instance_count_peak
```

The highest value of **com.rti.dds.publication.DataWriterCacheStatus.disposed_instance_count** (p. 588) over the lifetime of the **DataWriter** (p. 553).

8.61.2.7 unregistered_instance_count

```
long unregistered_instance_count
```

The number of instances currently in the **DataWriter** (p. 553)'s queue that the **DataWriter** (p. 553) has unregistered from via the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.

8.61.2.8 unregistered_instance_count_peak

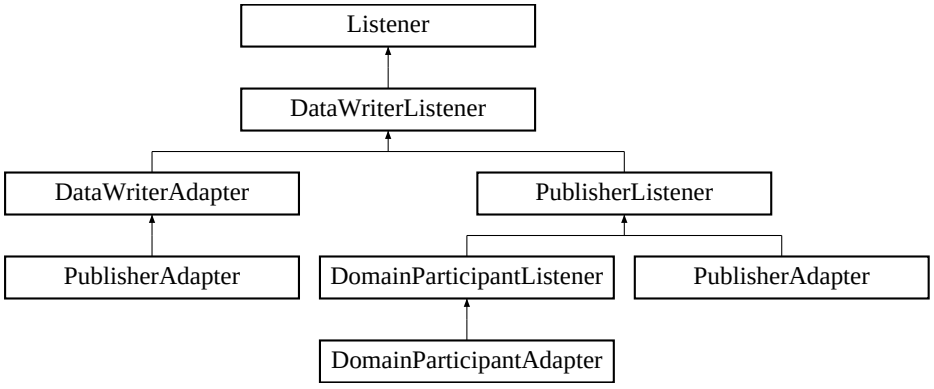
```
long unregistered_instance_count_peak
```

The highest value of **com.rti.dds.publication.DataWriterCacheStatus.unregistered_instance_count** (p. 589) over the lifetime of the **DataWriter** (p. 553).

8.62 DataWriterListener Interface Reference

*<<**interface**>> (p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for writer status.

Inheritance diagram for DataWriterListener:



Public Member Functions

- void **on_offered_deadline_missed** (**DataWriter** writer, **OfferedDeadlineMissedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.
- void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.
- void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.
- void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.
- void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  *<<**extension**>> (p. 155) A change has occurred in the writer's cache of unacknowledged samples.*
- void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  *<<**extension**>> (p. 155) A matched reliable reader has become active or become inactive.*
- void **on_sample_removed** (**DataWriter** writer, **Cookie_t** cookie)

  *<<**extension**>> (p. 155) Called when a sample is removed from the **DataWriter** (p. 553) queue.*
- void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Notifies when an instance is replaced in **DataWriter** (p. 553) queue.*
- void **on_application_acknowledgment** (**DataWriter** writer, **AcknowledgmentInfo** info)

  *<<**extension**>> (p. 155) Called when a sample is application-acknowledged*
- void **on_service_request_accepted** (**DataWriter** writer, **ServiceRequestAcceptedStatus** status)

  *<<**extension**>> (p. 155) Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.*

8.62.1 Detailed Description

*<<**interface**>> (p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for writer status.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.LivelinessLostStatus** (p. 1242);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1339);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1340);
com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS, **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1532);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS, **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1535);

See also

com.rti.dds.infrastructure.Listener (p. 1236)

Status Kinds (p. 262)

Operations Allowed in Listener Callbacks (p. 1238)

8.62.2 Member Function Documentation

8.62.2.1 on_offered_deadline_missed()

```
void on_offered_deadline_missed (
            DataWriter writer,
            OfferedDeadlineMissedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **DEADLINE** (p. 217) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **com.rti.dds.publication.DataWriter** (p. 553) failed to provide data for an instance.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 752), and **DataWriterAdapter** (p. 581).

8.62.2.2 on_offered_incompatible_qos()

```
void on_offered_incompatible_qos (
            DataWriter writer,
            OfferedIncompatibleQosStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 612) of the **com.rti.dds.publication.↩ DataWriter** (p. 553) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 450). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 553) has discovered a **com.rti.dds.↩ subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 752), and **DataWriterAdapter** (p. 582).

8.62.2.3 on_liveliness_lost()

```
void on_liveliness_lost (
              DataWriter writer,
              LivelinessLostStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **LIVELINESS** (p. 239) qos policy was not respected; this **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 553) simply remains not alive for another liveliness period.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 753), and **DataWriterAdapter** (p. 582).

8.62.2.4 on_publication_matched()

```
void on_publication_matched (
              DataWriter writer,
              PublicationMatchedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 553) has found a **com.rti.dds.subscription.↩ DataReader** (p. 450) that matches the **com.rti.dds.topic.Topic** (p. 1807), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 450) that was previously considered to be matched.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current publication match status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 753), and **DataWriterAdapter** (p. 583).

8.62.2.5 on_reliable_writer_cache_changed()

```
void on_reliable_writer_cache_changed (
            DataWriter writer,
            ReliableWriterCacheChangedStatus status )
```

<<***extension***>> *(p. 155)* A change has occurred in the writer's cache of unacknowledged samples.

This listener callback is triggered when:

- The cache is empty (contains no unacknowledged samples).
- The cache is full (the number of unacknowledged samples has reached the value specified in **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)).
- The number of unacknowledged samples has reached **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1607) or **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark** (p. 1607).

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 754), and **DataWriterAdapter** (p. 583).

8.62.2.6 on_reliable_reader_activity_changed()

```
void on_reliable_reader_activity_changed (
            DataWriter writer,
            ReliableReaderActivityChangedStatus status )
```

<<***extension***>> *(p. 155)* A matched reliable reader has become active or become inactive.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implemented in **DomainParticipantAdapter** (p. 754), and **DataWriterAdapter** (p. 584).

8.62.2.7 on_sample_removed()

```
void on_sample_removed (
              DataWriter writer,
              Cookie_t cookie )
```

<<***extension***>> *(*p. *155)* Called when a sample is removed from the **DataWriter** (p. 553) queue.

This callback is called only if the sample was written with a **com.rti.dds.infrastructure.Cookie_t** (p. 442) with **com.↩ rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), or if this writer uses **Zero Copy** (p. 56) transfer over shared memory" or **FlatData Topic-Types** (p. 55) "FlatData language binding".

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
cookie	<<***out***>> *(*p. *156)* • If this sample was written with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), this field contains a copy of the cookie set in **com.rti.dds.infrastructure.WriteParams_t** (p. 1994). • If this writer uses **Zero Copy transfer over shared memory** (p. 56) or **FlatData language binding** (p. 55), this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using com.rti.dds.infrastructure.Cookie_t.to_pointer

See also

com.rti.ndds.example.FooDataWriter.get_loan

Implemented in **DomainParticipantAdapter** (p. 755), and **DataWriterAdapter** (p. 584).

8.62.2.8 on_instance_replaced()

```
void on_instance_replaced (
              DataWriter writer,
              InstanceHandle_t handle )
```

<<***extension***>> *(*p. *155)* Notifies when an instance is replaced in **DataWriter** (p. 553) queue.

This callback is called when an instance is replaced by the **com.rti.dds.publication.DataWriter** (p. 553) due to instance resource limits being reached. This callback returns to the user the handle of the replaced instance, which can be used to get the key of the replaced instance using the **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114) API.

Because this callback can be called within the context of an in-progress write, dispose, or unregister call, most APIs on the **DataWriter** (p. 553) must not be used. The only **DataWriter** (p. 553) APIs that are safe to call within this callback are:

- **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114)
- com.rti.ndds.example.FooDataWriter.create_data
- com.rti.ndds.example.FooDataWriter.delete_data
- **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566)
- **com.rti.dds.publication.DataWriter.is_matched_subscription_active** (p. 567)
- **com.rti.dds.publication.DataWriter.get_matched_subscription_participant_data** (p. 569)
- **com.rti.dds.publication.DataWriter.get_topic** (p. 569)
- **com.rti.dds.publication.DataWriter.get_publisher** (p. 570)
- **com.rti.dds.publication.DataWriter.is_sample_app_acknowledged** (p. 571)

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
handle	<<***out***>> *(*p. *156)* Handle of the replaced instance

Implemented in **DomainParticipantAdapter** (p. 755), and **DataWriterAdapter** (p. 585).

8.62.2.9 on_application_acknowledgment()

```
void on_application_acknowledgment (
             DataWriter writer,
             AcknowledgmentInfo info )
```

<<***extension***>> *(*p. *155)* Called when a sample is application-acknowledged

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **com.rti.dds.subscription.DataReader** (p. 450). Also provides user-specified response data sent from the **com.rti.dds.subscription.DataReader** (p. 450) by the acknowledgment message.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
info	<<***out***>> *(*p. *156)* **com.rti.dds.publication.AcknowledgmentInfo** (p. 330) of the acknowledged sample

Implemented in **DomainParticipantAdapter** (p. 756), and **DataWriterAdapter** (p. 586).

8.62.2.10 on_service_request_accepted()

```
void on_service_request_accepted (
             DataWriter writer,
             ServiceRequestAcceptedStatus status )
```

<<***extension***>> *(*p. *155)* Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.↩ subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current service request accepted status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

See also

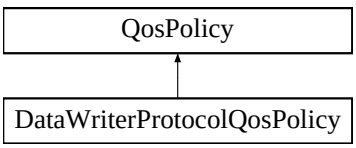
Topic Queries (p. 84)

Implemented in **DomainParticipantAdapter** (p. 757), and **DataWriterAdapter** (p. 586).

8.63 DataWriterProtocolQosPolicy Class Reference

Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 553) instances.

Inheritance diagram for DataWriterProtocolQosPolicy:



Public Attributes

- **GUID_t virtual_guid**

  The virtual GUID (Global Unique Identifier).
- int **rtps_object_id**

  The RTPS Object ID.
- boolean **push_on_write**

  Whether to push sample out when write is called.
- boolean **disable_positive_acks**

  Controls whether or not the writer expects positive acknowledgements from matching readers.
- boolean **disable_inline_keyhash**

  Controls whether or not a keyhash is propagated on the wire with each sample.
- boolean **serialize_key_with_dispose**

  Controls whether or not the serialized key is propagated on the wire with dispose samples.
- boolean **propagate_app_ack_with_no_response**

  *Controls whether or not a **com.rti.dds.publication.DataWriter** (p. 553) receives **com.rti.dds.publication.DataWriter↩ Listener.on_application_acknowledgment** (p. 595) notifications with an empty or invalid response.*
- final **RtpsReliableWriterProtocol_t rtps_reliable_writer**

  *RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **com.rti.dds.publication.↩ DataWriter** (p. 553). This parameter only has effect if both the writer and the matching reader are configured with **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.↩ infrastructure.ReliabilityQosPolicyKind** (p. 1531).*

8.63.1 Detailed Description

Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 553) instances.

DDS has a standard protocol for packet (user and meta data) exchange between applications using DDS for communications. This QoS policy and **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596) give you control over configurable portions of the protocol, including the configuration of the reliable data delivery mechanism of the protocol on a per DataWriter or DataReader basis.

These configuration parameters control timing, timeouts, and give you the ability to tradeoff between speed of data loss detection and repair versus network and CPU bandwidth used to maintain reliability.

It is important to tune the reliability protocol (on a per **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.↩ subscription.DataReader** (p. 450) basis) to meet the requirements of the end-user application so that data can be sent between DataWriters and DataReaders in an efficient and optimal manner in the presence of data loss.

You can also use this QoS policy to control how RTI Connext responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) for more information on the per-DataReader/DataWriter reliability configuration. **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.63.2 Member Data Documentation

8.63.2.1 virtual_guid

```
GUID_t virtual_guid
```

The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **com.rti.dds.publication.DataWriter** (p. 553).

RTI Connext uses the virtual GUID to associate a persisted writer history to a specific **com.rti.dds.publication.Data↩ Writer** (p. 553).

The RTI Connext Persistence Service uses the virtual GUID to send samples on behalf of the original **com.rti.dds.↩ publication.DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.GUID_t.GUID_AUTO (p. 1134)

8.63.2.2 rtps_object_id

```
int rtps_object_id
```

The RTPS Object ID.

This value is used to determine the RTPS object ID of a data writer according to the DDS-RTPS Interoperability Wire Protocol.

Only the last 3 bytes are used; the most significant byte is ignored.

If the default value is specified, RTI Connext will automatically assign the object ID based on a counter value (per participant) starting at 0x00800000. That value is incremented for each new data writer.

A rtps_object_id value in the interval [0x00800000,0x00ffffff] may collide with the automatic values assigned by RTI Connext. In those cases, the recomendation is not to use automatic object ID assigment.

[default] com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID (p. 1990)

[range] [0,0x00ffffff]

8.63.2.3 push_on_write

```
boolean push_on_write
```

Whether to push sample out when write is called.

If set to com.rti.dds.infrastructure.true (the default), the writer will send a sample every time write is called. Otherwise, the sample is put into the queue waiting for a NACK from remote reader(s) to be sent out.

[default] com.rti.dds.infrastructure.true

8.63.2.4 disable_positive_acks

```
boolean disable_positive_acks
```

Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to com.rti.dds.infrastructure.true, the writer does not expect readers to send send positive acknowledgments to the writer. Consequently, instead of keeping a sample queued until all readers have positively acknowledged it, the writer will keep a sample for at least **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_↩ min_sample_keep_duration** (p. 1614), after which the sample is logically considered as positively acknowledged.

If set to com.rti.dds.infrastructure.false (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 503) = com.rti.dds.infrastructure.false) and it applies the keep-duration to its non-acknowledging readers (**com.rti.dds.↩ infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 503) = com.rti.dds.infrastructure.true).

A writer with both acknowledging and non-acknowledging readers keeps a sample queued until acknowledgements have been received from all acknowledging readers and the keep-duration has elapsed for non-acknowledging readers.

[default] com.rti.dds.infrastructure.false

8.63.2.5 disable_inline_keyhash

```
boolean disable_inline_keyhash
```

Controls whether or not a keyhash is propagated on the wire with each sample.

This field only applies to keyed writers.

With each key, RTI Connext associates an internal 16-byte representation, called a keyhash.

When this field is com.rti.dds.infrastructure.false, the keyhash is sent on the wire with every data instance.

When this field is com.rti.dds.infrastructure.true, the keyhash is not sent on the wire and the readers must compute the value using the received data.

If the *reader* is CPU bound, sending the keyhash on the wire may increase performance, because the reader does not have to get the keyhash from the data.

If the *writer* is CPU bound, sending the keyhash on the wire may decrease performance, because it requires more bandwidth (16 more bytes per sample).

Note: Setting disable_inline_keyhash to com.rti.dds.infrastructure.true is not compatible with using RTI Real-Time Connect or RTI Recorder.

[default] com.rti.dds.infrastructure.false

8.63.2.6 serialize_key_with_dispose

```
boolean serialize_key_with_dispose
```

Controls whether or not the serialized key is propagated on the wire with dispose samples.

This field only applies to keyed writers.

We recommend setting this field to com.rti.dds.infrastructure.true if there are DataReaders where **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.propagate_dispose_of_unregistered_instances** (p. 504) is also com.rti.dds.infrastructure.true.

When setting serialize_key_with_dispose to FALSE, only a key hash is included in the dispose meta-sample sent by a DataWriter for a dispose action. If a dispose meta-sample only includes the key hash, then DataReaders must have previously received an actual data sample for the instance being disposed, in order for a DataReader to map a key hash/instance handle to actual key values.

If an actual data sample was never received for an instance and serialize_key_with_dispose is set to FALSE, then the DataReader application will not be able to determine the value of the key that was disposed, since **com.rti.ndds.example.FooDataReader.get_key_value** (p. 1095) will not be able to map an instance handle to actual key values.

By setting serialize_key_with_dispose to TRUE, the values of the key members of a data type will be sent in the dispose meta-sample for a dispose action by the DataWriter. This allows the DataReader to map an instance handle to the values of the key members even when receiving a dispose meta-sample without previously having received a data sample for the instance.

Important: When this field is com.rti.dds.infrastructure.true, batching will not be compatible with RTI Connext 4.3e, 4.4b, or 4.4c. The **com.rti.dds.subscription.DataReader** (p. 450) entities will receive incorrect data and/or encounter deserialization errors.

[default] com.rti.dds.infrastructure.false

8.63.2.7 propagate_app_ack_with_no_response

```
boolean propagate_app_ack_with_no_response
```

Controls whether or not a **com.rti.dds.publication.DataWriter** (p. 553) receives **com.rti.dds.publication.DataWriterListener.on_application_acknowledgment** (p. 595) notifications with an empty or invalid response.

When this field is set to com.rti.dds.infrastructure.false, the callback **com.rti.dds.publication.DataWriterListener.on_application_acknowledgment** (p. 595) will not be invoked if the sample being acknowledged has an empty or invalid response.

[default] com.rti.dds.infrastructure.true

8.63.2.8 rtps_reliable_writer

```
final RtpsReliableWriterProtocol_t rtps_reliable_writer
```

RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **com.rti.dds.publication.DataWriter** (p. 553). This parameter only has effect if both the writer and the matching reader are configured with **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.infrastructure.ReliabilityQosPolicyKind** (p. 1531).

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default] [default] See **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

8.64 DataWriterProtocolStatus Class Reference

<<***extension***>> *(*p. *155)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

Inherits Status.

Public Attributes

- long **pushed_sample_count**

  The number of user samples pushed on write from a local ***DataWriter*** *(*p. *553) to a matching remote DataReader.*
- long **pushed_sample_count_change**

  The change in ***com.rti.dds.publication.DataWriterProtocolStatus.pushed_sample_count*** *(*p. *603) since the last time the status was read.*
- long **pushed_sample_bytes**

  The number of bytes of user samples pushed on write from a local ***DataWriter*** *(*p. *553) to a matching remote DataReader.*
- long **pushed_sample_bytes_change**

  The change in ***com.rti.dds.publication.DataWriterProtocolStatus.pushed_sample_bytes*** *(*p. *604) since the last time the status was read.*
- long **filtered_sample_count**

  [Not supported.] *The number of user samples preemptively filtered by a local* ***DataWriter*** *(*p. *553) due to Content-Filtered Topics.*
- long **filtered_sample_count_change**

  [Not supported.] *The incremental change in the number of user samples preemptively filtered by a local* ***DataWriter*** *(*p. *553) due to Content-Filtered Topics since the last time the status was read.*
- long **filtered_sample_bytes**

  [Not supported.] *The number of user samples preemptively filtered by a local* ***DataWriter*** *(*p. *553) due to Content-Filtered Topics.*
- long **filtered_sample_bytes_change**

  [Not supported.] *The incremental change in the number of user samples preemptively filtered by a local* ***DataWriter*** *(*p. *553) due to Content-Filtered Topics since the last time the status was read.*
- long **sent_heartbeat_count**

  The number of Heartbeats sent between a local ***DataWriter*** *(*p. *553) and matching remote DataReader.*
- long **sent_heartbeat_count_change**

  The change in ***com.rti.dds.publication.DataWriterProtocolStatus.sent_heartbeat_count*** *(*p. *605) since the last time the status was read.*
- long **sent_heartbeat_bytes**

  The number of bytes of Heartbeats sent between a local ***DataWriter*** *(*p. *553) and matching remote DataReader.*
- long **sent_heartbeat_bytes_change**

  The change in ***com.rti.dds.publication.DataWriterProtocolStatus.sent_heartbeat_bytes*** *(*p. *605) since the last time the status was read.*
- long **pulled_sample_count**

  The number of user samples pulled from local ***DataWriter*** *(*p. *553) by matching DataReaders.*
- long **pulled_sample_count_change**

  The change in ***com.rti.dds.publication.DataWriterProtocolStatus.pulled_sample_count*** *(*p. *606) since the last time the status was read.*
- long **pulled_sample_bytes**

  The number of bytes of user samples pulled from local ***DataWriter*** *(*p. *553) by matching DataReaders.*

- long **pulled_sample_bytes_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.pulled_sample_bytes** (p. 606) since the last time the status was read.*
- long **received_ack_count**

  *The number of ACKs from a remote DataReader received by a local **DataWriter** (p. 553).*
- long **received_ack_count_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_ack_count** (p. 607) since the last time the status was read.*
- long **received_ack_bytes**

  *The number of bytes of ACKs from a remote DataReader received by a local **DataWriter** (p. 553).*
- long **received_ack_bytes_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_ack_bytes** (p. 607) since the last time the status was read.*
- long **received_nack_count**

  *The number of NACKs from a remote DataReader received by a local **DataWriter** (p. 553).*
- long **received_nack_count_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_nack_count** (p. 607) since the last time the status was read.*
- long **received_nack_bytes**

  *The number of bytes of NACKs from a remote DataReader received by a local **DataWriter** (p. 553).*
- long **received_nack_bytes_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_nack_bytes** (p. 608) since the last time the status was read.*
- long **sent_gap_count**

  *The number of GAPs sent from local **DataWriter** (p. 553) to matching remote DataReaders.*
- long **sent_gap_count_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_gap_count** (p. 608) since the last time the status was read.*
- long **sent_gap_bytes**

  *The number of bytes of GAPs sent from local **DataWriter** (p. 553) to matching remote DataReaders.*
- long **sent_gap_bytes_change**

  *The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_gap_bytes** (p. 609) since the last time the status was read.*
- long **rejected_sample_count**

  [Not supported.]
- long **rejected_sample_count_change**

  [Not supported.]
- int **send_window_size**

  *Current maximum number of outstanding samples allowed in the **DataWriter** (p. 553)'s queue.*
- **SequenceNumber_t first_available_sample_sequence_number**

  *The sequence number of the first available sample currently queued in the local **DataWriter** (p. 553).*
- **SequenceNumber_t last_available_sample_sequence_number**

  *The sequence number of the last available sample currently queued in the local **DataWriter** (p. 553).*
- **SequenceNumber_t first_unacknowledged_sample_sequence_number**

  *The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 553).*
- **SequenceNumber_t first_available_sample_virtual_sequence_number**

  *The virtual sequence number of the first available sample currently queued in the local **DataWriter** (p. 553).*
- **SequenceNumber_t last_available_sample_virtual_sequence_number**

*The virtual sequence number of the last available sample currently queued in the local **DataWriter** (p. 553).*

- **SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number**

  *The virtual sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 553).*
- **InstanceHandle_t first_unacknowledged_sample_subscription_handle**

  *The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local **DataWriter** (p. 553).*
- **SequenceNumber_t first_unelapsed_keep_duration_sample_sequence_number**

  The sequence number of the first sample whose keep duration has not yet elapsed.
- long **pushed_fragment_count**

  *The number of DATA_FRAG messages that have been pushed by this **DataWriter** (p. 553).*
- long **pushed_fragment_bytes**

  *The number of bytes of DATA_FRAG messages that have been pushed by this **DataWriter** (p. 553).*
- long **pulled_fragment_count**

  *The number of DATA_FRAG messages that have been pulled from this **DataWriter** (p. 553).*
- long **pulled_fragment_bytes**

  *The number of bytes of DATA_FRAG messages that have been pulled from this **DataWriter** (p. 553).*
- long **received_nack_fragment_count**

  *The number of NACK_FRAG messages that have been received by this **DataWriter** (p. 553).*
- long **received_nack_fragment_bytes**

  *The number of bytes of NACK_FRAG messages that have been received by this **DataWriter** (p. 553).*

8.64.1 Detailed Description

<<***extension***>> *(p. 155)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

8.64.2 Member Data Documentation

8.64.2.1 pushed_sample_count

```
long pushed_sample_count
```

The number of user samples pushed on write from a local **DataWriter** (p. 553) to a matching remote DataReader.

Counts protocol (RTPS) messages pushed by a **DataWriter** (p. 553) when writing, unregistering, and disposing. The count is the number of sends done internally, and it may be greater than the number of user writes.

For large data, counts whole samples, not fragments. The fragment count is tracked in the **com.rti.dds.publication.↩ DataWriterProtocolStatus.pushed_fragment_count** (p. 611) statistic.

8.64.2.2 pushed_sample_count_change

```
long pushed_sample_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.pushed_sample_count** (p. 603) since the last time the status was read.

Counts protocol (RTPS) messages pushed by a **DataWriter** (p. 553) when writing, unregistering, and disposing.

For large data, counts whole samples, not fragments.

8.64.2.3 pushed_sample_bytes

```
long pushed_sample_bytes
```

The number of bytes of user samples pushed on write from a local **DataWriter** (p. 553) to a matching remote Data↩ Reader.

Counts bytes of protocol (RTPS) messages pushed by a **DataWriter** (p. 553) when writing, unregistering, and disposing. The count of bytes corresponds to the number of sends done internally, and it may be greater than the number of user writes.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.64.2.4 pushed_sample_bytes_change

```
long pushed_sample_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.pushed_sample_bytes** (p. 604) since the last time the status was read.

Counts bytes of protocol (RTPS) messages pushed by a **DataWriter** (p. 553) when writing, unregistering, and disposing.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.64.2.5 filtered_sample_count

```
long filtered_sample_count
```

[Not supported.] The number of user samples preemptively filtered by a local **DataWriter** (p. 553) due to Content-↩ Filtered Topics.

8.64.2.6 filtered_sample_count_change

```
long filtered_sample_count_change
```

[Not supported.] The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 553) due to Content-Filtered Topics since the last time the status was read.

8.64.2.7 filtered_sample_bytes

```
long filtered_sample_bytes
```

[Not supported.] The number of user samples preemptively filtered by a local **DataWriter** (p. 553) due to Content-↩ Filtered Topics.

8.64.2.8 filtered_sample_bytes_change

```
long filtered_sample_bytes_change
```

[Not supported.] The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 553) due to Content-Filtered Topics since the last time the status was read.

8.64.2.9 sent_heartbeat_count

```
long sent_heartbeat_count
```

The number of Heartbeats sent between a local **DataWriter** (p. 553) and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

8.64.2.10 sent_heartbeat_count_change

```
long sent_heartbeat_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_heartbeat_count** (p. 605) since the last time the status was read.

8.64.2.11 sent_heartbeat_bytes

```
long sent_heartbeat_bytes
```

The number of bytes of Heartbeats sent between a local **DataWriter** (p. 553) and matching remote DataReader.

Because periodic and piggyback heartbeats are sent to remote readers and their locators differently in different situations, when a reader has more than one locator, this count may be larger than expected, to reflect the sending of Heartbeats to the multiple locators.

8.64.2.12 sent_heartbeat_bytes_change

```
long sent_heartbeat_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_heartbeat_bytes** (p. 605) since the last time the status was read.

8.64.2.13 pulled_sample_count

```
long pulled_sample_count
```

The number of user samples pulled from local **DataWriter** (p. 553) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 553) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 598) is com.rti.dds.infrastructure.false.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.64.2.14 pulled_sample_count_change

```
long pulled_sample_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.pulled_sample_count** (p. 606) since the last time the status was read.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 553) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 598) is com.rti.dds.infrastructure.false.

For large data, counts whole samples, not fragments.

8.64.2.15 pulled_sample_bytes

```
long pulled_sample_bytes
```

The number of bytes of user samples pulled from local **DataWriter** (p. 553) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 553) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 598) is com.rti.dds.infrastructure.false.

When data fragmentation is used, this statistic is incremented as fragments are written.

8.64.2.16 pulled_sample_bytes_change

```
long pulled_sample_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.pulled_sample_bytes** (p. 606) since the last time the status was read.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 553) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 598) is com.rti.dds.infrastructure.false.

For large data, counts bytes of whole samples, not fragments.

8.64.2.17 received_ack_count

```
long received_ack_count
```

The number of ACKs from a remote DataReader received by a local **DataWriter** (p. 553).

8.64.2.18 received_ack_count_change

```
long received_ack_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_ack_count** (p. 607) since the last time the status was read.

8.64.2.19 received_ack_bytes

```
long received_ack_bytes
```

The number of bytes of ACKs from a remote DataReader received by a local **DataWriter** (p. 553).

8.64.2.20 received_ack_bytes_change

```
long received_ack_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_ack_bytes** (p. 607) since the last time the status was read.

8.64.2.21 received_nack_count

```
long received_nack_count
```

The number of NACKs from a remote DataReader received by a local **DataWriter** (p. 553).

8.64.2.22 received_nack_count_change

```
long received_nack_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_nack_count** (p. 607) since the last time the status was read.

8.64.2.23 received_nack_bytes

```
long received_nack_bytes
```

The number of bytes of NACKs from a remote DataReader received by a local **DataWriter** (p. 553).

8.64.2.24 received_nack_bytes_change

```
long received_nack_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.received_nack_bytes** (p. 608) since the last time the status was read.

8.64.2.25 sent_gap_count

```
long sent_gap_count
```

The number of GAPs sent from local **DataWriter** (p. 553) to matching remote DataReaders.

8.64.2.26 sent_gap_count_change

```
long sent_gap_count_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_gap_count** (p. 608) since the last time the status was read.

8.64.2.27 sent_gap_bytes

```
long sent_gap_bytes
```

The number of bytes of GAPs sent from local **DataWriter** (p. 553) to matching remote DataReaders.

8.64.2.28 sent_gap_bytes_change

```
long sent_gap_bytes_change
```

The change in **com.rti.dds.publication.DataWriterProtocolStatus.sent_gap_bytes** (p. 609) since the last time the status was read.

8.64.2.29 rejected_sample_count

```
long rejected_sample_count
```

[Not supported.]

8.64.2.30 rejected_sample_count_change

```
long rejected_sample_count_change
```

[Not supported.]

8.64.2.31 send_window_size

```
int send_window_size
```

Current maximum number of outstanding samples allowed in the **DataWriter** (p. 553)'s queue.

Spans the range from **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1616) to **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617).

8.64.2.32 first_available_sample_sequence_number

```
SequenceNumber_t first_available_sample_sequence_number
```

The sequence number of the first available sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.33 last_available_sample_sequence_number

```
SequenceNumber_t last_available_sample_sequence_number
```

The sequence number of the last available sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.34 first_unacknowledged_sample_sequence_number

```
SequenceNumber_t first_unacknowledged_sample_sequence_number
```

The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.35 first_available_sample_virtual_sequence_number

```
SequenceNumber_t first_available_sample_virtual_sequence_number
```

The virtual sequence number of the first available sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.36 last_available_sample_virtual_sequence_number

```
SequenceNumber_t last_available_sample_virtual_sequence_number
```

The virtual sequence number of the last available sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.37 first_unacknowledged_sample_virtual_sequence_number

```
SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number
```

The virtual sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.38 first_unacknowledged_sample_subscription_handle

```
InstanceHandle_t first_unacknowledged_sample_subscription_handle
```

The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local **DataWriter** (p. 553).

Applies only for local **DataWriter** (p. 553) status.

8.64.2.39 first_unelapsed_keep_duration_sample_sequence_number

```
SequenceNumber_t first_unelapsed_keep_duration_sample_sequence_number
```

The sequence number of the first sample whose keep duration has not yet elapsed.

Applicable only when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_positive_acks** (p. 598) is set.

Sequence number of the first sample kept in the **DataWriter** (p. 553)'s queue whose keep_duration (applied when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_positive_acks** (p. 598) is set) has not yet elapsed.

Applies only for local **DataWriter** (p. 553) status.

8.64.2.40 pushed_fragment_count

```
long pushed_fragment_count
```

The number of DATA_FRAG messages that have been pushed by this **DataWriter** (p. 553).

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.64.2.41 pushed_fragment_bytes

```
long pushed_fragment_bytes
```

The number of bytes of DATA_FRAG messages that have been pushed by this **DataWriter** (p. 553).

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.64.2.42 pulled_fragment_count

```
long pulled_fragment_count
```

The number of DATA_FRAG messages that have been pulled from this **DataWriter** (p. 553).

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.64.2.43 pulled_fragment_bytes

```
long pulled_fragment_bytes
```

The number of bytes of DATA_FRAG messages that have been pulled from this **DataWriter** (p. 553).

This statistic is incremented as each DATA_FRAG message is sent, not when the entire sample has been sent. Applicable only when data is fragmented.

8.64.2.44 received_nack_fragment_count

```
long received_nack_fragment_count
```

The number of NACK_FRAG messages that have been received by this **DataWriter** (p. 553).

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.64.2.45 received_nack_fragment_bytes

```
long received_nack_fragment_bytes
```

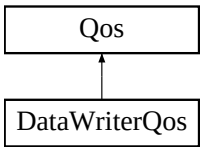
The number of bytes of NACK_FRAG messages that have been received by this **DataWriter** (p. 553).

NACK FRAG RTPS messages are sent when large data is used in conjunction with reliable communication. They have the same properties as NACK messages, but instead of applying to samples, they apply to fragments. Applicable only when data is fragmented.

8.65 DataWriterQos Class Reference

QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 553) entity.

Inheritance diagram for DataWriterQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**DataWriterQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **DataWriterQos*** (p. *612*) *object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **DataWriterQos*** (p. *612*) *object.*
- String **toString** (**DataWriterQos** baseQos)

  *Obtains a string representation of a **DataWriterQos*** (p. *612*) *object.*

Public Attributes

- final **DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY*** (p. *230*)*.*
- final **DurabilityServiceQosPolicy durability_service**

  *DurabilityService policy, **DURABILITY_SERVICE*** (p. *232*)*.*
- final **DeadlineQosPolicy deadline**

  *Deadline policy, **DEADLINE*** (p. *217*)*.*
- final **LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET*** (p. *238*)*.*
- final **LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS*** (p. *239*)*.*
- final **ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY*** (p. *258*)*.*
- final **DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER*** (p. *218*)*.*
- final **HistoryQosPolicy history**

  *History policy, **HISTORY*** (p. *237*)*.*
- final **ResourceLimitsQosPolicy resource_limits**

  *Resource limits policy, **RESOURCE_LIMITS*** (p. *259*)*.*
- final **TransportPriorityQosPolicy transport_priority**

  *Transport priority policy, **TRANSPORT_PRIORITY*** (p. *274*)*.*
- final **LifespanQosPolicy lifespan**

  *Lifespan policy, **LIFESPAN*** (p. *238*)*.*
- final **UserDataQosPolicy user_data**

  *User data policy, **USER_DATA*** (p. *278*)*.*
- final **OwnershipQosPolicy ownership**

  *Ownership policy, **OWNERSHIP*** (p. *244*)*.*
- final **OwnershipStrengthQosPolicy ownership_strength**

  *Ownership strength policy, **OWNERSHIP_STRENGTH*** (p. *245*)*.*
- final **WriterDataLifecycleQosPolicy writer_data_lifecycle**

  *Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE*** (p. *285*)*.*
- final **DataWriterResourceLimitsQosPolicy writer_resource_limits**

  *<<**extension**>>* (p. *155*) *Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS*** (p. *216*)*.*
- final **DataWriterProtocolQosPolicy protocol**

<<***extension***>> (p. *155*) ***com.rti.dds.publication.DataWriter*** (p. *553*) *protocol policy,* ***DATA_WRITER_PROTOCOL*** (p. *215*)

- final **TransportSelectionQosPolicy transport_selection**

  <<***extension***>> (p. *155*) *Transport plugin selection policy,* ***TRANSPORT_SELECTION*** (p. *275*).
- final **TransportUnicastQosPolicy unicast**

  <<***extension***>> (p. *155*) *Unicast transport policy,* ***TRANSPORT_UNICAST*** (p. *276*).
- final **PublishModeQosPolicy publish_mode**

  <<***extension***>> (p. *155*) *Publish mode policy,* ***PUBLISH_MODE*** (p. *249*).
- final **PropertyQosPolicy property**

  <<***extension***>> (p. *155*) *Property policy,* ***PROPERTY*** (p. *248*). *See also* `Property Reference Guide`.
- final **DataTagQosPolicy data_tags**

  DataTag policy, ***DATA_TAG*** (p. *214*).
- final **ServiceQosPolicy service**

  <<***extension***>> (p. *155*) *Service policy,* ***SERVICE*** (p. *261*).
- final **BatchQosPolicy batch**

  <<***extension***>> (p. *155*) *Batch policy,* ***BATCH*** (p. *170*).
- final **MultiChannelQosPolicy multi_channel**

  <<***extension***>> (p. *155*) *Multi channel policy,* ***MULTICHANNEL*** (p. *243*).
- final **AvailabilityQosPolicy availability**

  <<***extension***>> (p. *155*) *Availability policy,* ***AVAILABILITY*** (p. *169*).
- final **EntityNameQosPolicy publication_name**

  <<***extension***>> (p. *155*) *EntityName policy,* ***ENTITY_NAME*** (p. *234*).
- final **TopicQueryDispatchQosPolicy topic_query_dispatch**

  <<***extension***>> (p. *155*) *Topic Query dispatch policy,* ***TOPIC_QUERY_DISPATCH*** (p. *269*).
- final **DataRepresentationQosPolicy representation**

  Data representation policy, ***DATA_REPRESENTATION*** (p. *212*).
- final **TypeSupportQosPolicy type_support**

  <<***extension***>> (p. *155*) *Type support data,* ***TYPESUPPORT*** (p. *277*).

8.65.1 Detailed Description

QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 553) entity.

You must set certain members in a consistent manner:

- **com.rti.dds.publication.DataWriterQos.history** (p. 618) .depth <= **com.rti.dds.publication.DataWriter↩ Qos.resource_limits** (p. 618) .max_samples_per_instance
- **com.rti.dds.publication.DataWriterQos.resource_limits** (p. 618) .max_samples_per_instance <= **com.rti.↩ dds.publication.DataWriterQos.resource_limits** (p. 618) .max_samples
- **com.rti.dds.publication.DataWriterQos.resource_limits** (p. 618) .initial_samples <= **com.rti.dds.↩ publication.DataWriterQos.resource_limits** (p. 618) .max_samples
- **com.rti.dds.publication.DataWriterQos.resource_limits** (p. 618) .initial_instances <= **com.rti.dds.↩ publication.DataWriterQos.resource_limits** (p. 618) .max_instances

- length of **com.rti.dds.publication.DataWriterQos.user_data** (p. 619) .value <= **com.rti.dds.domain.↩ DomainParticipantQos.resource_limits** (p. 800) .writer_user_data_max_length

If any of the above are not true, **com.rti.dds.publication.DataWriter.set_qos** (p. 556) and **com.rti.dds.publication.↩ DataWriter.set_qos_with_profile** (p. 557) and **com.rti.dds.publication.Publisher.set_default_datawriter_↩ qos** (p. 1469) and **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1470) will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) and **com.rti.dds.publication.↩ Publisher.create_datawriter** (p. 1471) and **com.rti.dds.publication.Publisher.create_datawriter_with_profile** (p. 1473) and will return NULL.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

See also

QoS Policies (p. 250) allowed ranges within each Qos.

8.65.2 Member Function Documentation

8.65.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 615) overloads allow formatting the output and printing only the differences with respect to another **DataWriterQos** (p. 612) object.

```
DataWriterQos qos = new DataWriterQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
DataWriterQos baseQos = new DataWriterQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value DATAWRITER_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(DATAWRITER_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **DataWriterQos** (p. 612) and the documented default.

Returns

The string representation of the Qos.

References **DataWriterQos.toString()**.

Referenced by **DataWriterQos.toString()**.

8.65.2.2 toString() [2/4]

```
String toString (
              DataWriterQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **DataWriterQos** (p. 612) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.publication.Publisher.DATAWRITER_QOS_PRINT_ALL** (p. 72) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **Publisher.DATAWRITER_QOS_PRINT_ALL**.

8.65.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **DataWriterQos** (p. 612) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DataWriterQos.toString()**.

8.65.2.4 toString() [4/4]

```
String toString (
              DataWriterQos baseQos )
```

Obtains a string representation of a **DataWriterQos** (p. 612) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.publication.Publisher.DATAWRITER_QOS_PRINT_ALL** (p. 72) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DataWriterQos.toString()**.

8.65.3 Member Data Documentation

8.65.3.1 durability

```
final DurabilityQosPolicy durability
```

Durability policy, **DURABILITY** (p. 230).

8.65.3.2 durability_service

```
final DurabilityServiceQosPolicy durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 232).

8.65.3.3 deadline

```
final DeadlineQosPolicy deadline
```

Deadline policy, **DEADLINE** (p. 217).

8.65.3.4 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 238).

8.65.3.5 liveliness

```
final LivelinessQosPolicy liveliness
```

Liveliness policy, **LIVELINESS** (p. 239).

8.65.3.6 reliability

```
final ReliabilityQosPolicy reliability
```

Reliability policy, **RELIABILITY** (p. 258).

8.65.3.7 destination_order

```
final DestinationOrderQosPolicy destination_order
```

Destination order policy, **DESTINATION_ORDER** (p. 218).

8.65.3.8 history

```
final HistoryQosPolicy history
```

History policy, **HISTORY** (p. 237).

8.65.3.9 resource_limits

```
final ResourceLimitsQosPolicy resource_limits
```

Resource limits policy, **RESOURCE_LIMITS** (p. 259).

8.65.3.10 transport_priority

```
final TransportPriorityQosPolicy transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 274).

8.65.3.11 lifespan

```
final LifespanQosPolicy lifespan
```

Lifespan policy, **LIFESPAN** (p. 238).

8.65.3.12 user_data

```
final UserDataQosPolicy user_data
```

User data policy, **USER_DATA** (p. 278).

8.65.3.13 ownership

```
final OwnershipQosPolicy ownership
```

Ownership policy, **OWNERSHIP** (p. 244).

8.65.3.14 ownership_strength

```
final OwnershipStrengthQosPolicy ownership_strength
```

Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 245).

8.65.3.15 writer_data_lifecycle

```
final WriterDataLifecycleQosPolicy writer_data_lifecycle
```

Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 285).

8.65.3.16 writer_resource_limits

```
final DataWriterResourceLimitsQosPolicy writer_resource_limits
```

<<***extension***>> *(*p. *155)* Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 216).

8.65.3.17 protocol

```
final DataWriterProtocolQosPolicy protocol
```

<<***extension***>> *(*p. *155)* **com.rti.dds.publication.DataWriter** (p. 553) protocol policy, **DATA_WRITER_PROTOCOL** (p. 215)

8.65.3.18 transport_selection

```
final TransportSelectionQosPolicy transport_selection
```

<<***extension***>> *(*p. *155)* Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 275).

Specifies the transports available for use by the **com.rti.dds.publication.DataWriter** (p. 553).

8.65.3.19 unicast

```
final TransportUnicastQosPolicy unicast
```

<<***extension***>> *(*p. *155)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 276).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **com.rti.dds.subscription.DataReader** (p. 450) entities in the domain.

8.65.3.20 publish_mode

```
final PublishModeQosPolicy publish_mode
```

<<***extension***>> *(*p. *155)* Publish mode policy, **PUBLISH_MODE** (p. 249).

Determines whether the **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.65.3.21 property

```
final PropertyQosPolicy property
```

<<***extension***>> *(*p. *155)* Property policy, **PROPERTY** (p. 248). See also `Property Reference Guide`.

8.65.3.22 data_tags

```
final DataTagQosPolicy data_tags
```

DataTag policy, **DATA_TAG** (p. 214).

8.65.3.23 service

```
final ServiceQosPolicy service
```

<<***extension***>> *(*p. *155)* Service policy, **SERVICE** (p. 261).

8.65.3.24 batch

```
final BatchQosPolicy batch
```

<<***extension***>> *(*p. *155)* Batch policy, **BATCH** (p. 170).

8.65.3.25 multi_channel

```
final MultiChannelQosPolicy multi_channel
```

<<***extension***>> *(*p. *155)* Multi channel policy, **MULTICHANNEL** (p. 243).

8.65.3.26 availability

```
final  AvailabilityQosPolicy availability
```

<<***extension***>> *(*p. *155)* Availability policy, **AVAILABILITY** (p. 169).

8.65.3.27 publication_name

```
final  EntityNameQosPolicy publication_name
```

<<***extension***>> *(*p. *155)* EntityName policy, **ENTITY_NAME** (p. 234).

8.65.3.28 topic_query_dispatch

```
final  TopicQueryDispatchQosPolicy topic_query_dispatch
```

<<***extension***>> *(*p. *155)* Topic Query dispatch policy, **TOPIC_QUERY_DISPATCH** (p. 269).

8.65.3.29 representation

```
final  DataRepresentationQosPolicy representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 212).

8.65.3.30 type_support

```
final  TypeSupportQosPolicy type_support
```

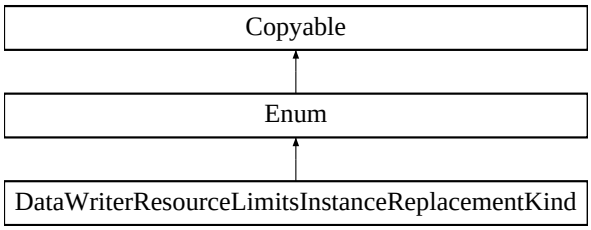
<<***extension***>> *(*p. *155)* Type support data, **TYPESUPPORT** (p. 277).

Optional value that is passed to a type plugin's on_endpoint_attached and serialization functions.

8.66 DataWriterResourceLimitsInstanceReplacementKind Class Reference

Sets the kinds of instances that can be replaced when instance resource limits are reached.

Inheritance diagram for DataWriterResourceLimitsInstanceReplacementKind:



Static Public Attributes

- static final **DataWriterResourceLimitsInstanceReplacementKind UNREGISTERED_INSTANCE_↩ REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) to reclaim unregistered acknowledged instances.*
- static final **DataWriterResourceLimitsInstanceReplacementKind ALIVE_INSTANCE_REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) to reclaim alive, acknowledged instances.*
- static final **DataWriterResourceLimitsInstanceReplacementKind DISPOSED_INSTANCE_REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) to reclaim disposed acknowledged instances.*
- static final **DataWriterResourceLimitsInstanceReplacementKind ALIVE_THEN_DISPOSED_INSTANCE_↩ REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) first to reclaim an alive, acknowledged instance, and then, if necessary, a disposed, acknowledged instance.*
- static final **DataWriterResourceLimitsInstanceReplacementKind DISPOSED_THEN_ALIVE_INSTANCE_↩ REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) first to reclaim a disposed, acknowledged instance, and then, if necessary, an alive, acknowledged instance.*
- static final **DataWriterResourceLimitsInstanceReplacementKind ALIVE_OR_DISPOSED_INSTANCE_↩ REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. *553) to reclaim a either an alive acknowledged instance or a disposed acknowledged instance.*

Additional Inherited Members

8.66.1 Detailed Description

Sets the kinds of instances that can be replaced when instance resource limits are reached.

When **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is reached, a **com.rti.dds.↩ publication.DataWriter** (p. 553) will try to make room for a new instance by attempting to reclaim an existing instance based on the instance replacement kind specified by **com.rti.dds.infrastructure.DataWriterResourceLimitsQos↩ Policy.instance_replacement** (p. 629).

Only instances whose states match the specified kinds are eligible to be replaced. In addition, an instance must have had all of its samples fully acknowledged for it to be considered replaceable.

For all kinds, a **com.rti.dds.publication.DataWriter** (p. 553) will replace the oldest instance satisfying that kind. For example, when the kind is com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind.Data↩ WriterResourceLimitsInstanceReplacementKind.UNREGISTERED_INSTANCE_REPLACEMENT, a **com.rti.dds.↩ publication.DataWriter** (p. 553) will remove the oldest, fully acknowledged, unregistered instance, if such an instance exists.

If no replaceable instance exists, the invoked function will either return with an appropriate out-of-resources return code, or in the case of a write, it may first block to wait for an instance to be acknowledged. Otherwise, the **com.↩ rti.dds.publication.DataWriter** (p. 553) will replace the old instance with the new instance, and invoke, if available, the **com.rti.dds.publication.DataWriterListener.on_instance_replaced** (p. 594) to notify the user about an instance being replaced.

A **com.rti.dds.publication.DataWriter** (p. 553) checks for replaceable instances in the following order, stopping once a replaceable instance is found:

- If **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.replace_empty_instances** (p. 629) is com.rti.dds.infrastructure.true, a **com.rti.dds.publication.DataWriter** (p. 553) first tries replacing instances that have no samples. These empty instances can be unregistered, disposed, or alive.
- Next, a **com.rti.dds.publication.DataWriter** (p. 553) tries replacing unregistered instances. Since an unregistered instance indicates that the **com.rti.dds.publication.DataWriter** (p. 553) is done modifying it, unregistered instances are replaced before instances of any other state (alive, disposed). This is the same as the com.↩ rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstance↩ ReplacementKind.UNREGISTERED_INSTANCE_REPLACEMENT kind.
- Then, a **com.rti.dds.publication.DataWriter** (p. 553) tries replacing what is specified by **com.rti.dds.↩ infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629). With unregistered instances already checked, this leaves alive and disposed instances. When both alive and disposed instances may be replaced, the kind specifies whether the particular order matters (e.g., DISPOSED_THEN_ALIVE, ALIVE_THEN_DISPOSED) or not (ALIVE_OR_DISPOSED).

QoS:

com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy (p. 626)

8.66.2 Member Data Documentation

8.66.2.1 UNREGISTERED_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind UNREGISTERED_INSTANCE_REPLACEMENT [static]
```

Allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim unregistered acknowledged instances.

By default, all instance replacement kinds first attempt to reclaim an unregistered, acknowledged instance. Used in **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629) **[default]**

8.66.2.2 ALIVE_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind ALIVE_INSTANCE_REPLACEMENT [static]
```

Allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim alive, acknowledged instances.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim an alive, acknowledged instance, where an alive instance is a registered, non-disposed instance. The least recently registered or written alive instance will be reclaimed.

8.66.2.3 DISPOSED_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind DISPOSED_INSTANCE_REPLACEMENT [static]
```

Allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim disposed acknowledged instances.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim a disposed, acknowledged instance. The least recently disposed instance will be reclaimed.

8.66.2.4 ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT
[static]
```

Allows a **com.rti.dds.publication.DataWriter** (p. 553) first to reclaim an alive, acknowledged instance, and then, if necessary, a disposed, acknowledged instance.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **com.rti.dds.publication.DataWriter** (p. 553) to first try reclaiming an alive, acknowledged instance. If no instance is reclaimable, then it tries reclaiming a disposed, acknowledged instance. The least recently used (i.e., registered, written, or disposed) instance will be reclaimed.

8.66.2.5 DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT
[static]
```

Allows a **com.rti.dds.publication.DataWriter** (p. 553) first to reclaim a disposed, acknowledged instance, and then, if necessary, an alive, acknowledged instance.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a **com.rti.dds.publication.DataWriter** (p. 553) to first try reclaiming a disposed, acknowledged instance. If no instance is reclaimable, then it tries reclaiming an alive, acknowledged instance. The least recently used (i.e., disposed, registered, or written) instance will be reclaimed.

8.66.2.6 ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT

```
final DataWriterResourceLimitsInstanceReplacementKind ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT
[static]
```

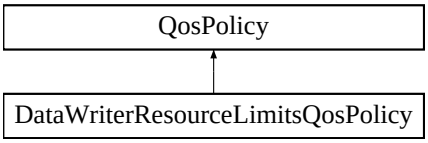
Allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim a either an alive acknowledged instance or a disposed acknowledged instance.

When an unregistered acknowledged instance is not available to reclaim, this kind allows a **com.rti.dds.publication.DataWriter** (p. 553) to reclaim either an alive, acknowledged instance or a disposed, acknowledged instance. If both instance kinds are available to reclaim, the **com.rti.dds.publication.DataWriter** (p. 553) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

8.67 DataWriterResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.

Inheritance diagram for DataWriterResourceLimitsQosPolicy:



Public Attributes

- int **initial_concurrent_blocking_threads**

  *The initial number of threads that are allowed to concurrently block on write call on the same **com.rti.dds.publication.DataWriter** (p. 553).*
- int **max_concurrent_blocking_threads**

  *The maximum number of threads that are allowed to concurrently block on write call on the same **com.rti.dds.publication.DataWriter** (p. 553).*
- int **max_remote_reader_filters**

  *The maximum number of remote DataReaders for which the **com.rti.dds.publication.DataWriter** (p. 553) will perform content-based filtering.*
- int **max_batches**

  *Represents the maximum number of batches a **com.rti.dds.publication.DataWriter** (p. 553) will manage.*
- int **initial_batches**

  *Represents the initial number of batches a **com.rti.dds.publication.DataWriter** (p. 553) will manage.*
- **DataWriterResourceLimitsInstanceReplacementKind instance_replacement**

  Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
- boolean **replace_empty_instances**

  Whether or not to replace empty instances during instance replacement.
- boolean **autoregister_instances**

  Whether or not to automatically register new instances.

- int **initial_virtual_writers**

  *The initial number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 553).*
- int **max_virtual_writers**

  *The maximum number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 553).*
- int **max_remote_readers**

  *The maximum number of remote readers supported by a **com.rti.dds.publication.DataWriter** (p. 553).*
- int **max_app_ack_remote_readers**

  *The maximum number of application-level acknowledging remote readers supported by a **com.rti.dds.publication.↩ DataWriter** (p. 553).*
- int **initial_active_topic_queries**

  *Represents the initial number of active topic queries a **com.rti.dds.publication.DataWriter** (p. 553) will manage.*
- int **max_active_topic_queries**

  *Represents the maximum number of active topic queries a **com.rti.dds.publication.DataWriter** (p. 553) will manage.*

8.67.1 Detailed Description

Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 553) allocates and uses physical memory for internal resources.

DataWriters must allocate internal structures to handle the simultaneously blocking of threads trying to call **com.↩ rti.ndds.example.FooDataWriter.write** (p. 1105) on the same **com.rti.dds.publication.DataWriter** (p. 553), for the storage used to batch small samples, and for content-based filters specified by DataReaders.

Most of these internal structures start at an initial size and, by default, will be grown as needed by dynamically allocating additional memory. You may set fixed, maximum sizes for these internal structures if you want to bound the amount of memory that can be used by a **com.rti.dds.publication.DataWriter** (p. 553). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **com.rti.dds.↩ publication.DataWriter** (p. 553).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.67.2 Member Data Documentation

8.67.2.1 initial_concurrent_blocking_threads

```
int initial_concurrent_blocking_threads
```

The initial number of threads that are allowed to concurrently block on write call on the same **com.rti.dds.publication.DataWriter** (p. 553).

This value only applies if **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) has its kind set to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) is $>$ 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

8.67.2.2 max_concurrent_blocking_threads

```
int max_concurrent_blocking_threads
```

The maximum number of threads that are allowed to concurrently block on write call on the same **com.rti.dds.publication.DataWriter** (p. 553).

This value only applies if **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) has its kind set to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) is $>$ 0.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 10000] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_concurrent_blocking_threads

8.67.2.3 max_remote_reader_filters

```
int max_remote_reader_filters
```

The maximum number of remote DataReaders for which the **com.rti.dds.publication.DataWriter** (p. 553) will perform content-based filtering.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [0, (2^31)-2] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

0: The **com.rti.dds.publication.DataWriter** (p. 553) will not perform filtering for any **com.rti.dds.subscription.DataReader** (p. 450).

1 to (2^31)-2: The DataWriter will filter for up to the specified number of DataReaders. In addition, the Datawriter will store the result of the filtering per sample per DataReader.

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259): The DataWriter will filter for up to (2^31)-2 DataReaders. However, in this case, the DataWriter will not store the filtering result per sample per DataReader. Thus, if a sample is resent (such as due to a loss of reliable communication), the sample will be filtered again.

8.67.2.4 max_batches

```
int max_batches
```

Represents the maximum number of batches a **com.rti.dds.publication.DataWriter** (p. 553) will manage.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

When batching is enabled, the maximum number of samples that a **com.rti.dds.publication.DataWriter** (p. 553) can store is limited by this value and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592).

[range] [1,100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) >= DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples if batching is enabled

See also

com.rti.dds.infrastructure.BatchQosPolicy (p. 355)

8.67.2.5 initial_batches

```
int initial_batches
```

Represents the initial number of batches a **com.rti.dds.publication.DataWriter** (p. 553) will manage.

[default] 8

[range] [1,100 million]

See also

com.rti.dds.infrastructure.BatchQosPolicy (p. 355)

8.67.2.6 instance_replacement

```
DataWriterResourceLimitsInstanceReplacementKind instance_replacement
```

Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

When a **com.rti.dds.publication.DataWriter** (p. 553)'s number of active instances is greater than **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592), it will try to make room by replacing an existing instance. This field specifies the kinds of instances allowed to be replaced.

If a replaceable instance is not available, either an out-of-resources exception will be returned, or the writer may block if the instance reclamation was done when writing.

[default] com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED_INSTANCE_REPLACEMENT

See also

com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind (p. 623)

8.67.2.7 replace_empty_instances

```
boolean replace_empty_instances
```

Whether or not to replace empty instances during instance replacement.

When a **com.rti.dds.publication.DataWriter** (p. 553) has more active instances than allowed by **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592), it tries to make room by replacing an existing instance. This field configures whether empty instances (i.e. instances with no samples) may be replaced. If set com.rti.dds.infrastructure.true, then a **com.rti.dds.publication.DataWriter** (p. 553) will first try reclaiming empty instances, before trying to replace whatever is specified by **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629).

[default] com.rti.dds.infrastructure.false

See also

com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind (p. 623)

8.67.2.8 autoregister_instances

```
boolean autoregister_instances
```

Whether or not to automatically register new instances.

[default] com.rti.dds.infrastructure.false

When set to true, it is possible to write with a non-NIL handle of an instance that is not registered: the write operation will succeed and the instance will be registered. Otherwise, that write operation would fail.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

8.67.2.9 initial_virtual_writers

```
int initial_virtual_writers
```

The initial number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 553).

[default] 1

[range] [1, 1000000], or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.67.2.10 max_virtual_writers

```
int max_virtual_writers
```

The maximum number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 553).

Sets the maximum number of unique virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 553), where virtual writers are added when samples are written with the virtual writer GUID.

This field is specially relevant in the configuration of Persistence Service DataWriters since these DataWriters will publish samples on behalf of multiple virtual writers.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1000000], or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.67.2.11 max_remote_readers

```
int max_remote_readers
```

The maximum number of remote readers supported by a **com.rti.dds.publication.DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1000000], or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.67.2.12 max_app_ack_remote_readers

```
int max_app_ack_remote_readers
```

The maximum number of application-level acknowledging remote readers supported by a **com.rti.dds.publication.↩ DataWriter** (p. 553).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1000000], or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.67.2.13 initial_active_topic_queries

```
int initial_active_topic_queries
```

Represents the initial number of active topic queries a **com.rti.dds.publication.DataWriter** (p. 553) will manage.

[default] 1

[range] [1, 1000000]

See also

com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy (p. 1833)

8.67.2.14 max_active_topic_queries

```
int max_active_topic_queries
```

Represents the maximum number of active topic queries a **com.rti.dds.publication.DataWriter** (p. 553) will manage.

When topic queries are enabled, the maximum number of topic queries that a **com.rti.dds.publication.DataWriter** (p. 553) can publish data samples for at the same time is limited by this value.

When the DataWriter receives one topic query above this limit, it will wait to process it until it finishes publishing all the samples for at least one of the current topic queries.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1000000] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

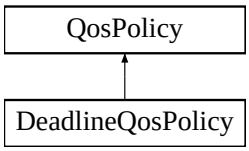
See also

com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy (p. 1833)

8.68 DeadlineQosPolicy Class Reference

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Inheritance diagram for DeadlineQosPolicy:



Public Attributes

- final **Duration_t period**

  Duration of the deadline period.

8.68.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **com.rti.dds.subscription.DataReader** (p. 450) expects a new sample updating the value of each instance at least once every `period`. That is, `period` specifies the maximum expected elapsed time between arriving data samples.

A **com.rti.dds.publication.DataWriter** (p. 553) indicates that the application commits to write a new value (using the **com.rti.dds.publication.DataWriter** (p. 553)) for each instance managed by the **com.rti.dds.publication.DataWriter** (p. 553) at least once every `period`.

This QoS can be used during system integration to ensure that applications have been coded to meet design specifications.

It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions to prevent total system failure when data is not received in time. For topics on which data is not expected to be periodic, `period` should be set to an infinite value.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS, com.rti.dds.infrastructure.↩ StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.infrastructure.StatusKind.↩ StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **YES** (p. 256)

8.68.2 Usage

This policy is useful for cases where a **com.rti.dds.topic.Topic** (p. 1807) is expected to have each instance updated periodically. On the publishing side this setting establishes a contract that the application must meet. On the subscribing side the setting establishes a minimum requirement for the remote publishers that are expected to supply the data values.

When RTI Connext 'matches' a **com.rti.dds.publication.DataWriter** (p. 553) and a **com.rti.dds.subscription.Data↩ Reader** (p. 450) it checks whether the settings are compatible (i.e., *offered deadline <= requested deadline*); if they are not, the two entities are informed (via the **com.rti.dds.infrastructure.Listener** (p. 1236) or **com.rti.dds.↩ infrastructure.Condition** (p. 429) mechanism) of the incompatibility of the QoS settings and communication will not occur.

Assuming that the reader and writer ends have compatible settings, the fulfilment of this contract is monitored by RTI Connext and the application is informed of any violations by means of the proper **com.rti.dds.infrastructure.Listener** (p. 1236) or **com.rti.dds.infrastructure.Condition** (p. 429).

8.68.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period* <= *requested period* holds.

8.68.4 Consistency

The setting of the **DEADLINE** (p. 217) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 267).

For these two policies to be consistent the settings must be such that *deadline period* >= *minimum_separation*.

An attempt to set these policies in an inconsistent manner will result in **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596) in **set_qos (abstract)** (p. 1030), or the **com.rti.dds.infrastructure.Entity** (p. 1029) will not be created.

For a **com.rti.dds.subscription.DataReader** (p. 450), the **DEADLINE** (p. 217) policy and **com.rti.dds.↩ infrastructure.TimeBasedFilterQosPolicy** (p. 1804) may interact such that even though the **com.rti.dds.↩ publication.DataWriter** (p. 553) is writing samples fast enough to fulfill its commitment to its own deadline, the **com.rti.dds.subscription.DataReader** (p. 450) may see violations of its deadline. This happens because RTI Connext will drop any samples received within the **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy.minimum_↩ separation** (p. 1806). To avoid triggering the **com.rti.dds.subscription.DataReader** (p. 450)'s deadline, even though the matched **com.rti.dds.publication.DataWriter** (p. 553) is meeting its own deadline, set the two QoS parameters so that:

reader deadline >= *reader minimum_separation + writer deadline*

See **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy** (p. 1804) for more information about the interactions between deadlines and time-based filters.

See also

com.rti.dds.infrastructure.TimeBasedFilterQosPolicy (p. 1804)

8.68.5 Member Data Documentation

8.68.5.1 period

```
final Duration_t period
```

Duration of the deadline period.

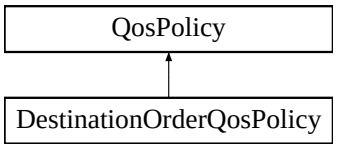
[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 nanosec, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), >= **com.rti.↩ dds.infrastructure.TimeBasedFilterQosPolicy.minimum_separation** (p. 1806)

8.69 DestinationOrderQosPolicy Class Reference

Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1807) and key).

Inheritance diagram for DestinationOrderQosPolicy:



Public Attributes

- **DestinationOrderQosPolicyKind kind**

  Specifies the desired kind of destination order.
- **DestinationOrderQosPolicyScopeKind scope = DestinationOrderQosPolicyScopeKind.INSTANCE_↩ SCOPE_DESTINATIONORDER_QOS**

  Specifies the desired scope of the source destination order.
- final **Duration_t source_timestamp_tolerance**

  <<***extension***>> *(p. 155) Allowed tolerance between source timestamps of consecutive samples.*

8.69.1 Detailed Description

Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1807) and key).

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.69.2 Usage

When multiple DataWriters send data for the same topic, the order in which data from different DataWriters are received by the applications of different DataReaders may be different. So different DataReaders may not receive the same "last" value when DataWriters stop sending data.

This QoS policy controls how each subscriber resolves the final value of a data instance that is written by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities (which may be associated with different **com.rti.dds.publication.Publisher** (p. 1466) entities) running on different nodes.

This QoS can be used to create systems that have the property of "eventual consistency." Thus intermediate states across multiple applications may be inconsistent, but when DataWriters stop sending changes to the same topic, all applications will end up having the same state.

This QoS policy can be set for both DataWriters and DataReaders.

For the DataReader:

The default setting, com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 245) policy allows it) the latest received value for the instance should be the one whose value is kept. That is, data will be delivered by a **com.rti.dds.subscription.DataReader** (p. 450) in the order in which it was *received* (which may lead to inconsistent final values).

For com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, if the scope is set to com.rti.dds.infrastructure.DestinationOrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.INSTANCE_SCOPE_DESTINATIONORDER_QOS (default), within each instance, the sample's source timestamp shall be used to determine the most recent information. This is the only setting that, in the case of concurrent same-strength DataWriters updating the same instance, ensures that all DataReaders end up with the same final value for the instance. If a DataReader receives a sample for an instance with a source timestamp that is older than the last source timestamp received for the instance, the sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

If scope is set to com.rti.dds.infrastructure.DestinationOrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.TOPIC_SCOPE_DESTINATIONORDER_QOS, the ordering is enforced per topic across all instances.

In addition, a DataReader will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than source_timestamp_tolerance. Otherwise, the DDS sample is dropped. The SAMPLE_REJECTED status or the SAMPLE_LOST status will not be updated.

For the DataWriter:

For the default setting, com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, the DataWriter will not enforce source timestamp ordering when writing samples using the **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110) or **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) API. The source timestamp of a new sample can be older than the source timestamp of the previous samples.

When using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, If scope is set to com.rti.dds.infrastructure.DestinationOrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.INSTANCE_SCOPE_DESTINATIONORDER_QOS (default), when writing a sample, the sample's timestamp must not be older than the timestamp of the previously written DDS sample for the same instance. If, however, the timestamp is older than the timestamp of the previously written DDS sample—but the difference is less than the source_timestamp_tolerance—the DDS sample will use the previously written DDS sample's timestamp as its timestamp. Otherwise, if the difference is greater than the tolerance, the write will fail with retcode **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

If scope is set to com.rti.dds.infrastructure.DestinationOrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.TOPIC_SCOPE_DESTINATIONORDER_QOS, a new sample timestamp must not be older than the timestamp of the previously written DDS sample, across all instances. (The ordering is enforced across all instances.)

8.69.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **com.rti.dds.infrastructure.↩ DestinationOrderQosPolicy.kind** (p. 637) are considered ordered such that com.rti.dds.infrastructure.Destination↩ OrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_↩ QOS < com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_↩ TIMESTAMP_DESTINATIONORDER_QOS

8.69.4 Member Data Documentation

8.69.4.1 kind

DestinationOrderQosPolicyKind kind

Specifies the desired kind of destination order.

[default] com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION↩ _TIMESTAMP_DESTINATIONORDER_QOS,

8.69.4.2 scope

DestinationOrderQosPolicyScopeKind scope = **DestinationOrderQosPolicyScopeKind.INSTANCE_SCOPE_↩ DESTINATIONORDER_QOS**

Specifies the desired scope of the source destination order.

Indicates if tolerance check and the current sample's timestamp is computed based on instance or topic basis.
[default] com.rti.dds.infrastructure.DestinationOrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.↩ INSTANCE_SCOPE_DESTINATIONORDER_QOS

8.69.4.3 source_timestamp_tolerance

final **Duration_t** source_timestamp_tolerance

<<***extension***>> *(p. 155)* Allowed tolerance between source timestamps of consecutive samples.

When a **com.rti.dds.publication.DataWriter** (p. 553) sets **com.rti.dds.infrastructure.DestinationOrderQos↩ PolicyKind** (p. 638) to com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.↩ BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, when writing a sample, its timestamp must not be less than the timestamp of the previously written sample. However, if it is less than the timestamp of the previously written sample but the difference is less than this tolerance, the sample will use the previously written sample's timestamp as its timestamp. Otherwise, if the difference is greater than this tolerance, the write will fail.

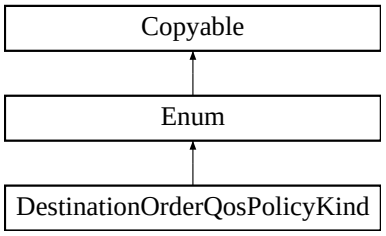
When a **com.rti.dds.subscription.DataReader** (p. 450) sets **com.rti.dds.infrastructure.DestinationOrderQos↩ PolicyKind** (p. 638) to com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.↩ BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, the **com.rti.dds.subscription.DataReader** (p. 450) will accept a sample only if the source timestamp is no farther in the future from the reception timestamp than this tolerance. Otherwise, the sample is dropped.

[default] 100 milliseconds for **com.rti.dds.publication.DataWriter** (p. 553), 30 seconds for **com.rti.dds.↩ subscription.DataReader** (p. 450)

8.70 DestinationOrderQosPolicyKind Class Reference

Kinds of destination order.

Inheritance diagram for DestinationOrderQosPolicyKind:



Static Public Attributes

- static final **DestinationOrderQosPolicyKind BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS**

  [default] *Indicates that data is ordered based on the reception time at each* ***com.rti.dds.subscription.Subscriber*** *(p. 1730).*
- static final **DestinationOrderQosPolicyKind BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS**

  Indicates that data is ordered based on a time-stamp placed at the source (by RTI Connext or by the application).

Additional Inherited Members

8.70.1 Detailed Description

Kinds of destination order.

QoS:

com.rti.dds.infrastructure.DestinationOrderQosPolicy (p. 635)

8.70.2 Member Data Documentation

8.70.2.1 BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS

```
final DestinationOrderQosPolicyKind BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS [static]
```

[default] Indicates that data is ordered based on the reception time at each **com.rti.dds.subscription.Subscriber** (p. 1730).

Since each subscriber may receive the data at different times there is no guarantee that the changes will be seen in the same order. Consequently, it is possible for each subscriber to end up with a different final value for the data.

8.70.2.2 BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS

```
final DestinationOrderQosPolicyKind BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS [static]
```

Indicates that data is ordered based on a time-stamp placed at the source (by RTI Connext or by the application).

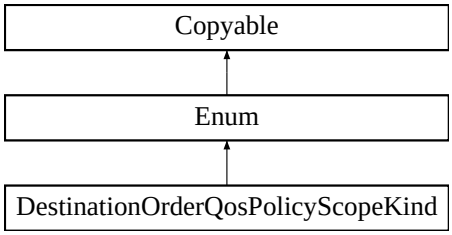
In any case this guarantees a consistent final value for the data in all subscribers.

Note: If Batching is needed along with com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrder↩ QosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS and com.rti.dds.infrastructure.Destination↩ OrderQosPolicyScopeKind.DestinationOrderQosPolicyScopeKind.INSTANCE_SCOPE_DESTINATIONORDER_QOS, then the **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 358) and **com.rti.dds.↩ infrastructure.BatchQosPolicy.thread_safe_write** (p. 359) setting of **com.rti.dds.infrastructure.BatchQosPolicy** (p. 355) should be set to com.rti.dds.infrastructure.Duration_t.ZERO and com.rti.dds.infrastructure.true respectively.

8.71 DestinationOrderQosPolicyScopeKind Class Reference

Scope of source destination order.

Inheritance diagram for DestinationOrderQosPolicyScopeKind:



Static Public Attributes

- static final **DestinationOrderQosPolicyScopeKind INSTANCE_SCOPE_DESTINATIONORDER_QOS**

  [default] *Indicates that data is ordered on a per instance basis if used along with com.rti.dds.infrastructure.Destination↩ OrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.*
- static final **DestinationOrderQosPolicyScopeKind TOPIC_SCOPE_DESTINATIONORDER_QOS**

  Indicates that data is ordered on a per topic basis if used along with com.rti.dds.infrastructure.DestinationOrderQos↩ PolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.

Additional Inherited Members

8.71.1 Detailed Description

Scope of source destination order.

QoS:

com.rti.dds.infrastructure.DestinationOrderQosPolicy (p. 635)

8.71.2 Member Data Documentation

8.71.2.1 INSTANCE_SCOPE_DESTINATIONORDER_QOS

```
final DestinationOrderQosPolicyScopeKind INSTANCE_SCOPE_DESTINATIONORDER_QOS [static]
```

[default] Indicates that data is ordered on a per instance basis if used along with com.rti.dds.infrastructure.Destination↩ OrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.

The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same instance. The tolerance check is also applied per instance.

8.71.2.2 TOPIC_SCOPE_DESTINATIONORDER_QOS

```
final DestinationOrderQosPolicyScopeKind TOPIC_SCOPE_DESTINATIONORDER_QOS [static]
```

Indicates that data is ordered on a per topic basis if used along with com.rti.dds.infrastructure.DestinationOrderQos↩ PolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.

The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same topic. The tolerance check is also applied per topic.

8.72 DiscoveryBuiltinReaderFragmentationResourceLimits_t Class Reference

Inherits Struct.

Public Member Functions

- **DiscoveryBuiltinReaderFragmentationResourceLimits_t** ()
- **DiscoveryBuiltinReaderFragmentationResourceLimits_t** (boolean **disable_fragmentation_support**, int **max_fragmented_samples**, int **initial_fragmented_samples**, int **max_fragmented_samples_per_remote↩ _writer**, int **max_fragments_per_sample**, boolean **dynamically_allocate_fragmented_samples**)

Public Attributes

- boolean **disable_fragmentation_support** = false
- int **max_fragmented_samples** = 1024
- int **initial_fragmented_samples** = 4
- int **max_fragmented_samples_per_remote_writer** = 256
- int **max_fragments_per_sample** = 512
- boolean **dynamically_allocate_fragmented_samples** = false

8.72.1 Detailed Description

DiscoveryBuiltinReaderFragmentationResourceLimits_t (p. 640)

8.72.2 Constructor & Destructor Documentation

8.72.2.1 DiscoveryBuiltinReaderFragmentationResourceLimits_t() [1/2]

```
DiscoveryBuiltinReaderFragmentationResourceLimits_t ( )
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_new_with_no_parameter

8.72.2.2 DiscoveryBuiltinReaderFragmentationResourceLimits_t() [2/2]

```
DiscoveryBuiltinReaderFragmentationResourceLimits_t (
            boolean disable_fragmentation_support,
            int max_fragmented_samples,
            int initial_fragmented_samples,
            int max_fragmented_samples_per_remote_writer,
            int max_fragments_per_sample,
            boolean dynamically_allocate_fragmented_samples )
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_new_with_ints

References **DiscoveryBuiltinReaderFragmentationResourceLimits_t.disable_fragmentation_support**, **DiscoveryBuiltinReaderFragmentationResourceLimits_t.dynamically_allocate_fragmented_samples**, **DiscoveryBuiltinReaderFragmentationResourceLimits_t.initial_fragmented_samples**, **DiscoveryBuiltinReaderFragmentationResourceLimits_t.max_fragmented_samples**, **DiscoveryBuiltinReaderFragmentationResourceLimits_t.max_fragmented_samples_per_remote_writer**, and **DiscoveryBuiltinReaderFragmentationResourceLimits_t.max_fragments_per_sample**.

8.72.3 Member Data Documentation

8.72.3.1 disable_fragmentation_support

```
boolean disable_fragmentation_support = false
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_disable_fragmentation_support

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentationResourceLimits_t()**.

8.72.3.2 max_fragmented_samples

```
int max_fragmented_samples = 1024
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragmented_samples

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentation↩ ResourceLimits_t()**.

8.72.3.3 initial_fragmented_samples

```
int initial_fragmented_samples = 4
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_initial_fragmented_samples

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentation↩ ResourceLimits_t()**.

8.72.3.4 max_fragmented_samples_per_remote_writer

```
int max_fragmented_samples_per_remote_writer = 256
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragmented_samples_per_remote_writer

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentation↩ ResourceLimits_t()**.

8.72.3.5 max_fragments_per_sample

```
int max_fragments_per_sample = 512
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragments_per_sample

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentation↩ ResourceLimits_t()**.

8.72.3.6 dynamically_allocate_fragmented_samples

```
boolean dynamically_allocate_fragmented_samples = false
```

DiscoveryBuiltinReaderFragmentationResourceLimits_t_dynamically_allocate_fragmented_samples

Referenced by **DiscoveryBuiltinReaderFragmentationResourceLimits_t.DiscoveryBuiltinReaderFragmentation↩ ResourceLimits_t()**.

8.73 DiscoveryConfigBuiltinChannelKind Class Reference

Built-in channels that can be enabled.

Static Public Attributes

- static final int **SERVICE_REQUEST_CHANNEL**

  [default] *Built-in ServiceRequest channel.*
- static final int **MASK_NONE**

  No bits are set, indicating that all channels that can be disabled, are disabled. Not all builtin channels can be disabled by the user. Only builtin channels represented by ***com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKind*** *(*p. *643) are able to be enabled or disabled by the user.*
- static final int **MASK_ALL**

  All bits are set, indicating that all channels are enabled.
- static final int **MASK_DEFAULT** = **SERVICE_REQUEST_CHANNEL**

  The default value of ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.enabled_builtin_channels*** *(*p. *655).*

8.73.1 Detailed Description

Built-in channels that can be enabled.

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKindMask

8.73.2 Member Data Documentation

8.73.2.1 SERVICE_REQUEST_CHANNEL

```
final int SERVICE_REQUEST_CHANNEL  [static]
```

[default] Built-in ServiceRequest channel.

Enables the ServiceRequest channel, which is required by the TopicQuery and locator reachability features. Disabling the ServiceRequest channel reduces resource consumption including network bandwidth, CPU utilization, and memory.

8.74 DiscoveryConfigBuiltinPluginKind Class Reference

Built-in discovery plugins that can be used.

Static Public Attributes

- static final int **SPDP**

  Simple Participant Discovery Protocol.
- static final int **SEDP**

  Simple Endpoint Discovery Protocol.
- static final int **SDP**

  ***[default]** Simple discovery plugin.*
- static final int **DPSE**

  Dynamic Participant discovery, Static Endpoint discovery.
- static final int **SPDP2**

  Simple Participant Discovery Protocol 2.0 (SPDP2)
- static final int **SDP2**

  Simple discovery plugin 2.0.
- static final int **MASK_NONE**

  No bits are set.
- static final int **MASK_DEFAULT**

  *The default value of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.builtin_discovery_plugins** (p. 655).*

8.74.1 Detailed Description

Built-in discovery plugins that can be used.

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind (p. 644)

8.74.2 Member Data Documentation

8.74.2.1 SPDP

```
final int SPDP  [static]
```

Simple Participant Discovery Protocol.

Enables the first phase of the Simple Discovery Protocol (SDP), in which DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant declaration messages, also known as participant DATA submessages or participant announcements.

8.74.2.2 SEDP

```
final int SEDP  [static]
```

Simple Endpoint Discovery Protocol.

Enables the second phase of the Simple Discovery Protocol (SDP), in which the information (GUID, QoS, etc.) about your application's DataReaders and DataWriters is exchanged by sending publication/subscription declarations in DATA messages, also known as publication DATAs and subscription DATAs.

8.74.2.3 SDP

```
final int SDP  [static]
```

[default] Simple discovery plugin.

It is equivalent to SPDP + SEDP.

8.74.2.4 DPSE

```
final int DPSE  [static]
```

Dynamic Participant discovery, Static Endpoint discovery.

Enables static endpoint discovery for a DomainParticipant. In this type of discovery, information from remote endpoints is extracted from a local DDS-XML file instead of being received over the network, reducing the number of exchanged packets and consequently reducing bandwidth consumption used for discovery. Using this value in **com.↩ rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind** (p. 644) requires the 'librtilbedisc' library (included in the RTI Connext Professional bundles) to be reachable (PATH, LD_LIBRARY_PATH or DYLD_LIBRARY_PATH).

8.74.2.5 SPDP2

```
final int SPDP2  [static]
```

Simple Participant Discovery Protocol 2.0 (SPDP2)

Enables the Simple Participant Discovery Protocol 2.0, in which a DomainParticipant's details are communicated to all other DomainParticipants in the same DDS domain by sending participant bootstrap messages. These bootstrap messages contain only a subset of the information in the Simple Participant Discovery Protocol (SPDP) participant announcements that is required to match two participants and bootstrap the system. The DomainParticpant's full configuration is then sent reliably with participant configuration announcements. Two DomainParticipants that use SPDP2 will maintain liveliness using liveliness participant messages.

8.74.2.6 SDP2

```
final int SDP2  [static]
```

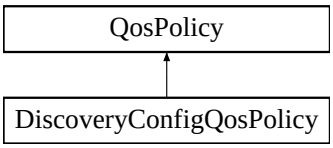
Simple discovery plugin 2.0.

It is equivalent to SPDP2 + SEDP.

8.75 DiscoveryConfigQosPolicy Class Reference

Settings for discovery configuration.

Inheritance diagram for DiscoveryConfigQosPolicy:



Public Attributes

- final **Duration_t participant_liveliness_lease_duration**

  The liveliness lease duration for the participant.
- final **Duration_t participant_liveliness_assert_period**

  The period to assert liveliness for the participant.
- final **Duration_t participant_announcement_period**

  The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
- **RemoteParticipantPurgeKind remote_participant_purge_kind**

  The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
- final **Duration_t max_liveliness_loss_detection_period**

  The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
- int **initial_participant_announcements**

  The number of initial announcements sent when a participant is first enabled.
- int **new_remote_participant_announcements**

  The number of participant announcements sent when a remote participant is newly discovered.
- final **Duration_t min_initial_participant_announcement_period**

  The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
- final **Duration_t max_initial_participant_announcement_period**

  The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
- final **BuiltinTopicReaderResourceLimits_t participant_reader_resource_limits**

  Resource limits.
- final **BuiltinTopicReaderResourceLimits_t publication_reader_resource_limits**

  Resource limits.
- final **BuiltinTopicReaderResourceLimits_t subscription_reader_resource_limits**

  Resource limits.
- final **RtpsReliableWriterProtocol_t publication_writer**

  RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
- final **WriterDataLifecycleQosPolicy publication_writer_data_lifecycle**

  Writer data lifecycle settings for a built-in publication writer.

- final **RtpsReliableWriterProtocol_t subscription_writer**

  RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
- final **WriterDataLifecycleQosPolicy subscription_writer_data_lifecycle**

  Writer data lifecycle settings for a built-in subscription writer.
- final **RtpsReliableReaderProtocol_t publication_reader**

  RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
- final **RtpsReliableReaderProtocol_t subscription_reader**

  RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
- int **builtin_discovery_plugins**

  Mask of built-in discovery plugin kinds.
- int **enabled_builtin_channels**

  The mask specifying which built-in channels should be enabled.
- **ReliabilityQosPolicyKind participant_message_reader_reliability_kind**

  Reliability policy for a built-in participant message reader.
- final **RtpsReliableReaderProtocol_t participant_message_reader**

  *RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_reader_reliability_kind** (p. 655) is set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).*
- final **RtpsReliableWriterProtocol_t participant_message_writer**

  *RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.infrastructure.ReliabilityQosPolicyKind** (p. 1531).*
- final **PublishModeQosPolicy publication_writer_publish_mode**

  Publish mode policy for the built-in publication writer.
- final **PublishModeQosPolicy subscription_writer_publish_mode**

  Publish mode policy for the built-in subscription writer.
- final **AsynchronousPublisherQosPolicy asynchronous_publisher**

  *Asynchronous publishing settings for the discovery **com.rti.dds.publication.Publisher** (p. 1466) and all entities that are created by it.*
- final **Duration_t default_domain_announcement_period**

  The period to announce a participant to the default domain 0.
- boolean **ignore_default_domain_announcements**

  Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.
- final **RtpsReliableWriterProtocol_t service_request_writer**

  *RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) writer.*
- final **WriterDataLifecycleQosPolicy service_request_writer_data_lifecycle**

  *Writer data lifecycle settings for a built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) writer.*
- final **PublishModeQosPolicy service_request_writer_publish_mode**

  Publish mode policy for the built-in service request writer.
- final **RtpsReliableReaderProtocol_t service_request_reader**

  *RTPS reliable reader protocol-related configuration settings for a built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) reader.*
- final **Duration_t locator_reachability_assert_period**

  Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
- final **Duration_t locator_reachability_lease_duration**

The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

- final **Duration_t locator_reachability_change_detection_period**

  Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
- final **RtpsReliableWriterProtocol_t secure_volatile_writer**

  RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
- final **PublishModeQosPolicy secure_volatile_writer_publish_mode**

  Publish mode policy for the built-in secure volatile writer.
- final **RtpsReliableReaderProtocol_t secure_volatile_reader**

  RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
- int **endpoint_type_object_lb_serialization_threshold** = 0

  Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery.
- final **Duration_t dns_tracker_polling_period**

  Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.
- final **BuiltinTopicReaderResourceLimits_t participant_configuration_reader_resource_limits**

  Resource limits for the built-in topic participant configuration reader.
- final **RtpsReliableWriterProtocol_t participant_configuration_writer**

  RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
- final **WriterDataLifecycleQosPolicy participant_configuration_writer_data_lifecycle**

  Writer data lifecycle settings for a built-in participant configuration writer.
- final **RtpsReliableReaderProtocol_t participant_configuration_reader**

  RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
- final **PublishModeQosPolicy participant_configuration_writer_publish_mode**

  Publish mode policy for the built-in participant configuration writer.

8.75.1 Detailed Description

Settings for discovery configuration.

<<***extension***>> *(*p. *155)* This QoS policy controls the amount of delay in discovering entities in the system and the amount of discovery traffic in the network.

The amount of network traffic required by the discovery process can vary widely, based on how your application has chosen to configure the middleware's network addressing (e.g., unicast vs. multicast, multicast TTL, etc.), the size of the system, whether all applications are started at the same time or whether start times are staggered, and other factors. Your application can use this policy to make tradeoffs between discovery completion time and network bandwidth utilization. In addition, you can introduce random back-off periods into the discovery process to decrease the probability of network contention when many applications start simultaneously.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.75.2 Member Data Documentation

8.75.2.1 participant_liveliness_lease_duration

```
final Duration_t participant_liveliness_lease_duration
```

The liveliness lease duration for the participant.

This is the same as the expiration time of the DomainParticipant as defined in the RTPS protocol.

If the participant has not refreshed its own liveliness to other participants at least once within this period, it may be considered as stale by other participants in the network.

Should be strictly greater than **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_assert_period** (p. 649).

[default] 100 seconds

[range] [1 nanosec,1 year], > participant_liveliness_assert_period

8.75.2.2 participant_liveliness_assert_period

```
final Duration_t participant_liveliness_assert_period
```

The period to assert liveliness for the participant.

The period at which the participant will refresh its liveliness to all the peers.

Should be strictly less than **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_lease_duration** (p. 649).

[default] 30 seconds

[range] [1 nanosec,1 year], < participant_liveliness_lease_duration

8.75.2.3 participant_announcement_period

```
final Duration_t participant_announcement_period
```

The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).

The **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) list **com.rti.dds.domain.DomainParticipant.add_peer** (p. 729) API are used to configure a set of potential peers that a DomainParticipant may discover. The **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_announcement_period** (p. 649) configures how frequently a DomainParticipant will announce itself to the subset of the configured potential peers that it has not matched with yet. Once a DomainParticipant matches with a DomainParticipant at one of configured potential peer locators, it will no longer announce itself to that locator at this period unless liveliness is lost.

This QoS policy is only supported when using the Simple Participant Discovery Protocol 2.0 (SPDP2). Setting this value when using the Simple Participant Discovery Protocol (SPDP) or other participant discovery protocols is not supported and will result in an error.

[default] com.rti.dds.infrastructure.Duration_t.AUTO (Takes the value of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_assert_period** (p. 649))

[range] [1 nanosec,1 year]

8.75.2.4 remote_participant_purge_kind

```
RemoteParticipantPurgeKind remote_participant_purge_kind
```

The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.

Most users will not need to change this value from its default, **com.rti.dds.infrastructure.RemoteParticipantPurge↩ Kind.LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE** (p. 1541). However, **com.rti.dds.infrastructure.↩ RemoteParticipantPurgeKind.NO_REMOTE_PARTICIPANT_PURGE** (p. 1541) may be a good choice if the following conditions apply:

1. Discovery communication with a remote participant may be lost while data communication remains intact. Such will not typically be the case if discovery takes place over the Simple Discovery Protocol, but may be the case if the RTI Enterprise Discovery Service is used.
2. Extensive and prolonged lack of discovery communication between participants is not expected to be common, either because participant loss itself is expected to be rare, or because participants may be lost sporadically but will typically return again.
3. Maintaining inter-participant liveliness is problematic, perhaps because a participant has no writers with the appropriate **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247).

[default] com.rti.dds.infrastructure.RemoteParticipantPurgeKind.LIVELINESS_BASED_REMOTE_PARTICIPANT↩ _PURGE (p. 1541)

8.75.2.5 max_liveliness_loss_detection_period

```
final Duration_t max_liveliness_loss_detection_period
```

The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.

Notification of the loss of liveliness of a remote entity may come more quickly than this duration, depending on the liveliness contract between the local and remote entities and the capabilities of the discovery mechanism in use. For example, a **com.rti.dds.subscription.DataReader** (p. 450) will learn of the loss of liveliness of a matched **com.rti.↩ dds.publication.DataWriter** (p. 553) within the reader's offered liveliness lease duration.

Shortening this duration will increase the responsiveness of entities to communication failures. However, it will also increase the CPU usage of the application, as the liveliness of remote entities will be examined more frequently.

[default] 60 seconds

[range] [1 nanosec, 1 year]

8.75.2.6 initial_participant_announcements

```
int initial_participant_announcements
```

The number of initial announcements sent when a participant is first enabled.

[default] 5

[range] [1,1 million]

8.75.2.7 new_remote_participant_announcements

```
int new_remote_participant_announcements
```

The number of participant announcements sent when a remote participant is newly discovered.

These announcements are only sent to the newly discovered remote participant, they are not also broadcast to the initial_peers list.

[default] 2

[range] [0,1 million]

8.75.2.8 min_initial_participant_announcement_period

```
final Duration_t min_initial_participant_announcement_period
```

The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between this and **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 651) is introduced in between initial announcements when a new remote participant is discovered.

The setting of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period** (p. 651) must be consistent with **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 651). For these two values to be consistent, they must verify that:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period (p. 651) <= **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 651).

[default] 10 milliseconds

[range] [1 nanosec,1 year]

8.75.2.9 max_initial_participant_announcement_period

```
final Duration_t max_initial_participant_announcement_period
```

The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

A random delay between **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period** (p. 651) and this is introduced in between initial announcements when a new remote participant is discovered.

The setting of **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 651) must be consistent with **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period** (p. 651). For these two values to be consistent, they must verify that:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period (p. 651) <= **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 651).

[default] 1 second

[range] [1 nanosec,1 year]

8.75.2.10 participant_reader_resource_limits

```
final BuiltinTopicReaderResourceLimits_t participant_reader_resource_limits
```

Resource limits.

Resource limit of the built-in topic participant reader. For details, see **com.rti.dds.infrastructure.BuiltinTopicReader↩ ResourceLimits_t** (p. 378).

8.75.2.11 publication_reader_resource_limits

```
final BuiltinTopicReaderResourceLimits_t publication_reader_resource_limits
```

Resource limits.

Resource limit of the built-in topic publication reader. For details, see **com.rti.dds.infrastructure.BuiltinTopicReader↩ ResourceLimits_t** (p. 378).

8.75.2.12 subscription_reader_resource_limits

```
final BuiltinTopicReaderResourceLimits_t subscription_reader_resource_limits
```

Resource limits.

Resource limit of the built-in topic subscription reader. For details, see **com.rti.dds.infrastructure.BuiltinTopic↩ ReaderResourceLimits_t** (p. 378).

8.75.2.13 publication_writer

```
final RtpsReliableWriterProtocol_t publication_writer
```

RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);
samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;

nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.14 publication_writer_data_lifecycle

`final` **`WriterDataLifecycleQosPolicy`** `publication_writer_data_lifecycle`

Writer data lifecycle settings for a built-in publication writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006). **com.rti.dds.↩ infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 2007) will always be forced to com.rti.dds.infrastructure.true.

8.75.2.15 subscription_writer

`final` **`RtpsReliableWriterProtocol_t`** `subscription_writer`

RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);
samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;

disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.16 subscription_writer_data_lifecycle

```
final WriterDataLifecycleQosPolicy subscription_writer_data_lifecycle
```

Writer data lifecycle settings for a built-in subscription writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006). **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 2007) will always be forced to com.rti.dds.infrastructure.true.

8.75.2.17 publication_reader

```
final RtpsReliableReaderProtocol_t publication_reader
```

RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.18 subscription_reader

```
final RtpsReliableReaderProtocol_t subscription_reader
```

RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.19 builtin_discovery_plugins

```
int builtin_discovery_plugins
```

Mask of built-in discovery plugin kinds.

There are several built-in discovery plugins. This mask enables the different plugins. Any plugin not enabled will not be created.

[default] com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind.SDP (p. 645)

See also

com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind (p. 644)

8.75.2.20 enabled_builtin_channels

```
int enabled_builtin_channels
```

Initial value:

```
=
            DiscoveryConfigBuiltinChannelKind.MASK_DEFAULT
```

The mask specifying which built-in channels should be enabled.

While there are a number of built-in channels that are used by Connext DDS, the only built-in channel which can currently be enabled or disabled is the Service Request Channel. This channel is used by the Locator Reachability and Topic Query features. If you are not using these features and wish to reduce network traffic and endpoint resource usage, you may disable the service request channel with this QoS.

[default] com.rti.dds.infrastructure.DiscoveryConfigBuiltinChannelKind.DiscoveryConfigBuiltinChannelKind.SERVICE↩
_REQUEST_CHANNEL

8.75.2.21 participant_message_reader_reliability_kind

ReliabilityQosPolicyKind participant_message_reader_reliability_kind

Reliability policy for a built-in participant message reader.

For details, refer to the **com.rti.dds.infrastructure.ReliabilityQosPolicyKind** (p. 1531).

[default] com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS (p. 1532)

8.75.2.22 participant_message_reader

final **RtpsReliableReaderProtocol_t** participant_message_reader

RTPS reliable reader protocol-related configuration settings for a built-in participant message reader. This parameter only has effect if **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_reader_reliability_kind** (p. 655) is set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.23 participant_message_writer

final **RtpsReliableWriterProtocol_t** participant_message_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in participant message writer. This parameter only has effect if the matching participant message reader is configured with **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) **com.rti.dds.infrastructure.ReliabilityQosPolicyKind** (p. 1531).

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 1.0 seconds;
fast_heartbeat_period 1.0 seconds;
late_joiner_heartbeat_period 1.0 seconds;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);

samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 1;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 1s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.24 publication_writer_publish_mode

`final` **PublishModeQosPolicy** `publication_writer_publish_mode`

Publish mode policy for the built-in publication writer.

Determines whether the Discovery built-in publication **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.75.2.25 subscription_writer_publish_mode

`final` **PublishModeQosPolicy** `subscription_writer_publish_mode`

Publish mode policy for the built-in subscription writer.

Determines whether the Discovery built-in subscription **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.75.2.26 asynchronous_publisher

`final` **AsynchronousPublisherQosPolicy** `asynchronous_publisher`

Asynchronous publishing settings for the discovery **com.rti.dds.publication.Publisher** (p. 1466) and all entities that are created by it.

8.75.2.27 default_domain_announcement_period

```
final Duration_t default_domain_announcement_period
```

The period to announce a participant to the default domain 0.

The period at which a participant will announce itself to the default domain 0 using the default UDPv4 multicast group address for discovery traffic on that domain.

For domain 0, the default discovery multicast address is 239.255.0.1:7400.

To disable announcement to the default domain, set this period to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

When this period is set to a value other than **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) and **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.ignore_default_domain_announcements** (p. 658) is set to com.rti.dds.infrastructure.false, you can get information about participants running in different domains by creating a participant in domain 0 and implementing the on_data_available callback in the **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) built-in DataReader's listener.

You can learn the domain ID associated with a participant by looking at the field builtin.ParticipantBuiltinTopicData.domain_id.

[default] 30 seconds

[range] [1 nanosec,1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

See also

builtin.ParticipantBuiltinTopicData.domain_id

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.ignore_default_domain_announcements (p. 658)

8.75.2.28 ignore_default_domain_announcements

```
boolean ignore_default_domain_announcements
```

Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.

This setting only applies to participants running on the default domain 0 and using the default port mapping.

When this setting is set to com.rti.dds.infrastructure.true, a participant running on the default domain 0 will ignore announcements from participants running on different domain IDs.

When this setting is set to com.rti.dds.infrastructure.false, a participant running on the default domain 0 will provide announcements from participants running on different domain IDs to the application via the **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) built-in DataReader.

[default] com.rti.dds.infrastructure.true

See also

builtin.ParticipantBuiltinTopicData.domain_id

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.default_domain_announcement_period (p. 657)

8.75.2.29 service_request_writer

```
final RtpsReliableWriterProtocol_t service_request_writer
```

RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) writer.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);
samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.30 service_request_writer_data_lifecycle

```
final WriterDataLifecycleQosPolicy service_request_writer_data_lifecycle
```

Writer data lifecycle settings for a built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006).

8.75.2.31 service_request_writer_publish_mode

```
final PublishModeQosPolicy service_request_writer_publish_mode
```

Publish mode policy for the built-in service request writer.

Determines whether the Discovery built-in service request **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.75.2.32 service_request_reader

```
final RtpsReliableReaderProtocol_t service_request_reader
```

RTPS reliable reader protocol-related configuration settings for a built-in **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) reader.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.33 locator_reachability_assert_period

```
final Duration_t locator_reachability_assert_period
```

Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.

This setting configures the period at which this **com.rti.dds.domain.DomainParticipant** (p. 670) will ping all the locators that it has discovered from other DomainParticipants. This period should be strictly less than **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration** (p. 660).

If **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration** (p. 660) is **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) this parameter is ignored. The DomainParticipant will not assert remote locators.

[default] 20 seconds

[range] [1 nanosec,1 year]

See also

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration (p. 660)

8.75.2.34 locator_reachability_lease_duration

```
final Duration_t locator_reachability_lease_duration
```

The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

For the purpose of this explanation, we will use 'local' to refer to the DomainParticipant in which we configure locator_reachability_lease_duration and 'remote' to refer to the other DomainParticipants communicating with the local DomainParticipant.

This setting configures a timeout announced to the remote DomainParticipants. This timeout is used by the remote DomainParticipants as the maximum period by which a remote locator must be asserted by the local DomainParticipant (through a REACHABILITY PING message) before considering this locator as "unreachable" from the local DomainParticipant.

When a remote DomainParticipant detects that one of its locators is not reachable from the local DomainParticipant, it will notify the local DomainParticipant of this event. From that moment on, and until notified otherwise, the local DomainParticipant will not send RTPS messages to remote DomainParticipants using this locator.

If this value is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), the local DomainParticipant will send RTPS messages to a remote DomainParticipant on the locators announced by the remote DomainParticipant, regardless of whether or not the remote DomainParticipant can be reached using these locators.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 nanosec,1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.75.2.35 locator_reachability_change_detection_period

```
final Duration_t locator_reachability_change_detection_period
```

Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.

This setting determines the maximum period at which this DomainParticipant will check to see if its locators are reachable from other DomainParticipants according to the other DomainParticipants' **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration** (p. 660) value.

If **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration** (p. 660) is **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) this parameter is ignored. The DomainParticipant will not schedule an event to see if its locators are reachable from other DomainParticipants.

[default] 60 seconds

[range] [1 nanosec,1 year]

See also

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.locator_reachability_lease_duration (p. 660)

8.75.2.36 secure_volatile_writer

```
final RtpsReliableWriterProtocol_t secure_volatile_writer
```

RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 1.0 second;
fast_heartbeat_period 250.0 milliseconds;
late_joiner_heartbeat_period 1.0 second;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);
samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 1;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 9216 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 millisecond;
disable_positive_acks_max_sample_keep_duration 1.0 second;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 1.0 second;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.37 secure_volatile_writer_publish_mode

```
final PublishModeQosPolicy secure_volatile_writer_publish_mode
```

Publish mode policy for the built-in secure volatile writer.

Determines whether the built-in secure volatile **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.75.2.38 secure_volatile_reader

```
final RtpsReliableReaderProtocol_t secure_volatile_reader
```

RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.39 endpoint_type_object_lb_serialization_threshold

```
int endpoint_type_object_lb_serialization_threshold = 0
```

Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery.

Minimum size (in bytes) of the serialized TypeObject that will trigger the serialization of a TypeObjectLb instead of the regular TypeObject.

For example, setting this property to 1000 will trigger the serialization of the TypeObjectLb for TypeObjects whose serialized size is greater than 1000 Bytes.

The sentinel value -1 disables TypeObject compression.

[default] 0. The default value 0 enables TypeObject compression by always sending TypeObjectLb.

[range] [-1, 2147483647]

8.75.2.40 dns_tracker_polling_period

```
final Duration_t dns_tracker_polling_period
```

Initial value:

```
=
        new Duration_t(Duration_t.DURATION_INFINITE)
```

Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

RTI Connext allows the use of hostnames instead of IP addresses when configuring initial peers for specific transports (e.g.: UDPv4 and UDPv6). The DNS tracker keeps the IP addresses of these hostnames updated. The DNS tracker builds a list of hostnames from the initial peers of a DomainParticipant, queries the DNS for those hostnames, and updates the resolved IP addresses when the IP addresses change. The frequency of these queries is defined by the DNS tracker polling period. When the period is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), the tracker is disabled.

RTI Connext keeps information regarding the hostnames of peers if they are part of the **com.rti.dds.infrastructure.↩ DiscoveryQosPolicy.initial_peers** (p. 668). The information regarding peers added through the **com.rti.dds.↩ domain.DomainParticipant.add_peer** (p. 729) operation is kept only if the DNS tracker has been enabled before adding a peer.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 second,1 year], **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.75.2.41 participant_configuration_reader_resource_limits

```
final BuiltinTopicReaderResourceLimits_t participant_configuration_reader_resource_limits
```

Resource limits for the built-in topic participant configuration reader.

For details, see **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t** (p. 378).

8.75.2.42 participant_configuration_writer

```
final RtpsReliableWriterProtocol_t participant_configuration_writer
```

RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605)

[default]
low_watermark 0;
high_watermark 1;
heartbeat_period 3.0 seconds;
fast_heartbeat_period 3.0 seconds;
late_joiner_heartbeat_period 3.0 seconds;
virtual_heartbeat_period **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846);
samples_per_virtual_heartbeat **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers com.rti.dds.infrastructure.false;
heartbeats_per_max_samples 8;
min_nack_response_delay 0.0 seconds;
max_nack_response_delay 0.0 seconds;
nack_suppression_duration 0.0 seconds;
max_bytes_per_nack_response 131072 bytes;
disable_positive_acks_min_sample_keep_duration 1.0 milliseconds;
disable_positive_acks_max_sample_keep_duration 1.0 seconds;
disable_positive_acks_enable_adaptive_sample_keep_duration com.rti.dds.infrastructure.true;
disable_positive_acks_decrease_sample_keep_duration_factor 95;
disable_positive_acks_increase_sample_keep_duration_factor 150;
min_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
max_send_window_size **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat com.rti.dds.infrastructure.false;
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat com.rti.dds.infrastructure.false;

8.75.2.43 participant_configuration_writer_data_lifecycle

```
final WriterDataLifecycleQosPolicy participant_configuration_writer_data_lifecycle
```

Writer data lifecycle settings for a built-in participant configuration writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 2006). **com.rti.dds.↩ infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 2007) will always be forced to com.rti.dds.infrastructure.true.

8.75.2.44 participant_configuration_reader

```
final RtpsReliableReaderProtocol_t participant_configuration_reader
```

RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.

For details, refer to the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t** (p. 1599)

[default]
min_heartbeat_response_delay 0.0 seconds;
max_heartbeat_response_delay 0.0 seconds;
heartbeat_suppression_duration 0.0625 seconds;
nack_period 5.0 seconds;
receive_window_size 256;
round_trip_time 0.0 seconds;
app_ack_period 5.0 seconds;
samples_per_app_ack 1;

8.75.2.45 participant_configuration_writer_publish_mode

```
final PublishModeQosPolicy participant_configuration_writer_publish_mode
```

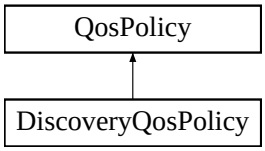
Publish mode policy for the built-in participant configuration writer.

Determines whether the Discovery built-in participant configuration **com.rti.dds.publication.DataWriter** (p. 553) publishes data synchronously or asynchronously and how.

8.76 DiscoveryQosPolicy Class Reference

<<***extension***>> *(p. 155)* Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Inheritance diagram for DiscoveryQosPolicy:



Public Attributes

- final **StringSeq enabled_transports**

  The transports available for use by the Discovery mechanism.
- final **StringSeq multicast_receive_addresses**

  *Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.*
- int **metatraffic_transport_priority**

  The transport priority to use for the Discovery meta-traffic.
- final **StringSeq initial_peers**

  Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.
- boolean **accept_unknown_peers**

  Whether to accept a new participant that is not in the initial peers list.
- boolean **enable_endpoint_discovery**

  Whether to automatically enable endpoint discovery for all the remote participants.

8.76.1 Detailed Description

<<***extension***>> *(*p. *155)* Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.76.2 Usage

This QoS policy identifies where on the network this application can *potentially* discover other applications with which to communicate.

The middleware will periodically send network packets to these locations, announcing itself to any remote applications that may be present, and will listen for announcements from those applications.

This QoS policy is an extension to the DDS standard.

See also

NDDS_DISCOVERY_PEERS (p. 222)

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 646)

8.76.3 Member Data Documentation

8.76.3.1 enabled_transports

```
final StringSeq enabled_transports
```

The transports available for use by the Discovery mechanism.

Only these transports can be used by the discovery mechanism to send meta-traffic via the builtin endpoints (built-in **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.publication.DataWriter** (p. 553)).

Also determines the unicast addresses on which the Discovery mechanism will listen for meta-traffic. These along with the `domain_id` and `participant_id` determine the unicast locators on which the Discovery mechanism can receive meta-data.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 269). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence. All the transports available to the DomainParticipant are available for use by the Discovery mechanism.

[range] Sequence of non-null,non-empty strings.

8.76.3.2 multicast_receive_addresses

```
final StringSeq multicast_receive_addresses
```

Initial value:

```
=
        new StringSeq()
```

Specifies the multicast group addresses on which discovery-related **meta-traffic** can be received by the Domain↩ Participant.

The multicast group addresses on which the Discovery mechanism will listen for meta-traffic.

Each element of this list must be a valid multicast address (IPv4 or IPv6) in the proper format (see **Address Format** (p. 224)).

The `domain_id` determines the multicast port on which the Discovery mechanism can receive meta-data.

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **com.rti.dds.↩ infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) is cleared and the RTI discovery process will not listen for discovery messages via multicast.

If NDDS_DISCOVERY_PEERS contains one or more multicast addresses, the addresses will be stored in **com.rti.↩ dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667), starting at element 0. They will be stored in the order they appear NDDS_DISCOVERY_PEERS.

Note: Currently, RTI Connext will only listen for discovery traffic on the first multicast address (element 0) in **com.rti.↩ dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667).

[default] builtin.udpv4://239.255.0.1 (See also **NDDS_DISCOVERY_PEERS** (p. 222))

[range] Sequence of length [0,1], whose elements are multicast addresses. Currently only the first multicast address (if any) is used. The rest are ignored.

See also

Address Format (p. 224)

8.76.3.3 metatraffic_transport_priority

```
int metatraffic_transport_priority
```

The transport priority to use for the Discovery meta-traffic.

The discovery metatraffic will be sent by the built-in **com.rti.dds.publication.DataWriter** (p. 553) using this transport priority.

[default] 0

[range] [0, MAX_UINT]

8.76.3.4 initial_peers

```
final StringSeq initial_peers
```

Determines the initial list of peers that will be contacted by the Discovery mechanism to send announcements about the presence of this participant.

As part of the participant discovery phase, the **com.rti.dds.domain.DomainParticipant** (p. 670) will announce itself to the domain by sending participant DATA messages. The initial_peers specifies the initial list of peers that will be contacted. A remote **com.rti.dds.domain.DomainParticipant** (p. 670) is discovered by receiving participant announcements from a remote peer. When the new remote **com.rti.dds.domain.DomainParticipant** (p. 670) has been added to the participant's database, the endpoint discovery phase commences and information about the DataWriters and DataReaders is exchanged.

Each element of this list must be a peer descriptor in the proper format (see **Peer Descriptor Format** (p. 223)).

[default] builtin.udpv4://239.255.0.1, builtin.udpv4://127.0.0.1, builtin.shmem:// (See also **NDDS_DISCOVERY_PEERS** (p. 222))

[range] Sequence of arbitrary length.

See also

Peer Descriptor Format (p. 223)

com.rti.dds.domain.DomainParticipant.add_peer() (p. 729)

8.76.3.5 accept_unknown_peers

```
boolean accept_unknown_peers
```

Whether to accept a new participant that is not in the initial peers list.

If com.rti.dds.infrastructure.false, the participant will only communicate with those in the initial peers list and those added via **com.rti.dds.domain.DomainParticipant.add_peer()** (p. 729).

If com.rti.dds.infrastructure.true, the participant will also communicate with all discovered remote participants.

Note: If accept_unknown_peers is com.rti.dds.infrastructure.false and shared memory is disabled, applications on the same node will *not* communicate if only 'localhost' is specified in the peers list. If shared memory is disabled or 'shmem://' is not specified in the peers list, to communicate with other applications on the same node through the loopback interface, you must put the actual node address or hostname in **NDDS_DISCOVERY_PEERS** (p. 222).

[default] com.rti.dds.infrastructure.true

8.76.3.6 enable_endpoint_discovery

```
boolean enable_endpoint_discovery
```

Whether to automatically enable endpoint discovery for all the remote participants.

If com.rti.dds.infrastructure.true, endpoint discovery will automatically occur for every discovered remote participant.

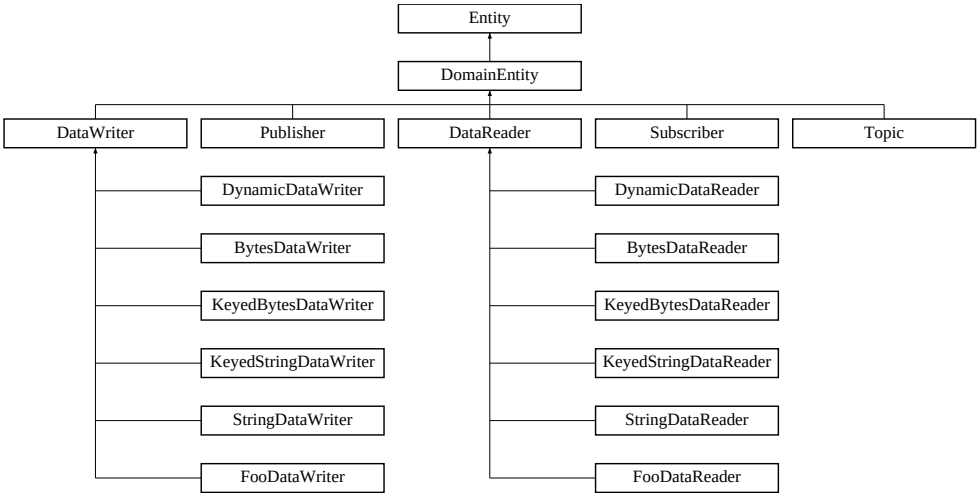
If com.rti.dds.infrastructure.false, endpoint discovery will be initially disabled and manual activation is required for each discovered participant by calling **com.rti.dds.domain.DomainParticipant.resume_endpoint_discovery** (p. 739).

[default] com.rti.dds.infrastructure.true

8.77 DomainEntity Interface Reference

<<***interface***>> *(p. 156)* Abstract base class for all DDS entities except for the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Inheritance diagram for DomainEntity:



Additional Inherited Members

8.77.1 Detailed Description

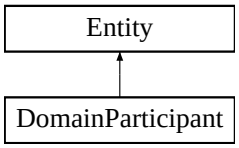
<<***interface***>> *(p. 156)* Abstract base class for all DDS entities except for the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Its sole purpose is to *conceptually* express that **com.rti.dds.domain.DomainParticipant** (p. 670) is a special kind of **com.rti.dds.infrastructure.Entity** (p. 1029) that acts as a container of all other **com.rti.dds.infrastructure.Entity** (p. 1029) but itself cannot contain other **com.rti.dds.domain.DomainParticipant** (p. 670).

8.78 DomainParticipant Interface Reference

<<***interface***>> *(p. 156)* Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 669) objects.

Inheritance diagram for DomainParticipant:



Public Member Functions

- void **get_default_flowcontroller_property** (**FlowControllerProperty_t** prop)

  <<***extension***>> *(p. 155) Copies the default* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 1059) values for this domain participant into the given* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 1059) instance.*
- void **set_default_flowcontroller_property** (**FlowControllerProperty_t** prop)

  <<***extension***>> *(p. 155) Set the default* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 1059) values for this domain participant.*
- void **get_default_topic_qos** (**TopicQos** qos)

  Copies the default ***com.rti.dds.topic.TopicQos*** *(p. 1824) values for this domain participant into the given* ***com.rti.dds.topic.TopicQos*** *(p. 1824) instance.*
- void **set_default_topic_qos** (**TopicQos** qos)

  Set the default ***com.rti.dds.topic.TopicQos*** *(p. 1824) values for this domain participant.*
- void **set_default_topic_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(p. 155) Set the default* ***com.rti.dds.topic.TopicQos*** *(p. 1824) values for this domain participant based on the input XML QoS profile.*
- void **get_default_publisher_qos** (**PublisherQos** qos)

  Copy the default ***com.rti.dds.publication.PublisherQos*** *(p. 1490) values into the provided* ***com.rti.dds.publication.PublisherQos*** *(p. 1490) instance.*
- void **set_default_publisher_qos** (**PublisherQos** qos)

  Set the default ***com.rti.dds.publication.PublisherQos*** *(p. 1490) values for this* ***DomainParticipant*** *(p. 670).*
- void **set_default_publisher_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(p. 155) Set the default* ***com.rti.dds.publication.PublisherQos*** *(p. 1490) values for this* ***DomainParticipant*** *(p. 670) based on the input XML QoS profile.*
- void **get_default_datawriter_qos** (**DataWriterQos** qos)

  <<***extension***>> *(p. 155) Copy the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 612) values into the provided* ***com.rti.dds.publication.DataWriterQos*** *(p. 612) instance.*
- void **set_default_datawriter_qos** (**DataWriterQos** qos)

  <<***extension***>> *(p. 155) Set the default DataWriterQos values for this* ***DomainParticipant*** *(p. 670).*
- void **set_default_datawriter_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(p. 155) Set the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 612) values for this domain participant based on the input XML QoS profile.*
- void **get_default_datareader_qos** (**DataReaderQos** qos)

  <<***extension***>> *(p. 155) Copy the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 517) values into the provided* ***com.rti.dds.subscription.DataReaderQos*** *(p. 517) instance.*
- void **set_default_subscriber_qos** (**SubscriberQos** qos)

*Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values for this **DomainParticipant** (p. 670).*

- void **set_default_subscriber_qos_with_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values for this **Domain↩ Participant** (p. 670) based on the input XML QoS profile.*

- void **get_default_subscriber_qos** (**SubscriberQos** qos)

  *Copy the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values into the provided **com.rti.dds.↩ subscription.SubscriberQos** (p. 1756) instance.*

- void **set_default_datareader_qos** (**DataReaderQos** qos)

  *<<**extension**>> (p. 155) Set the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this domain participant.*

- void **set_default_datareader_qos_with_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Set the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this **Domain↩ Participant** (p. 670) based on the input XML QoS profile.*

- **FlowController create_flowcontroller** (String name, **FlowControllerProperty_t** prop)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.publication.FlowController** (p. 1055) with the desired property.*

- void **delete_flowcontroller** (**FlowController** fc)

  *<<**extension**>> (p. 155) Deletes an existing **com.rti.dds.publication.FlowController** (p. 1055).*

- **Publisher create_publisher** (**PublisherQos** qos, **PublisherListener** listener, int mask)

  *Creates a **com.rti.dds.publication.Publisher** (p. 1466) with the desired QoS policies and attaches to it the specified **com.rti.dds.publication.PublisherListener** (p. 1488).*

- **Publisher create_publisher_with_profile** (String library_name, String profile_name, **PublisherListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a new **com.rti.dds.publication.Publisher** (p. 1466) object using the **com.rti.dds.↩ publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.*

- void **delete_publisher** (**Publisher** p)

  *Deletes an existing **com.rti.dds.publication.Publisher** (p. 1466).*

- **Subscriber create_subscriber** (**SubscriberQos** qos, **SubscriberListener** listener, int mask)

  *Creates a **com.rti.dds.subscription.Subscriber** (p. 1730) with the desired QoS policies and attaches to it the specified **com.rti.dds.subscription.SubscriberListener** (p. 1755).*

- **Subscriber create_subscriber_with_profile** (String library_name, String profile_name, **SubscriberListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a new **com.rti.dds.subscription.Subscriber** (p. 1730) object using the **com.rti.↩ dds.publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.*

- void **delete_subscriber** (**Subscriber** s)

  *Deletes an existing **com.rti.dds.subscription.Subscriber** (p. 1730).*

- **DataWriter create_datawriter** (**Topic** topic, **DataWriterQos** qos, **DataWriterListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.publication.DataWriter** (p. 553) that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1466).*

- **DataWriter create_datawriter_with_profile** (**Topic** topic, String library_name, String profile_name, **Data↩ WriterListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.publication.DataWriter** (p. 553) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1466).*

- void **delete_datawriter** (**DataWriter** a_datawriter)

  *<<**extension**>> (p. 155) Deletes a **com.rti.dds.publication.DataWriter** (p. 553) that belongs to the implicit **com.rti.↩ dds.publication.Publisher** (p. 1466).*

- **DataReader create_datareader** (**TopicDescription** topic, **DataReaderQos** qos, **DataReaderListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.subscription.DataReader** (p. 450) that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).*

- **DataReader create_datareader_with_profile** (**TopicDescription** topic, String library_name, String profile_name, **DataReaderListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.subscription.DataReader** (p. 450) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **delete_datareader** (**DataReader** a_datareader)

  *<<**extension**>> (p. 155) Deletes a **com.rti.dds.subscription.DataReader** (p. 450) that belongs to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).*
- **Topic create_topic** (String topic_name, String type_name, **TopicQos** qos, **TopicListener** listener, int mask)

  *Creates a **com.rti.dds.topic.Topic** (p. 1807) with the desired QoS policies and attaches to it the specified **com.rti.dds.topic.TopicListener** (p. 1822).*
- **Topic create_topic_with_profile** (String topic_name, String type_name, String library_name, String profile_name, **TopicListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a new **com.rti.dds.topic.Topic** (p. 1807) object using the **com.rti.dds.publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.*
- void **delete_topic** (**Topic** topic)

  *Deletes a **com.rti.dds.topic.Topic** (p. 1807).*
- **ContentFilteredTopic create_contentfilteredtopic** (String name, **Topic** related_topic, String filter_expression, **StringSeq** expression_parameters)

  *Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), that can be used to do content-based subscriptions.*
- **ContentFilteredTopic create_contentfilteredtopic_with_filter** (String name, **Topic** related_topic, String filter_expression, **StringSeq** expression_parameters, String filter_name)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) using the specified filter to do content-based subscriptions.*
- void **delete_contentfilteredtopic** (**ContentFilteredTopic** a_contentfilteredtopic)

  *Deletes a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).*
- **MultiTopic create_multitopic** (String name, String type_name, String subscription_expression, **StringSeq** expression_parameters)

  ***[Not supported (optional)]** Creates a MultiTopic that can be used to subscribe to multiple topics and combine/filter the received data into a resulting type.*
- void **delete_multitopic** (**MultiTopic** a_multitopic)

  ***[Not supported (optional)]** Deletes a **com.rti.dds.topic.MultiTopic** (p. 1320).*
- void **set_qos** (**DomainParticipantQos** qos)

  *Change the QoS of this **DomainParticipant** (p. 670).*
- void **set_qos_with_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Change the QoS of this domain participant using the input XML QoS profile.*
- void **get_qos** (**DomainParticipantQos** qos)

  Get the participant QoS.
- String **get_default_library** ()

  *<<**extension**>> (p. 155) Gets the default XML library associated with a **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **set_default_library** (String library_name)

  *<<**extension**>> (p. 155) Sets the default XML library for a **com.rti.dds.domain.DomainParticipant** (p. 670).*
- String **get_default_profile** ()

  *<<**extension**>> (p. 155) Gets the default XML profile associated with a **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **set_default_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Sets the default XML profile for a **com.rti.dds.domain.DomainParticipant** (p. 670).*
- String **get_default_profile_library** ()

  *<<**extension**>> (p. 155) Gets the library where the default XML QoS profile is contained for a **com.rti.dds.domain.DomainParticipant** (p. 670).*

- void **set_listener** (**DomainParticipantListener** l, int mask)

  Sets the participant listener.
- **DomainParticipantListener get_listener** ()

  Get the participant listener.
- void **get_publishers** (**PublisherSeq** publishers)

  *<<**extension**>> (p. 155) Allows the application to access all the publishers the participant has.*
- void **get_subscribers** (**SubscriberSeq** subscribers)

  *<<**extension**>> (p. 155) Allows the application to access all the subscribers the participant has.*
- **Subscriber get_builtin_subscriber** ()

  *Accesses the **built-in com.rti.dds.subscription.Subscriber** (p. 1730).*
- **FlowController lookup_flowcontroller** (String name)

  *<<**extension**>> (p. 155) Looks up an existing locally-created **com.rti.dds.publication.FlowController** (p. 1055), based on its name.*
- **Topic find_topic** (String topic_name, **Duration_t** timeout)

  *Finds an existing (or ready to exist) **com.rti.dds.topic.Topic** (p. 1807), based on its name.*
- **TopicDescription lookup_topicdescription** (String topic_name)

  *Looks up an existing, locally created **com.rti.dds.topic.TopicDescription** (p. 1820), based on its name.*
- void **ignore_participant** (**InstanceHandle_t** handle)

  *Instructs RTI Connext to locally ignore a remote **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **banish_ignored_participants** ()

  *<<**extension**>> (p. 155) Prevents ignored remote DomainParticipants from receiving traffic from the local **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **ignore_topic** (**InstanceHandle_t** handle)

  *Instructs RTI Connext to locally ignore a **com.rti.dds.topic.Topic** (p. 1807).*
- void **ignore_publication** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a publication.
- void **ignore_subscription** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a subscription.
- int **get_domain_id** ()

  Get the unique domain identifier.
- void **assert_liveliness** ()

  *Manually asserts the liveliness of this **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **delete_contained_entities** ()

  *Delete all the entities that were created by means of the "create" operations on the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **add_peer** (String peer_desc_string)

  *<<**extension**>> (p. 155) Attempt to contact one or more additional peer participants.*
- void **remove_peer** (String peer_desc_string)

  *<<**extension**>> (p. 155) Remove one or more peer participants from the list of peers with which this **com.rti.dds.domain.DomainParticipant** (p. 670) will try to communicate.*
- void **get_dns_tracker_polling_period** (**Duration_t** polling_period)

  *<<**extension**>> (p. 155) Retrieves the frequency used by the DNS tracker thread to query the DNS service.*
- void **set_dns_tracker_polling_period** (**Duration_t** polling_period)

  *<<**extension**>> (p. 155) Configures the frequency in which the DNS tracker queries the DNS service.*
- void **get_current_time** (**Time_t** current_time)

  Returns the current value of the time.
- void **get_discovered_participants** (**InstanceHandleSeq** participant_handles)

  *Returns a list of discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities.*

- void **get_discovered_participants_from_subject_name** (**InstanceHandleSeq** participant_handles, String subject_name)

  *<<**extension**>> (p. 155) Returns a list of discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities that have the given **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038).*
- void **get_discovered_participant_data** (**ParticipantBuiltinTopicData** participant_data, **InstanceHandle_t** participant_handle)

  *Returns **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) for the specified **com.rti.dds.domain.DomainParticipant** (p. 670).*
- String **get_discovered_participant_subject_name** (**InstanceHandle_t** participant_handle)

  *<<**extension**>> (p. 155) Returns **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038) for the specified **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **get_discovered_topics** (**InstanceHandleSeq** topic_handles)

  *Returns list of discovered **com.rti.dds.topic.Topic** (p. 1807) objects.*
- void **get_discovered_topic_data** (**TopicBuiltinTopicData** topic_data, **InstanceHandle_t** topic_handle)

  *Returns builtin.TopicBuiltinTopicData for the specified **com.rti.dds.topic.Topic** (p. 1807).*
- boolean **contains_entity** (**InstanceHandle_t** a_handle)

  *Completes successfully with com.rti.dds.infrastructure.true if the referenced **com.rti.dds.infrastructure.Entity** (p. 1029) is contained by the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **register_durable_subscription** (**EndpointGroup_t** group, String topic_name)

  *<<**extension**>> (p. 155) Registers a Durable Subscription on the specified **com.rti.dds.topic.Topic** (p. 1807) on all Persistence Services.*
- void **delete_durable_subscription** (**EndpointGroup_t** group)

  *<<**extension**>> (p. 155) Deletes an existing Durable Subscription on all Persistence Services.*
- void **resume_endpoint_discovery** (**InstanceHandle_t** remote_participant_handle)

  *<<**extension**>> (p. 155) Initiates endpoint discovery with the specified remote **com.rti.dds.domain.DomainParticipant** (p. 670).*
- void **register_contentfilter** (String filter_name, **ContentFilter** contentfilter)

  *<<**extension**>> (p. 155) Register a content filter which can be used to create a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).*
- **ContentFilter** **lookup_contentfilter** (String filter_name)

  *<<**extension**>> (p. 155) Lookup a content filter previously registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740).*
- void **unregister_contentfilter** (String filter_name)

  *<<**extension**>> (p. 155) Unregister a content filter previously registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740).*
- **TypeCode** **get_typecode** (String type_name)

  *<<**extension**>> (p. 155) Retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) of a type registered with the **com.rti.dds.domain.DomainParticipant** (p. 670) based on the registration name.*
- void **get_participant_protocol_status** (**DomainParticipantProtocolStatus** status)

  *<<**extension**>> (p. 155) Get the domain participant protocol status for this participant.*
- **Publisher** **get_implicit_publisher** ()

  *<<**extension**>> (p. 155) Returns the implicit **com.rti.dds.publication.Publisher** (p. 1466). If an implicit Publisher does not already exist, this creates one.*
- **Subscriber** **get_implicit_subscriber** ()

  *<<**extension**>> (p. 155) Returns the implicit **com.rti.dds.subscription.Subscriber** (p. 1730). If an implicit Subscriber does not already exist, this creates one.*
- abstract **Publisher** **lookup_publisher_by_name** (String publisher_name)

  *<<**extension**>> (p. 155) Looks up a **com.rti.dds.publication.Publisher** (p. 1466) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).*
- abstract **Subscriber** **lookup_subscriber_by_name** (String subscriber_name)

*<<**extension**>> (p. 155) Retrieves a **com.rti.dds.subscription.Subscriber** (p. 1730) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).*

- abstract **DataWriter lookup_datawriter_by_name** (String datawriter_full_name)

  *<<**extension**>> (p. 155) Looks up a **com.rti.dds.publication.DataWriter** (p. 553) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).*

- abstract **DataReader lookup_datareader_by_name** (String datareader_full_name)

  *<<**extension**>> (p. 155) Retrieves up a **com.rti.dds.subscription.DataReader** (p. 450) by its entity name in this **com.rti.dds.domain.DomainParticipant** (p. 670).*

- void **take_discovery_snapshot** ()

  Take a snapshot of the remote participants discovered by a local one.

- void **take_discovery_snapshot** (String file_name)

  Take a snapshot of the remote participants discovered by a local one.

Static Public Attributes

- static final **TopicQos TOPIC_QOS_DEFAULT** = new **TopicQos**()

  *Special value for creating a **com.rti.dds.topic.Topic** (p. 1807) with default QoS.*

- static final **TopicQos TOPIC_QOS_PRINT_ALL** = new **TopicQos**()

  *Special value which can be supplied to **com.rti.dds.topic.TopicQos.toString(TopicQos baseQos, QosPrintFormat format)** (p. 1826) indicating that all of the QoS should be printed.*

- static final **PublisherQos PUBLISHER_QOS_DEFAULT** = new **PublisherQos**()

  *Special value for creating a **com.rti.dds.publication.Publisher** (p. 1466) with default QoS.*

- static final **PublisherQos PUBLISHER_QOS_PRINT_ALL** = new **PublisherQos**()

  *Special value which can be supplied to **com.rti.dds.publication.PublisherQos.toString(PublisherQos baseQos, QosPrintFormat format)** (p. 1491) indicating that all of the QoS should be printed.*

- static final **SubscriberQos SUBSCRIBER_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.subscription.SubscriberQos.toString(SubscriberQos baseQos, QosPrintFormat format)** (p. 1758) indicating that all of the QoS should be printed.*

- static final **SubscriberQos SUBSCRIBER_QOS_DEFAULT**

  *Special value for creating a **com.rti.dds.subscription.Subscriber** (p. 1730) with default QoS.*

- static final **DomainParticipantQos DOMAINPARTICIPANT_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.domain.DomainParticipantQos.toString(DomainParticipantQos baseQos, QosPrintFormat format)** (p. 798) indicating that all of the QoS should be printed.*

- static final **DomainParticipantFactoryQos DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL**

  *Special value which can be supplied to **com.rti.dds.domain.DomainParticipantFactoryQos.toString(DomainParticipantFactoryQos baseQos, QosPrintFormat format)** (p. 789) indicating that all of the QoS should be printed.*

- static final **FlowControllerProperty_t FLOW_CONTROLLER_PROPERTY_DEFAULT**

  *<<**extension**>> (p. 155) Special value for creating a **com.rti.dds.publication.FlowController** (p. 1055) with default property.*

- static final String **SQLFILTER_NAME**

  *<<**extension**>> (p. 155) The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

- static final String **STRINGMATCHFILTER_NAME**

  *<<**extension**>> (p. 155) The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

8.78.1 Detailed Description

<<***interface***>> *(*p. *156)* Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 669) objects.

The **DomainParticipant** (p. 670) object plays several roles:

- It acts as a container for all other **com.rti.dds.infrastructure.Entity** (p. 1029) objects.
- It acts as a *factory* for the **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.topic.Topic** (p. 1807) and **com.rti.dds.topic.MultiTopic** (p. 1320) **com.rti.dds.↩ infrastructure.Entity** (p. 1029) objects.
- It represents the participation of the application on a communication plane that isolates applications running on the same set of physical computers from each other. A domain establishes a virtual network linking all applications that share the same `domainId` and isolating them from applications running on different domains. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other.
- It provides administration services in the domain, offering operations that allow the application to ignore locally any information about a given participant (**com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723)), publication (**com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 725)), subscription (**com.rti.dds.domain.DomainParticipant.ignore_subscription()** (p. 726)) or topic (**com.rti.dds.domain.↩ DomainParticipant.ignore_topic()** (p. 724)).

The following operations may be called even if the **com.rti.dds.domain.DomainParticipant** (p. 670) is not enabled. Operations NOT in this list will fail with **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) \ if called on a disabled **DomainParticipant** (p. 670).

- **com.rti.dds.infrastructure.Entity.enable** (p. 1032),
- **com.rti.dds.domain.DomainParticipant.set_qos** (p. 714), **com.rti.dds.domain.DomainParticipant.set_↩ qos_with_profile** (p. 714), **com.rti.dds.domain.DomainParticipant.get_qos** (p. 715),
- **com.rti.dds.domain.DomainParticipant.set_listener** (p. 718), **com.rti.dds.domain.DomainParticipant.get↩ _listener** (p. 719),
- Factory operations: **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 691), **com.rti.↩ dds.domain.DomainParticipant.create_topic** (p. 706), **com.rti.dds.domain.DomainParticipant.create↩ _topic_with_profile** (p. 708), **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693), **com.↩ rti.dds.domain.DomainParticipant.create_publisher_with_profile** (p. 694), **com.rti.dds.domain.Domain↩ Participant.create_subscriber** (p. 697), **com.rti.dds.domain.DomainParticipant.create_subscriber_with↩ _profile** (p. 698), **com.rti.dds.domain.DomainParticipant.delete_flowcontroller** (p. 692), **com.rti.dds.↩ domain.DomainParticipant.delete_topic** (p. 709), **com.rti.dds.domain.DomainParticipant.delete_publisher** (p. 696), **com.rti.dds.domain.DomainParticipant.delete_subscriber** (p. 699), **com.rti.dds.domain.Domain↩ Participant.set_default_topic_qos** (p. 679), **com.rti.dds.domain.DomainParticipant.set_default_topic_↩ qos_with_profile** (p. 680), **com.rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 679), **com.↩ rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 682), **com.rti.dds.domain.Domain↩ Participant.set_default_publisher_qos_with_profile** (p. 683), **com.rti.dds.domain.DomainParticipant.↩ get_default_publisher_qos** (p. 681), **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos**

(p. 687), **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos_with_profile** (p. 688), **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 689), **com.rti.dds.domain.DomainParticipant.delete_contained_entities** (p. 728), **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690), **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos_with_profile** (p. 691), **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 686), **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 685), **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos_with_profile** (p. 686), **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 684), **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716), **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717) **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678), **com.rti.dds.domain.DomainParticipant.get_default_flowcontroller_property** (p. 677), ;

- Operations for looking up topics: **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 722);
- Operations that access status: **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034).

QoS:

com.rti.dds.domain.DomainParticipantQos (p. 795)

Status:

Status Kinds (p. 262)

Listener:

com.rti.dds.domain.DomainParticipantListener (p. 792)

See also

Operations Allowed in Listener Callbacks (p. 1238)

8.78.2 Member Function Documentation

8.78.2.1 get_default_flowcontroller_property()

```
void get_default_flowcontroller_property (
            FlowControllerProperty_t prop )
```

<<***extension***>> *(p. 155)* Copies the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) values for this domain participant into the given **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) instance.

The retrieved `property` will match the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678), or else, if the call was never made, the default values listed in **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059).

MT Safety:

UNSAFE. It is not safe to retrieve the default flow controller properties from a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678)

Parameters

prop	<<***in***>> *(*p. *156)* Default property to be retrieved. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 49)

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 691)

8.78.2.2 set_default_flowcontroller_property()

```
void set_default_flowcontroller_property (
             FlowControllerProperty_t prop )
```

<<***extension***>> *(*p. *155)* Set the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) values for this domain participant.

This default value will be used for newly created **com.rti.dds.publication.FlowController** (p. 1055) if **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) is specified as the `property` parameter when **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 691) is called.

Precondition

The specified property values must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default flow controller properties for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678) , **com.rti.dds.domain.DomainParticipant.get_default_flowcontroller_property** (p. 677) or calling **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 691) with **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) as the `qos` parameter.

Parameters

prop	<<***in***>> *(*p. *156)* Default property to be set. The special value **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) may be passed as `property` to indicate that the default property should be reset to the default values the factory would use if **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 49)

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 691)

8.78.2.3 get_default_topic_qos()

```
void get_default_topic_qos (
              TopicQos qos )
```

Copies the default **com.rti.dds.topic.TopicQos** (p. 1824) values for this domain participant into the given **com.rti.dds.topic.TopicQos** (p. 1824) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679), or **com.rti.dds.domain.DomainParticipant.set_default_topic_qos_with_profile** (p. 680) or else, if the call was never made, the default values listed in **com.rti.dds.topic.TopicQos** (p. 1824).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default Topic QoS from a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679)

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be retrieved. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT (p. 45)

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.78.2.4 set_default_topic_qos()

```
void set_default_topic_qos (
              TopicQos qos )
```

Set the default **com.rti.dds.topic.TopicQos** (p. 1824) values for this domain participant.

This default value will be used for newly created **com.rti.dds.topic.Topic** (p. 1807) if **com.rti.dds.domain.Domain↩ Participant.TOPIC_QOS_DEFAULT** (p. 45) is specified as the qos parameter when **com.rti.dds.domain.Domain↩ Participant.create_topic** (p. 706) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.↩ dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679), **com.↩ rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 679) or calling **com.rti.dds.domain.Domain↩ Participant.create_topic** (p. 706) with **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) as the qos parameter.

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be set. The special value **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) may be passed as qos to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT (p. 45)

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.78.2.5 set_default_topic_qos_with_profile()

```
void set_default_topic_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.topic.TopicQos** (p. 1824) values for this domain participant based on the input XML QoS profile.

This default value will be used for newly created **com.rti.dds.topic.Topic** (p. 1807) if **com.rti.dds.domain.Domain↩ Participant.TOPIC_QOS_DEFAULT** (p. 45) is specified as the `qos` parameter when **com.rti.dds.domain.Domain↩ Participant.create_topic** (p. 706) is called.

Precondition

The **com.rti.dds.topic.TopicQos** (p. 1824) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679), **com.↩ rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 679) or calling **com.rti.dds.domain.Domain↩ Participant.create_topic** (p. 706) with **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

If the input profile cannot be found the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT (p. 45)

com.rti.dds.domain.DomainParticipant.create_topic_with_profile (p. 708)

8.78.2.6 get_default_publisher_qos()

```
void get_default_publisher_qos (
             PublisherQos qos )
```

Copy the default **com.rti.dds.publication.PublisherQos** (p. 1490) values into the provided **com.rti.dds.publication.↩ PublisherQos** (p. 1490) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.↩ DomainParticipant.set_default_publisher_qos** (p. 682), or **com.rti.dds.domain.DomainParticipant.set_default↩ _publisher_qos_with_profile** (p. 683), or else, if the call was never made, the default values listed in **com.rti.dds.↩ publication.PublisherQos** (p. 1490).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **com.rti.dds.domain.DomainParticipant.get_default_publisher↩ _qos** (p. 681), will be used to create the **com.rti.dds.publication.Publisher** (p. 1466).

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher↩ _qos** (p. 682)

Parameters

qos	<<***inout***>> *(p. 156)* Qos to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 46)

com.rti.dds.domain.DomainParticipant.create_publisher (p. 693)

8.78.2.7 set_default_publisher_qos()

```
void set_default_publisher_qos (
             PublisherQos qos )
```

Set the default **com.rti.dds.publication.PublisherQos** (p. 1490) values for this **DomainParticipant** (p. 670).

This set of default values will be used for a newly created **com.rti.dds.publication.Publisher** (p. 1466) if **com.rti.dds.↩ domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) is specified as the `qos` parameter when **com.rti.↩ dds.domain.DomainParticipant.create_publisher** (p. 693) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 682), **com.rti.dds.domain.DomainParticipant.get_default_publisher_qos** (p. 681) or calling **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693) with **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be set. The special value **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 682) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 46)

com.rti.dds.domain.DomainParticipant.create_publisher (p. 693)

8.78.2.8 set_default_publisher_qos_with_profile()

```
void set_default_publisher_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.publication.PublisherQos** (p. 1490) values for this **DomainParticipant** (p. 670) based on the input XML QoS profile.

This set of default values will be used for a newly created **com.rti.dds.publication.Publisher** (p. 1466) if **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693) is called.

Precondition

The **com.rti.dds.publication.PublisherQos** (p. 1490) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 682), **com.rti.dds.domain.DomainParticipant.get_default_publisher_qos** (p. 681) or calling **com.rti.dds.domain.↩ DomainParticipant.create_publisher** (p. 693) with **com.rti.dds.domain.DomainParticipant.PUBLISHER_↩ QOS_DEFAULT** (p. 46) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 46)

com.rti.dds.domain.DomainParticipant.create_publisher_with_profile (p. 694)

8.78.2.9 get_default_datawriter_qos()

```
void get_default_datawriter_qos (
               DataWriterQos qos )
```

<<***extension***>> *(p. 155)* Copy the default **com.rti.dds.publication.DataWriterQos** (p. 612) values into the provided **com.rti.dds.publication.DataWriterQos** (p. 612) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.↩ DomainParticipant.set_default_datawriter_qos** (p. 685), or **com.rti.dds.domain.DomainParticipant.set_default↩ _datawriter_qos_with_profile** (p. 686), or else, if the call was never made, the default values listed in **com.rti.dds.↩ publication.DataWriterQos** (p. 612).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataWriterQoS from a DomainPartipant while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 685).

Parameters

qos	<<***inout***>> *(p. 156)* Qos to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.78.2.10 set_default_datawriter_qos()

```
void set_default_datawriter_qos (
              DataWriterQos qos )
```

<<***extension***>> *(p. 155)* Set the default DataWriterQos values for this **DomainParticipant** (p. 670).

This set of default values will be inherited for a newly created **com.rti.dds.publication.Publisher** (p. 1466).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 685) or **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 684) or calling **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 700) with **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be set. The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 685) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.78.2.11 set_default_datawriter_qos_with_profile()

```
void set_default_datawriter_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.publication.DataWriterQos** (p. 612) values for this domain participant based on the input XML QoS profile.

This set of default values will be inherited for a newly created **com.rti.dds.publication.Publisher** (p. 1466).

Precondition

The **com.rti.dds.publication.DataWriterQos** (p. 612) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 685) or **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 684)

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.78.2.12 get_default_datareader_qos()

```
void get_default_datareader_qos (
              DataReaderQos qos )
```

<<***extension***>> *(p. 155)* Copy the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values into the provided **com.rti.dds.subscription.DataReaderQos** (p. 517) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690), or **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos_with_profile** (p. 691), or else, if the call was never made, the default values listed in **com.rti.dds.subscription.DataReaderQos** (p. 517).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default DataReader QoS from a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690).

Parameters

qos	<<***inout***>> *(p. 156)* Qos to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.78.2.13 set_default_subscriber_qos()

```
void set_default_subscriber_qos (
              SubscriberQos qos )
```

Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values for this **DomainParticipant** (p. 670).

This set of default values will be used for a newly created **com.rti.dds.subscription.Subscriber** (p. 1730) if **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687), **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 689) or calling **com.rti.↩ dds.domain.DomainParticipant.create_subscriber** (p. 697) with **com.rti.dds.domain.DomainParticipant.↩ SUBSCRIBER_QOS_DEFAULT** (p. 47) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be set. The special value **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.78.2.14 set_default_subscriber_qos_with_profile()

```
void set_default_subscriber_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values for this **DomainParticipant** (p. 670) based on the input XML QoS profile.

This set of default values will be used for a newly created **com.rti.dds.subscription.Subscriber** (p. 1730) if **com.↩ rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697) is called.

Precondition

The **com.rti.dds.subscription.SubscriberQos** (p. 1756) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687), **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 689) or calling **com.rti.↩ dds.domain.DomainParticipant.create_subscriber** (p. 697) with **com.rti.dds.domain.DomainParticipant.↩ SUBSCRIBER_QOS_DEFAULT** (p. 47) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 47)

com.rti.dds.domain.DomainParticipant.create_subscriber_with_profile (p. 698)

8.78.2.15 get_default_subscriber_qos()

```
void get_default_subscriber_qos (
               SubscriberQos qos )
```

Copy the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) values into the provided **com.rti.dds.subscription.SubscriberQos** (p. 1756) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687), or **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos_with_profile** (p. 688), or else, if the call was never made, the default values listed in **com.rti.dds.subscription.SubscriberQos** (p. 1756).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 697) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 689), will be used to create the **com.rti.dds.subscription.Subscriber** (p. 1730).

MT Safety:

UNSAFE. It is not safe to retrieve the default Subscriber QoS from a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687).

Parameters

qos	<<***inout***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 47)

com.rti.dds.domain.DomainParticipant.create_subscriber (p. 697)

8.78.2.16 set_default_datareader_qos()

```
void set_default_datareader_qos (
             DataReaderQos qos )
```

<<***extension***>> *(*p. *155)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this domain participant.

This set of default values will be inherited for a newly created **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.↩ dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690) or **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 686).

Parameters

qos	<<***in***>> *(*p. *156)* Default qos to be set. The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.78.2.17 set_default_datareader_qos_with_profile()

```
void set_default_datareader_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this **DomainParticipant** (p. 670) based on the input XML QoS profile.

This set of default values will be inherited for a newly created **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

The **com.rti.dds.subscription.DataReaderQos** (p. 517) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a **DomainParticipant** (p. 670) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690) or **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 686).

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.78.2.18 create_flowcontroller()

```
FlowController create_flowcontroller (
            String name,
             FlowControllerProperty_t prop )
```

<<***extension***>> *(*p. *155)* Creates a **com.rti.dds.publication.FlowController** (p. 1055) with the desired property.

The created **com.rti.dds.publication.FlowController** (p. 1055) is associated with a **com.rti.dds.publication.DataWriter** (p. 553) via **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1497). A single FlowController may service multiple DataWriters instances, even if they belong to a different **com.rti.dds.publication.Publisher** (p. 1466). The `property` determines how the FlowController shapes the network traffic.

Precondition

The specified `property` must be consistent, or the operation will fail and no **com.rti.dds.publication.FlowController** (p. 1055) will be created.

MT Safety:

UNSAFE. If **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) is used for `property`, it is not safe to create the flow controller while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 678) or trying to lookup that flow controller with **com.rti.dds.domain.DomainParticipant.lookup_flowcontroller** (p. 721).

Parameters

name	<<***in***>> *(*p. *156)* name of the **com.rti.dds.publication.FlowController** (p. 1055) to create. A **com.rti.dds.publication.DataWriter** (p. 553) is associated with a **com.rti.dds.publication.FlowController** (p. 1055) by name. Limited to 255 characters.
prop	<<***in***>> *(*p. *156)* property to be used for creating the new **com.rti.dds.publication.FlowController** (p. 1055). The special value **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) can be used to indicate that the **com.rti.dds.publication.FlowController** (p. 1055) should be created with the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.

Returns

Newly created flow controller object or NULL on failure.

See also

com.rti.dds.publication.FlowControllerProperty_t (p. 1059) for rules on consistency among property

com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 49)

com.rti.dds.domain.DomainParticipant.get_default_flowcontroller_property (p. 677)

8.78.2.19 delete_flowcontroller()

```
void delete_flowcontroller (
              FlowController fc )
```

<<***extension***>> *(*p. *155)* Deletes an existing **com.rti.dds.publication.FlowController** (p. 1055).

Precondition

The **com.rti.dds.publication.FlowController** (p. 1055) must not have any attached **com.rti.dds.publication.↩ DataWriter** (p. 553) objects. If there are any attached **com.rti.dds.publication.DataWriter** (p. 553) objects, it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

The **com.rti.dds.publication.FlowController** (p. 1055) must have been created by this **com.rti.dds.domain.↩ DomainParticipant** (p. 670), or else it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_↩ NOT_MET** (p. 1598).

Postcondition

The **com.rti.dds.publication.FlowController** (p. 1055) is deleted if this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

fc	<<***in***>> *(*p. *156)* The **com.rti.dds.publication.FlowController** (p. 1055) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.78.2.20 create_publisher()

```
Publisher create_publisher (
             PublisherQos qos,
             PublisherListener listener,
            int mask )
```

Creates a **com.rti.dds.publication.Publisher** (p. 1466) with the desired QoS policies and attaches to it the specified **com.rti.dds.publication.PublisherListener** (p. 1488).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.publication.Publisher** (p. 1466) will be created.

MT Safety:

UNSAFE. If **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) is used for `qos`, it is not safe to create the publisher while another thread may be simultaneously calling **com.rti.dds.domain.↩ DomainParticipant.set_default_publisher_qos** (p. 682).

Parameters

qos	<<***in***>> *(p. 156)* QoS to be used for creating the new **com.rti.dds.publication.Publisher** (p. 1466). The special value **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) can be used to indicate that the **com.rti.dds.publication.Publisher** (p. 1466) should be created with the default **com.rti.dds.publication.PublisherQos** (p. 1490) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.
listener	<<***in***>> *(p. 156)*. Listener to be attached to the newly created **com.rti.dds.publication.Publisher** (p. 1466).
mask	<<***in***>> (p. *156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.PublisherQos (p. 1490) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 46)

com.rti.dds.domain.DomainParticipant.create_publisher_with_profile (p. 694)

com.rti.dds.domain.DomainParticipant.get_default_publisher_qos (p. 681)

com.rti.dds.publication.Publisher.set_listener (p. 1480)

8.78.2.21 create_publisher_with_profile()

```
Publisher create_publisher_with_profile (
            String library_name,
            String profile_name,
             PublisherListener listener,
            int mask )
```

<<***extension***>> *(p. 155)* Creates a new **com.rti.dds.publication.Publisher** (p. 1466) object using the **com.rti.dds.↩ publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.

Precondition

The **com.rti.dds.publication.PublisherQos** (p. 1490) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.publication.Publisher** (p. 1466) will be created.

Parameters

library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).
listener	<<***in***>> (p. *156).* Listener to be attached to the newly created **com.rti.dds.publication.Publisher** (p. 1466).
mask	<<***in***>> (p. *156).* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.PublisherQos (p. 1490) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_publisher (p. 693)

com.rti.dds.domain.DomainParticipant.get_default_publisher_qos (p. 681)

com.rti.dds.publication.Publisher.set_listener (p. 1480)

8.78.2.22 delete_publisher()

```
void delete_publisher (
                Publisher p )
```

Deletes an existing **com.rti.dds.publication.Publisher** (p. 1466).

Precondition

The **com.rti.dds.publication.Publisher** (p. 1466) must not have any attached **com.rti.dds.publication.Data↩ Writer** (p. 553) objects. If there are existing **com.rti.dds.publication.DataWriter** (p. 553) objects, it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

com.rti.dds.publication.Publisher (p. 1466) must have been created by this **com.rti.dds.domain.Domain↩ Participant** (p. 670), or else it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.publication.Publisher** (p. 1466) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

p	<<***in***>> *(p. 156)* **com.rti.dds.publication.Publisher** (p. 1466) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.78.2.23 create_subscriber()

```
Subscriber create_subscriber (
             SubscriberQos qos,
             SubscriberListener listener,
            int mask )
```

Creates a **com.rti.dds.subscription.Subscriber** (p. 1730) with the desired QoS policies and attaches to it the specified **com.rti.dds.subscription.SubscriberListener** (p. 1755).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.subscription.↩ Subscriber** (p. 1730) will be created.

MT Safety:

UNSAFE. If **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) is used for `qos`, it is not safe to create the subscriber while another thread may be simultaneously calling **com.rti.dds.domain.↩ DomainParticipant.set_default_subscriber_qos** (p. 687).

Parameters

qos	<<***in***>> *(p. 156)* QoS to be used for creating the new **com.rti.dds.subscription.Subscriber** (p. 1730). The special value **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) can be used to indicate that the **com.rti.dds.subscription.Subscriber** (p. 1730) should be created with the default **com.rti.dds.subscription.SubscriberQos** (p. 1756) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.
listener	<<***in***>> *(p. 156)*. Listener to be attached to the newly created **com.rti.dds.subscription.Subscriber** (p. 1730).
mask	<<***in***>> *(p. 156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.SubscriberQos (p. 1756) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 47)

com.rti.dds.domain.DomainParticipant.create_subscriber_with_profile (p. 698)

com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 689)

com.rti.dds.subscription.Subscriber.set_listener (p. 1748)

8.78.2.24 create_subscriber_with_profile()

```
Subscriber create_subscriber_with_profile (
            String library_name,
            String profile_name,
             SubscriberListener listener,
            int mask )
```

<<***extension***>> *(p. 155)* Creates a new **com.rti.dds.subscription.Subscriber** (p. 1730) object using the **com.rti.dds.publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.

Precondition

The **com.rti.dds.subscription.SubscriberQos** (p. 1756) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.subscription.Subscriber** (p. 1730) will be created.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).
listener	<<***in***>> *(p. 156).* Listener to be attached to the newly created **com.rti.dds.subscription.Subscriber** (p. 1730).
mask	<<***in***>> *(p. 156).* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.SubscriberQos (p. 1756) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_subscriber (p. 697)

com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 689)

com.rti.dds.subscription.Subscriber.set_listener (p. 1748)

8.78.2.25 delete_subscriber()

```
void delete_subscriber (
              Subscriber s )
```

Deletes an existing **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

The **com.rti.dds.subscription.Subscriber** (p. 1730) must not have any attached **com.rti.dds.subscription.↩ DataReader** (p. 450) objects. If there are existing **com.rti.dds.subscription.DataReader** (p. 450) objects, it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

The **com.rti.dds.subscription.Subscriber** (p. 1730) must have been created by this **com.rti.dds.domain.↩ DomainParticipant** (p. 670), or else it will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION↩ _NOT_MET** (p. 1598).

Postcondition

A Listener installed on the **com.rti.dds.subscription.Subscriber** (p. 1730) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

s	<<***in***>> *(p. 156)* **com.rti.dds.subscription.Subscriber** (p. 1730) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.78.2.26 create_datawriter()

```
DataWriter create_datawriter (
            Topic topic,
            DataWriterQos qos,
            DataWriterListener listener,
           int mask )
```

<<***extension***>> *(*p. *155)* Creates a **com.rti.dds.publication.DataWriter** (p. 553) that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1466).

Precondition

The given **com.rti.dds.topic.Topic** (p. 1807) must have been created from the same **DomainParticipant** (p. 670) as the implicit Publisher. If it was created from a different **DomainParticipant** (p. 670), this method will fail.

The **com.rti.dds.publication.DataWriter** (p. 553) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **com.rti.dds.domain.DomainParticipant.↩ PUBLISHER_QOS_DEFAULT** (p. 46) when the following methods are called: **com.rti.dds.domain.Domain↩ Participant.create_datawriter** (p. 700), **com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile** (p. 701), or **com.rti.dds.domain.DomainParticipant.get_implicit_publisher** (p. 744).

MT Safety:

UNSAFE. If **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) is used for the `qos` parameter, it is not safe to create the DataWriter while another thread may be simultaneously calling **com.rti.dds.↩ domain.DomainParticipant.set_default_datawriter_qos** (p. 685).

Parameters

topic	<<***in***>> *(*p. *156)* The **com.rti.dds.topic.Topic** (p. 1807) that the **com.rti.dds.publication.DataWriter** (p. 553) will be associated with. Cannot be NULL.
qos	<<***in***>> *(*p. *156)* QoS to be used for creating the new **com.rti.dds.publication.DataWriter** (p. 553). The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 553) should be created with the default **com.rti.dds.publication.DataWriterQos** (p. 612) set in the implicit **com.rti.dds.publication.Publisher** (p. 1466). The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_USE_TOPIC_QOS** (p. 71) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 553) should be created with the combination of the default **com.rti.dds.publication.DataWriterQos** (p. 612) set on the **com.rti.dds.publication.Publisher** (p. 1466) and the **com.rti.dds.topic.TopicQos** (p. 1824) of the **com.rti.dds.topic.Topic** (p. 1807). Cannot be NULL.
listener	<<***in***>> *(*p. *156)* The listener of the **com.rti.dds.publication.DataWriter** (p. 553).
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.publication.DataWriter** (p. 553) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT (p. 71)

com.rti.dds.publication.Publisher.DATAWRITER_QOS_USE_TOPIC_QOS (p. 71)

com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile (p. 701)

com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos (p. 684)

com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 744)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.publication.DataWriter.set_listener (p. 558)

8.78.2.27 create_datawriter_with_profile()

```
DataWriter create_datawriter_with_profile (
              Topic topic,
             String library_name,
             String profile_name,
              DataWriterListener listener,
             int mask )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.publication.DataWriter** (p. 553) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1466).

Precondition

The given **com.rti.dds.topic.Topic** (p. 1807) must have been created from the same **DomainParticipant** (p. 670) as the implicit Publisher. If it was created from a different **DomainParticipant** (p. 670), this method will return NULL.

The **com.rti.dds.publication.DataWriter** (p. 553) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) when the following methods are called: **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 700), **com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile** (p. 701), or **com.rti.dds.domain.DomainParticipant.get_implicit_publisher** (p. 744)

Parameters

topic	<<***in***>> *(p. 156)* The **com.rti.dds.topic.Topic** (p. 1807) that the **com.rti.dds.publication.DataWriter** (p. 553) will be associated with. Cannot be NULL.

Parameters

library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).
listener	<<***in***>> *(*p. *156)* The listener of the **com.rti.dds.publication.DataWriter** (p. 553).
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.publication.DataWriter** (p. 553) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_datawriter (p. 700)

com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos (p. 684)

com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 744)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.publication.DataWriter.set_listener (p. 558)

8.78.2.28 delete_datawriter()

```
void delete_datawriter (
            DataWriter a_datawriter )
```

<<***extension***>> *(*p. *155)* Deletes a **com.rti.dds.publication.DataWriter** (p. 553) that belongs to the implicit **com.rti.dds.publication.Publisher** (p. 1466).

The deletion of the **com.rti.dds.publication.DataWriter** (p. 553) will automatically unregister all instances.

Precondition

If the **com.rti.dds.publication.DataWriter** (p. 553) does not belong to the implicit **com.rti.dds.publication.Publisher** (p. 1466), the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.publication.DataWriter** (p. 553) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datawriter	<<***in***>> *(*p. *156)* The **com.rti.dds.publication.DataWriter** (p. 553) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

See also

com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 744)

8.78.2.29 create_datareader()

```
DataReader create_datareader (
             TopicDescription topic,
             DataReaderQos qos,
             DataReaderListener listener,
            int mask )
```

<<***extension***>> *(*p. *155)* Creates a **com.rti.dds.subscription.DataReader** (p. 450) that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

The given **com.rti.dds.topic.TopicDescription** (p. 1820) must have been created from the same **Domain↩ Participant** (p. 670) as the implicit Subscriber. If it was created from a different **DomainParticipant** (p. 670), this method will return NULL.

The **com.rti.dds.subscription.DataReader** (p. 450) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **com.rti.dds.domain.DomainParticipant.↩ SUBSCRIBER_QOS_DEFAULT** (p. 47) when the following methods are called: **com.rti.dds.domain.Domain↩ Participant.create_datareader** (p. 703), **com.rti.dds.domain.DomainParticipant.create_datareader_with_profile** (p. 704), or **com.rti.dds.domain.DomainParticipant.get_implicit_subscriber** (p. 744).

MT Safety:

UNSAFE. If **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **com.rti.↩ dds.domain.DomainParticipant.set_default_datareader_qos** (p. 690).

Parameters

topic	<<***in***>> *(p. 156)* The **com.rti.dds.topic.TopicDescription** (p. 1820) that the **com.rti.dds.subscription.DataReader** (p. 450) will be associated with. Cannot be NULL.
qos	<<***in***>> *(p. 156)* The qos of the **com.rti.dds.subscription.DataReader** (p. 450). The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 450) should be created with the default **com.rti.dds.subscription.DataReaderQos** (p. 517) set in the implicit **com.rti.dds.subscription.Subscriber** (p. 1730). If **com.rti.dds.topic.TopicDescription** (p. 1820) is of type **com.rti.dds.topic.Topic** (p. 1807) or **com.rti.dds.topic.ContentFilteredTopic** (p. 436), the special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 80) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 450) should be created with the combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 517) set on the implicit **com.rti.dds.subscription.Subscriber** (p. 1730) and the **com.rti.dds.topic.TopicQos** (p. 1824) (in the case of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), the **com.rti.dds.topic.TopicQos** (p. 1824) of the related **com.rti.dds.topic.Topic** (p. 1807)). if **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 80) is used, `topic` cannot be a **com.rti.dds.topic.MultiTopic** (p. 1320). Cannot be NULL.
listener	<<***in***>> *(p. 156)* The listener of the **com.rti.dds.subscription.DataReader** (p. 450).
mask	<<***in***>> *(p. 156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.subscription.DataReader** (p. 450) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_datareader_with_profile (p. 704)

com.rti.dds.domain.DomainParticipant.get_default_datareader_qos (p. 686)

com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 744)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.subscription.DataReader.set_listener (p. 459)

8.78.2.30 create_datareader_with_profile()

```
DataReader create_datareader_with_profile (
             TopicDescription topic,
            String library_name,
            String profile_name,
             DataReaderListener listener,
            int mask )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.subscription.DataReader** (p. 450) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

The given **com.rti.dds.topic.TopicDescription** (p. 1820) must have been created from the same **Domain↩ Participant** (p. 670) as the implicit subscriber. If it was created from a different **DomainParticipant** (p. 670), this method will return NULL.

The **com.rti.dds.subscription.DataReader** (p. 450) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **com.rti.dds.domain.DomainParticipant.↩ SUBSCRIBER_QOS_DEFAULT** (p. 47) when the following methods are called: **com.rti.dds.domain.Domain↩ Participant.create_datareader** (p. 703), **com.rti.dds.domain.DomainParticipant.create_datareader_with_profile** (p. 704), or **com.rti.dds.domain.DomainParticipant.get_implicit_subscriber** (p. 744)

Parameters

topic	*<<**in**>> (p. 156)* The **com.rti.dds.topic.TopicDescription** (p. 1820) that the **com.rti.dds.subscription.DataReader** (p. 450) will be associated with. Cannot be NULL.
library_name	*<<**in**>> (p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	*<<**in**>> (p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).
listener	*<<**in**>> (p. 156)* The listener of the **com.rti.dds.subscription.DataReader** (p. 450).
mask	*<<**in**>> (p. 156).* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.subscription.DataReader** (p. 450) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_datareader (p. 703)

com.rti.dds.domain.DomainParticipant.get_default_datareader_qos (p. 686)

com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 744)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.subscription.DataReader.set_listener (p. 459)

8.78.2.31 delete_datareader()

```
void delete_datareader (
            DataReader a_datareader )
```

<<***extension***>> *(p. 155)* Deletes a **com.rti.dds.subscription.DataReader** (p. 450) that belongs to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

If the **com.rti.dds.subscription.DataReader** (p. 450) does not belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730), or if ther are any existing **com.rti.dds.subscription.ReadCondition** (p. 1514) or **com.rti.dds.subscription.QueryCondition** (p. 1510) objects that are attached to the **com.rti.dds.subscription.DataReader** (p. 450), or if there are outstanding loans on samples (as a result of a call to read(), take(), or one of the variants thereof), the operation fails with the error **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.subscription.DataReader** (p. 450) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datareader	<<***in***>> *(p. 156)* The **com.rti.dds.subscription.DataReader** (p. 450) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

See also

com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 744)

8.78.2.32 create_topic()

```
Topic create_topic (
            String topic_name,
```

```
String type_name,
 TopicQos qos,
 TopicListener listener,
int mask )
```

Creates a **com.rti.dds.topic.Topic** (p. 1807) with the desired QoS policies and attaches to it the specified **com.rti.dds.topic.TopicListener** (p. 1822).

Precondition

The application is not allowed to create two **com.rti.dds.topic.Topic** (p. 1807) objects with the same `topic_name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 670). If the application attempts this, this method will fail and return a NULL topic.

The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.topic.Topic** (p. 1807) will be created.

Prior to creating a **com.rti.dds.topic.Topic** (p. 1807), the type must have been registered with RTI Connext. This is done using the **com.rti.ndds.example.FooTypeSupport.register_type** (p. 1119) operation on a derived class of the **com.rti.dds.topic.TypeSupport** (p. 1941) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 722).

MT Safety:

UNSAFE. If **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) is used for `qos`, it is not safe to create the topic while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679).

Parameters

topic_name	<<***in***>> *(p. 156)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.
type_name	<<***in***>> *(p. 156)* The type to which the new **com.rti.dds.topic.Topic** (p. 1807) will be bound. Cannot be NULL.
qos	<<***in***>> *(p. 156)* QoS to be used for creating the new **com.rti.dds.topic.Topic** (p. 1807). The special value **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) can be used to indicate that the **com.rti.dds.topic.Topic** (p. 1807) should be created with the default **com.rti.dds.topic.TopicQos** (p. 1824) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.
listener	<<***in***>> *(p. 156)*. Listener to be attached to the newly created **com.rti.dds.topic.Topic** (p. 1807).
mask	<<***in***>> *(p. 156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.topic.TopicQos (p. 1824) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT (p. 45)

com.rti.dds.domain.DomainParticipant.create_topic_with_profile (p. 708)

com.rti.dds.domain.DomainParticipant.get_default_topic_qos (p. 679)

com.rti.dds.topic.Topic.set_listener (p. 1811)

8.78.2.33 create_topic_with_profile()

```
Topic create_topic_with_profile (
            String topic_name,
            String type_name,
            String library_name,
            String profile_name,
             TopicListener listener,
            int mask )
```

<<***extension***>> *(p. 155)* Creates a new **com.rti.dds.topic.Topic** (p. 1807) object using the **com.rti.dds.publication.PublisherQos** (p. 1490) associated with the input XML QoS profile.

Precondition

The application is not allowed to create two **com.rti.dds.topic.TopicDescription** (p. 1820) objects with the same `topic_name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 670). If the application attempts this, this method will fail and return a NULL topic.

The **com.rti.dds.topic.TopicQos** (p. 1824) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.topic.Topic** (p. 1807) will be created.

Prior to creating a **com.rti.dds.topic.Topic** (p. 1807), the type must have been registered with RTI Connext. This is done using the **com.rti.ndds.example.FooTypeSupport.register_type** (p. 1119) operation on a derived class of the **com.rti.dds.topic.TypeSupport** (p. 1941) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 722).

Parameters

topic_name	<<***in***>> *(p. 156)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.
type_name	<<***in***>> *(p. 156)* The type to which the new **com.rti.dds.topic.Topic** (p. 1807) will be bound. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default
	profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).
listener	<<***in***>> *(p. 156).* Listener to be attached to the newly created **com.rti.dds.topic.Topic** (p. 1807).
mask	<<***in***>> *(p. 156).* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.topic.TopicQos (p. 1824) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

com.rti.dds.domain.DomainParticipant.get_default_topic_qos (p. 679)

com.rti.dds.topic.Topic.set_listener (p. 1811)

8.78.2.34 delete_topic()

```
void delete_topic (
              Topic topic )
```

Deletes a **com.rti.dds.topic.Topic** (p. 1807).

Precondition

If the **com.rti.dds.topic.Topic** (p. 1807) does not belong to the application's **com.rti.dds.domain.Domain↩ Participant** (p. 670), this operation fails with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_↩ MET** (p. 1598).

Make sure no objects are using the topic. More specifically, there must be no existing **com.rti.dds.↩ subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.topic.Content↩ FilteredTopic** (p. 436), or **com.rti.dds.topic.MultiTopic** (p. 1320) objects belonging to the same **com.rti.dds.↩ domain.DomainParticipant** (p. 670) that are using the **com.rti.dds.topic.Topic** (p. 1807). If delete_topic is called on a **com.rti.dds.topic.Topic** (p. 1807) with any of these existing objects attached to it, it will fail with **com.rti.↩ dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.topic.Topic** (p. 1807) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

topic	<<***in***>> *(p. 156)* **com.rti.dds.topic.Topic** (p. 1807) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

8.78.2.35 create_contentfilteredtopic()

```
ContentFilteredTopic create_contentfilteredtopic (
            String name,
             Topic related_topic,
            String filter_expression,
             StringSeq expression_parameters )
```

Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), that can be used to do content-based subscriptions.

The **com.rti.dds.topic.ContentFilteredTopic** (p. 436) only relates to samples published under that **com.rti.dds.topic.Topic** (p. 1807), filtered according to their content. The filtering is done by means of evaluating a logical expression that involves the values of some of the data-fields in the sample. The logical expression derived from the `filter_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 104) describes the syntax of `filter_expression` and `expression_parameters`.

Precondition

The application is not allowed to create two **com.rti.dds.topic.ContentFilteredTopic** (p. 436) objects with the same `topic_name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 670). If the application attempts this, this method will fail and returns NULL.

If `related_topic` does not belong to this **com.rti.dds.domain.DomainParticipant** (p. 670), this operation returns NULL.

This function will create a content filter using the builtin SQL filter which implements a superset of the DDS specification. This filter **requires** that all IDL types have been compiled with typecodes. If this precondition is not met, this operation returns NULL.

Parameters

name	<<***in***>> *(p. 156)* Name for the new content filtered topic, must not exceed 255 characters. Cannot be NULL.
related_topic	<<***in***>> *(p. 156)* **com.rti.dds.topic.Topic** (p. 1807) to be filtered. Cannot be NULL.
filter_expression	<<***in***>> *(p. 156)* Cannot be NULL
expression_parameters	<<***in***>> *(p. 156)* Cannot be NULL. An empty sequence **must** be used if the filter expression does not contain any parameters. Length of sequence cannot be greater than 100.

Returns

newly created **com.rti.dds.topic.ContentFilteredTopic** (p. 436), or NULL on failure

8.78.2.36 create_contentfilteredtopic_with_filter()

```
ContentFilteredTopic create_contentfilteredtopic_with_filter (
            String name,
             Topic related_topic,
            String filter_expression,
             StringSeq expression_parameters,
            String filter_name )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) using the specified filter to do content-based subscriptions.

Parameters

name	<<***in***>> *(p. 156)* Name for the new content filtered topic. Cannot exceed 255 characters. Cannot be NULL.
related_topic	<<***in***>> *(p. 156)* **com.rti.dds.topic.Topic** (p. 1807) to be filtered. Cannot be NULL.
filter_expression	<<***in***>> *(p. 156)* Cannot be NULL.
expression_parameters	<<***in***>> *(p. 156)* Cannot be NULL. . An empty sequence **must** be used if the filter expression does not contain any parameters. Length of the sequence cannot be greater than 100.
filter_name	<<***in***>> *(p. 156)* Name of content filter to use. Must previously have been registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740) on the same **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL. Builtin filter names are **com.rti.dds.domain.DomainParticipant.SQLFILTER_NAME** (p. 50) and **com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50)

Returns

newly created **com.rti.dds.topic.ContentFilteredTopic** (p. 436), or NULL on failure

8.78.2.37 delete_contentfilteredtopic()

```
void delete_contentfilteredtopic (
             ContentFilteredTopic a_contentfilteredtopic )
```

Deletes a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

Precondition

The deletion of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) is not allowed if there are any existing **com.↩ rti.dds.subscription.DataReader** (p. 450) objects that are using the **com.rti.dds.topic.ContentFilteredTopic** (p. 436). If the operation is called on a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) with existing **com.↩ rti.dds.subscription.DataReader** (p. 450) objects attached to it, it will fail with **com.rti.dds.infrastructure.↩ RETCODE_PRECONDITION_NOT_MET** (p. 1598).

The **com.rti.dds.topic.ContentFilteredTopic** (p. 436) must be created by this **com.rti.dds.domain.↩ DomainParticipant** (p. 670), or else this operation will fail with **com.rti.dds.infrastructure.RETCODE_↩ PRECONDITION_NOT_MET** (p. 1598).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_contentfilteredtopic	*<<**in**>> (p. 156)*

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

8.78.2.38 create_multitopic()

```
MultiTopic create_multitopic (
            String name,
            String type_name,
            String subscription_expression,
             StringSeq expression_parameters )
```

[Not supported (optional)] Creates a MultiTopic that can be used to subscribe to multiple topics and combine/filter the received data into a resulting type.

The resulting type is specified by the `type_name` argument. The list of topics and the logic used to combine, filter, and rearrange the information from each **com.rti.dds.topic.Topic** (p. 1807) are specified using the `subscription↩ _expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 104) describes the syntax of `subscription_expression` and `expression↩ _parameters`.

Precondition

The application is not allowed to create two **com.rti.dds.topic.TopicDescription** (p. 1820) objects with the same `name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 670). If the application attempts this, this method will fail and return NULL.

Prior to creating a **com.rti.dds.topic.MultiTopic** (p. 1320), the type must have been registered with RTI Connext. This is done using the **com.rti.ndds.example.FooTypeSupport.register_type** (p. 1119) operation on a derived class of the **com.rti.dds.topic.TypeSupport** (p. 1941) interface. Otherwise, this method will return NULL.

Parameters

name	<<***in***>> *(p. 156)* Name of the newly create **com.rti.dds.topic.MultiTopic** (p. 1320). Cannot be NULL.
type_name	<<***in***>> *(p. 156)* Cannot be NULL.
subscription_expression	<<***in***>> *(p. 156)* Cannot be NULL.
expression_parameters	<<***in***>> *(p. 156)* Cannot be NULL.

Returns

NULL

8.78.2.39 delete_multitopic()

```
void delete_multitopic (
              MultiTopic a_multitopic )
```

[Not supported (optional)] Deletes a **com.rti.dds.topic.MultiTopic** (p. 1320).

Precondition

The deletion of a **com.rti.dds.topic.MultiTopic** (p. 1320) is not allowed if there are any existing **com.rti.dds.↩ subscription.DataReader** (p. 450) objects that are using the **com.rti.dds.topic.MultiTopic** (p. 1320). If the delete_multitopic operation is called on a **com.rti.dds.topic.MultiTopic** (p. 1320) with existing **com.rti.dds.↩ subscription.DataReader** (p. 450) objects attached to it, it will fail with **com.rti.dds.infrastructure.RETCODE↩ _PRECONDITION_NOT_MET** (p. 1598).

The **com.rti.dds.topic.MultiTopic** (p. 1320) must be created by this **com.rti.dds.domain.DomainParticipant** (p. 670), or else this operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_multitopic	<<***in***>> *(p. 156)*

8.78.2.40 set_qos()

```
void set_qos (
            DomainParticipantQos qos )
```

Change the QoS of this **DomainParticipant** (p. 670).

The **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 799) and **com.rti.dds.domain.DomainParticipant↩ Qos.entity_factory** (p. 799) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(p. 156)* Set of policies to be applied to **com.rti.dds.domain.DomainParticipant** (p. 670). Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled. The special value **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42) can be used to indicate that the QoS of the **com.rti.dds.domain.DomainParticipant** (p. 670) should be changed to match the current default **com.rti.dds.domain.DomainParticipantQos** (p. 795) set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 761). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) if an immutable policy is changed, or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) if policies are inconsistent

See also

com.rti.dds.domain.DomainParticipantQos (p. 795) for rules on consistency among QoS policies

set_qos (abstract) (p. 1030)

8.78.2.41 set_qos_with_profile()

```
void set_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Change the QoS of this domain participant using the input XML QoS profile.

The **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 799) and **com.rti.dds.domain.DomainParticipant↩ Qos.entity_factory** (p. 799) can be changed. The other policies are immutable.

Parameters

library_name	*<<**in**>> (p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	*<<**in**>> (p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) if immutable policy is changed, or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) if policies are inconsistent

See also

com.rti.dds.domain.DomainParticipantQos (p. 795) for rules on consistency among QoS

8.78.2.42 get_qos()

```
void get_qos (
              DomainParticipantQos qos )
```

Get the participant QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	*<<**inout**>> (p. 156)* QoS to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.78.2.43 get_default_library()

```
String get_default_library ( )
```

<<***extension***>> *(p. 155)* Gets the default XML library associated with a **com.rti.dds.domain.DomainParticipant** (p. 670).

Returns

The default library or null if the default library was not set.

See also

com.rti.dds.domain.DomainParticipant.set_default_library (p. 716)

8.78.2.44 set_default_library()

```
void set_default_library (
             String library_name )
```

<<***extension***>> *(p. 155)* Sets the default XML library for a **com.rti.dds.domain.DomainParticipant** (p. 670).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this **DomainParticipant** (p. 670)'s operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **com.rti.dds.domain.DomainParticipant** (p. 670) inherits the default from the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).

Parameters

library_name	<<***in***>> *(p. 156)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.get_default_library (p. 715)

8.78.2.45 get_default_profile()

```
String get_default_profile ( )
```

<<***extension***>> *(p. 155)* Gets the default XML profile associated with a **com.rti.dds.domain.DomainParticipant** (p. 670).

Returns

The default profile or null if the default profile was not set.

See also

com.rti.dds.domain.DomainParticipant.set_default_profile (p. 717)

8.78.2.46 set_default_profile()

```
void set_default_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Sets the default XML profile for a **com.rti.dds.domain.DomainParticipant** (p. 670).

This method specifies the profile that will be used as the default the next time a default **DomainParticipant** (p. 670) profile is needed during a call to one of this **DomainParticipant** (p. 670)'s operations. When calling a **com.rti.dds.↩ domain.DomainParticipant** (p. 670) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **com.rti.dds.domain.DomainParticipant** (p. 670) inherits the default from the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) (see **com.rti.dds.domain.DomainParticipantFactory.set↩ _default_profile** (p. 774)).

This method does not set the default QoS for entities created by the **com.rti.dds.domain.DomainParticipant** (p. 670); for this functionality, use the methods set_default_<entity>_qos_with_profile (you may pass in NULL after having called **set_default_profile()** (p. 717)).

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **com.rti.↩ dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 768).

Parameters

library_name	<<***in***>> *(p. 156)* The library name containing the profile.
profile_name	<<***in***>> *(p. 156)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.get_default_profile (p. 716)

com.rti.dds.domain.DomainParticipant.get_default_profile_library (p. 718)

8.78.2.47 get_default_profile_library()

```
String get_default_profile_library ( )
```

<<***extension***>> *(p. 155)* Gets the library where the default XML QoS profile is contained for a **com.rti.dds.domain.DomainParticipant** (p. 670).

The default profile library is automatically set when **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717) is called.

This library can be different than the **com.rti.dds.domain.DomainParticipant** (p. 670) default library (see **com.rti.dds.domain.DomainParticipant.get_default_library** (p. 715)).

Returns

The default profile library or null if the default profile was not set.

See also

com.rti.dds.domain.DomainParticipant.set_default_profile (p. 717)

8.78.2.48 set_listener()

```
void set_listener (
            DomainParticipantListener l,
            int mask )
```

Sets the participant listener.

Parameters

l	<<***in***>> *(p. 156)* Listener to be installed on the entity.
mask	<<***in***>> *(p. 156)* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

MT Safety:

Unsafe. This method is not synchronized with the listener callbacks, so it is possible to set a new listener on a participant when the old listener is in a callback.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.78.2.49 get_listener()

```
DomainParticipantListener get_listener ( )
```

Get the participant listener.

Returns

Existing listener attached to the **com.rti.dds.domain.DomainParticipant** (p. 670).

See also

get_listener (abstract) (p. 1032)

8.78.2.50 get_publishers()

```
void get_publishers (
            PublisherSeq publishers )
```

<<***extension***>> *(p. 155)* Allows the application to access all the publishers the participant has.

If the sequence doesn't own its buffer, and its maximum is less than the total number of publishers, it will be filled up to its maximum, and fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

Safe.

Parameters

publishers	<<***inout***>> *(*p. *156)* a PublisherSeq object where the set or list of publishers will be returned

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

8.78.2.51 get_subscribers()

```
void get_subscribers (
              SubscriberSeq subscribers )
```

<<***extension***>> *(*p. *155)* Allows the application to access all the subscribers the participant has.

If the sequence doesn't own its buffer, and its maximum is less than the total number of subscribers, it will be filled up to its maximum, and fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

Safe.

Parameters

subscribers	<<***inout***>> *(*p. *156)* a SubscriberSeq object where the set or list of subscribers will be returned

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

8.78.2.52 get_builtin_subscriber()

```
Subscriber get_builtin_subscriber ( )
```

Accesses the **built-in com.rti.dds.subscription.Subscriber** (p. 1730).

Each **com.rti.dds.domain.DomainParticipant** (p. 670) contains several built-in **com.rti.dds.topic.Topic** (p. 1807) objects as well as corresponding **com.rti.dds.subscription.DataReader** (p. 450) objects to access them. All of

these **com.rti.dds.subscription.DataReader** (p. 450) objects belong to a single built-in **com.rti.dds.subscription.Subscriber** (p. 1730).

The built-in Topics are used to communicate information about other **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), and **com.rti.dds.publication.DataWriter** (p. 553) objects.

The built-in subscriber is created when this operation is called for the first time. The built-in subscriber is deleted automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is deleted.

Returns

The built-in **com.rti.dds.subscription.Subscriber** (p. 1730) singleton.

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

builtin.TopicBuiltinTopicData

8.78.2.53 lookup_flowcontroller()

```
FlowController lookup_flowcontroller (
            String name )
```

<<***extension***>> *(p. 155)* Looks up an existing locally-created **com.rti.dds.publication.FlowController** (p. 1055), based on its name.

Looks up a previously created **com.rti.dds.publication.FlowController** (p. 1055), including the built-in ones. Once a **com.rti.dds.publication.FlowController** (p. 1055) has been deleted, subsequent lookups will fail.

MT Safety:

UNSAFE. It is not safe to lookup a flow controller description while another thread is creating that flow controller.

Parameters

name	<<***in***>> *(p. 156)* Name of **com.rti.dds.publication.FlowController** (p. 1055) to search for. Limited to 255 characters. Cannot be NULL.

Returns

The flow controller if it has already been created locally, or NULL otherwise.

8.78.2.54 find_topic()

```
Topic find_topic (
            String topic_name,
             Duration_t timeout )
```

Finds an existing (or ready to exist) **com.rti.dds.topic.Topic** (p. 1807), based on its name.

This call can be used to block for a specified duration to wait for the **com.rti.dds.topic.Topic** (p. 1807) to be created.

If the requested **com.rti.dds.topic.Topic** (p. 1807) already exists, it is returned. Otherwise, **find_topic()** (p. 721) waits until another thread creates it or else returns when the specified timeout occurs.

find_topic() (p. 721) is useful when multiple threads are concurrently creating and looking up topics. In that case, one thread can call **find_topic()** (p. 721) and, if another thread has not yet created the topic being looked up, it can wait for some period of time for it to do so. In almost all other cases, it is more straightforward to call **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 722).

The **com.rti.dds.domain.DomainParticipant** (p. 670) must already be enabled.

Note: *Each* **com.rti.dds.topic.Topic** (p. 1807) obtained by **com.rti.dds.domain.DomainParticipant.find_topic** (p. 721) must also be deleted by means of **com.rti.dds.domain.DomainParticipant.delete_topic** (p. 709). If **com.rti.dds.topic.Topic** (p. 1807) is obtained multiple times by means of **com.rti.dds.domain.DomainParticipant.find_topic** (p. 721) or **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706), it must also be deleted that same number of times using **com.rti.dds.domain.DomainParticipant.delete_topic** (p. 709).

Parameters

topic_name	<<***in***>> *(p. 156)* Name of the **com.rti.dds.topic.Topic** (p. 1807) to search for. Cannot be NULL.
timeout	<<***in***>> *(p. 156)* The time to wait if the **com.rti.dds.topic.Topic** (p. 1807) does not exist already. Cannot be NULL.

Returns

the topic, if it exists, or NULL

8.78.2.55 lookup_topicdescription()

```
TopicDescription lookup_topicdescription (
            String topic_name )
```

Looks up an existing, locally created **com.rti.dds.topic.TopicDescription** (p. 1820), based on its name.

com.rti.dds.topic.TopicDescription (p. 1820) is the base class for **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.MultiTopic** (p. 1320) and **com.rti.dds.topic.ContentFilteredTopic** (p. 436). So you can narrow the **com.rti.dds.topic.TopicDescription** (p. 1820) returned from this operation to a **com.rti.dds.topic.Topic** (p. 1807) or **com.rti.dds.topic.ContentFilteredTopic** (p. 436) as appropriate.

Unlike **com.rti.dds.domain.DomainParticipant.find_topic** (p. 721), which logically returns a new **com.rti.dds.topic.Topic** (p. 1807) object that must be independently deleted, *this* operation returns a reference to the original local object.

The **com.rti.dds.domain.DomainParticipant** (p. 670) does not have to be enabled when you call **lookup_topicdescription()** (p. 722).

The returned topic may be either enabled or disabled.

MT Safety:

UNSAFE. It is not safe to lookup a topic description while another thread is creating that topic.

Parameters

topic_name	*<<**in**>> (p. 156)* Name of **com.rti.dds.topic.TopicDescription** (p. 1820) to search for. This string must be no more than 255 characters; it cannot be NULL.

Returns

The topic description, if it has already been created locally, otherwise it returns NULL.

8.78.2.56 ignore_participant()

```
void ignore_participant (
               InstanceHandle_t handle )
```

Instructs RTI Connext to locally ignore a remote **com.rti.dds.domain.DomainParticipant** (p. 670).

From the time of this call onwards, RTI Connext will locally behave as if the remote participant did not exist. This means it will ignore any topic, publication, or subscription that originates on that **com.rti.dds.domain.DomainParticipant** (p. 670).

There is no way to reverse this operation.

This operation can be used in conjunction with the discovery of remote participants offered by means of the **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) to provide access control.

Application data can be associated with a **com.rti.dds.domain.DomainParticipant** (p. 670) by means of the **USER_DATA** (p. 278) policy. This application data is propagated as a field in the built-in topic and can be used by an application to implement its own access control policy.

The **com.rti.dds.domain.DomainParticipant** (p. 670) to ignore is identified by the handle argument. This `handle` is the one that appears in the **com.rti.dds.subscription.SampleInfo** (p. 1634) retrieved when reading the data-samples available for the built-in **com.rti.dds.subscription.DataReader** (p. 450) to the **com.rti.dds.domain.DomainParticipant** (p. 670) topic. The built-in **com.rti.dds.subscription.DataReader** (p. 450) is read with the same **com.rti.ndds.example.FooDataReader.read** (p. 1069) and **com.rti.ndds.example.FooDataReader.take** (p. 1071) operations used for any **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

handle	<<***in***>> *(p. 156)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of the **com.rti.dds.domain.DomainParticipant** (p. 670) to be ignored. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

com.rti.dds.domain.builtin.ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 162)

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.78.2.57 banish_ignored_participants()

```
void banish_ignored_participants ( )
```

<<***extension***>> *(p. 155)* Prevents ignored remote DomainParticipants from receiving traffic from the local **com.rti.dds.domain.DomainParticipant** (p. 670).

This method complements **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723): ignore_participant prevents the local **com.rti.dds.domain.DomainParticipant** (p. 670) from processing traffic from the remote **DomainParticipant** (p. 670), while this method prevents already ignored remote DomainParticipants from processing traffic from the local **DomainParticipant** (p. 670).

Note: this method is currently only supported when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.domain.DomainParticipant.ignore_participant (p. 723)

8.78.2.58 ignore_topic()

```
void ignore_topic (
              InstanceHandle_t handle )
```

Instructs RTI Connext to locally ignore a **com.rti.dds.topic.Topic** (p. 1807).

This means it will locally ignore any publication, or subscription to the **com.rti.dds.topic.Topic** (p. 1807).

There is no way to reverse this operation.

This operation can be used to save local resources when the application knows that it will never publish or subscribe to data under certain topics.

The **com.rti.dds.topic.Topic** (p. 1807) to ignore is identified by the `handle` argument. This is the handle of a **com.rti.dds.topic.Topic** (p. 1807) that appears in the **com.rti.dds.subscription.SampleInfo** (p. 1634) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 450) for the **com.rti.dds.topic.Topic** (p. 1807).

Parameters

handle	<<***in***>> *(p. 156)* Handle of the **com.rti.dds.topic.Topic** (p. 1807) to be ignored. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

builtin.TopicBuiltinTopicData

com.rti.dds.topic.builtin.TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME (p. 165)

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.78.2.59 ignore_publication()

```
void ignore_publication (
              InstanceHandle_t handle )
```

Instructs RTI Connext to locally ignore a publication.

A publication is defined by the association of a topic name, user data, and partition set on the **com.rti.dds.↩ publication.Publisher** (p. 1466) (see **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452)). After this call, any data written by that publication's **com.rti.dds.publication.DataWriter** (p. 553) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The publication (DataWriter) to ignore is identified by the `handle` argument.

- To ignore a *remote* DataWriter, the `handle` can be obtained from the **com.rti.dds.subscription.SampleInfo** (p. 1634) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 450) for the **com.rti.dds.publication** (p. 310) topic.

- To ignore a *local* DataWriter, the `handle` can be obtained by calling **com.rti.dds.infrastructure.Entity.get_↩ instance_handle** (p. 1034) for the local DataWriter.

There is no way to reverse this operation.

Parameters

handle	<<***in***>> *(*p. *156)* Handle of the **com.rti.dds.publication.DataWriter** (p. 553) to be ignored. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

com.rti.dds.publication.builtin.PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME (p. 163)

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.78.2.60 ignore_subscription()

```
void ignore_subscription (
            InstanceHandle_t handle )
```

Instructs RTI Connext to locally ignore a subscription.

A subscription is defined by the association of a topic name, user data, and partition set on the **com.rti.dds.↩ subscription.Subscriber** (p. 1730) (see **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762)). After this call, any data received related to that subscription's **com.rti.dds.subscription.DataReader** (p. 450) will be ignored.

This operation can be used to ignore local *and* remote DataReaders.

The subscription to ignore is identified by the `handle` argument.

- To ignore a *remote* DataReader, the `handle` can be obtained from the **com.rti.dds.subscription.SampleInfo** (p. 1634) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 450) for the **com.rti.dds.subscription** (p. 312) topic.

- To ignore a *local* DataReader, the `handle` can be obtained by calling **com.rti.dds.infrastructure.Entity.get_↩ instance_handle** (p. 1034) for the local DataReader.

There is no way to reverse this operation.

Parameters

handle	<<***in***>> *(p. 156)* Handle of the **com.rti.dds.subscription.DataReader** (p. 450) to be ignored. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME (p. 164)

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.78.2.61 get_domain_id()

```
int get_domain_id ( )
```

Get the unique domain identifier.

This operation retrieves the domain id used to create the **com.rti.dds.domain.DomainParticipant** (p. 670). The domain id identifies the DDS domain to which the **com.rti.dds.domain.DomainParticipant** (p. 670) belongs. Each DDS domain represents a separate data 'communication plane' isolated from other domains.

Returns

the unique `domainId` that was used to create the domain

See also

com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 765)

com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile (p. 782)

8.78.2.62 assert_liveliness()

```
void assert_liveliness ( )
```

Manually asserts the liveliness of this **com.rti.dds.domain.DomainParticipant** (p. 670).

This is used in combination with the **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243) to indicate to RTI Connext that the entity remains active.

You need to use this operation if the **com.rti.dds.domain.DomainParticipant** (p. 670) contains **com.rti.dds.publication.DataWriter** (p. 553) entities with the **com.rti.dds.infrastructure.LivelinessQosPolicy.kind** (p. 1246) set to com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS and it only affects the liveliness of those **com.rti.dds.publication.DataWriter** (p. 553) entities. Otherwise, it has no effect.

Note: writing data via the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) or **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 553) itself and its **com.rti.dds.domain.DomainParticipant** (p. 670). Consequently the use of **assert_liveliness()** (p. 727) is only needed if the application is not writing data regularly.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1243)

8.78.2.63 delete_contained_entities()

```
void delete_contained_entities ( )
```

Delete all the entities that were created by means of the "create" operations on the **com.rti.dds.domain.DomainParticipant** (p. 670).

This operation deletes all contained **com.rti.dds.publication.Publisher** (p. 1466) (including an implicit Publisher, if one exists), **com.rti.dds.subscription.Subscriber** (p. 1730) (including implicit Subscriber), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320) objects.

Prior to deleting each contained entity, this operation will recursively call the corresponding **delete_contained_entities()** (p. 728) operation on each contained entity (if applicable). This pattern is applied recursively. In this manner the operation **delete_contained_entities()** (p. 728) on the **com.rti.dds.domain.DomainParticipant** (p. 670) will end up recursively deleting all the entities contained in the **com.rti.dds.domain.DomainParticipant** (p. 670), including the **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.DataReader** (p. 450), as well as the **com.rti.dds.subscription.QueryCondition** (p. 1510), **com.rti.dds.subscription.ReadCondition** (p. 1514), and **com.rti.dds.subscription.TopicQuery** (p. 1830) objects belonging to the contained **com.rti.dds.subscription.DataReader** (p. 450).

The operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if any of the contained entities is in a state where it cannot be deleted .

If **delete_contained_entities()** (p. 728) completes successfully, the application may delete the **com.rti.dds.domain.DomainParticipant** (p. 670).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.78.2.64 add_peer()

```
void add_peer (
            String peer_desc_string )
```

<<***extension***>> *(*p. *155)* Attempt to contact one or more additional peer participants.

Add the given peer description to the list of peers with which this **com.rti.dds.domain.DomainParticipant** (p. 670) will try to communicate.

This method may be called at any time after this **com.rti.dds.domain.DomainParticipant** (p. 670) has been created (before or after it has been enabled).

If this method is called after **com.rti.dds.infrastructure.Entity.enable** (p. 1032), an attempt will be made to contact the new peer(s) immediately.

If this method is called *before* the **DomainParticipant** (p. 670) is enabled, the peer description will simply be added to the list that was populated by **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668); the first attempted contact will take place after this **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.

Adding a peer description with this method does not guarantee that any peer(s) discovered as a result will exactly correspond to those described:

- This **com.rti.dds.domain.DomainParticipant** (p. 670) will attempt to discover peer participants at the given locations but may not succeed if no such participants are available. In this case, this method will not wait for contact attempt(s) to be made and it will not report an error.
- If remote participants described by the given peer description *are* discovered, the distributed application is configured with asymmetric peer lists, and **com.rti.dds.infrastructure.DiscoveryQosPolicy.accept_unknown_peers** (p. 668) is set to com.rti.dds.infrastructure.true. Thus, this **com.rti.dds.domain.DomainParticipant** (p. 670) may actually discover *more* peers than are described in the given peer description.

To be informed of the exact remote participants that are discovered, regardless of which peers this **com.rti.dds.↩ domain.DomainParticipant** (p. 670) *attempts* to discover, use the built-in participant topic: **com.rti.dds.domain.↩ builtin.ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME** (p. 162).

To remove specific peer locators, you may use **com.rti.dds.domain.DomainParticipant.remove_peer** (p. 730). If a peer is removed, the add_peer operation will add it back to the list of peers.

To stop communicating with a peer **com.rti.dds.domain.DomainParticipant** (p. 670) that has been discovered, use **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723).

Adding a peer description with this method has no effect on the **com.rti.dds.infrastructure.DiscoveryQosPolicy.↩ initial_peers** (p. 668) that may be subsequently retrieved with **com.rti.dds.domain.DomainParticipant.get_qos()** (p. 715) (because **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 665) is immutable).

Parameters

peer_desc_string	<<***in***>> *(p. 156)* New peer descriptor to be added. The format is specified in **Peer Descriptor Format** (p. 223). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

Peer Descriptor Format (p. 223)

com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 668)

com.rti.dds.domain.builtin.ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 162)

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.78.2.65 remove_peer()

```
void remove_peer (
             String peer_desc_string )
```

<<***extension***>> *(p. 155)* Remove one or more peer participants from the list of peers with which this **com.rti.dds.domain.DomainParticipant** (p. 670) will try to communicate.

This method may be called any time after this **com.rti.dds.domain.DomainParticipant** (p. 670) has been enabled

Calling this method has the following effects:

- If a **com.rti.dds.domain.DomainParticipant** (p. 670) was already discovered, it will be locally removed along with all its entities.
- Any further requests coming from a **com.rti.dds.domain.DomainParticipant** (p. 670) located on any of the removed peers will be ignored.
- All the locators contained in the peer description will be removed from the peer list. The local **com.rti.dds.domain.DomainParticipant** (p. 670) will stop sending announcement to those locators.

If remote participants located on a peer that was previously removed are discovered, they will be ignored untill the related peer is added back by using **com.rti.dds.domain.DomainParticipant.add_peer** (p. 729).

Removing a peer description with this method has no effect on the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) that may be subsequently retrieved with **com.rti.dds.domain.DomainParticipant.get_qos()** (p. 715) (because **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 665) is immutable).

Parameters

peer_desc_string	<<***in***>> *(p. 156)* Peer descriptor to be removed. The format is specified in **Peer Descriptor Format** (p. 223). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

Peer Descriptor Format (p. 223)

com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 668)

com.rti.dds.domain.DomainParticipant.add_peer (p. 729)

8.78.2.66 get_dns_tracker_polling_period()

```
void get_dns_tracker_polling_period (
              Duration_t polling_period )
```

<<***extension***>> *(p. 155)* Retrieves the frequency used by the DNS tracker thread to query the DNS service.

The DNS tracker queries the DNS for hostnames specified in the initial peers of a **DomainParticipant** (p. 670). The frequency of these queries is defined by **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.dns_tracker_polling_period** (p. 663). If the value returned is **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), the DNS tracker is disabled.

Parameters

polling_period	<<***out***>> *(p. 156)* Duration that the API populates with the period of the DNS tracker.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.dns_tracker_polling_period (p. 663)

8.78.2.67 set_dns_tracker_polling_period()

```
void set_dns_tracker_polling_period (
            Duration_t polling_period )
```

<<***extension***>> *(*p. *155)* Configures the frequency in which the DNS tracker queries the DNS service.

This API allows you to change the frequency of the polling period for the DNS tracker. The range of accepted values, in seconds, goes from 1 second to 1 year. **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) is also accepted as a valid value. If the duration is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), the DNS tracker is disabled.

Modifying the DNS tracker polling period through this has no effect on the **com.rti.dds.infrastructure.Discovery↩ ConfigQosPolicy.dns_tracker_polling_period** (p. 663) when it is retrieved with **com.rti.dds.domain.Domain↩ Participant.get_qos()** (p. 715).

Parameters

polling_period	<<***in***>> *(*p. *156)* Duration that is set as the polling period for the DNS tracker.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.dns_tracker_polling_period (p. 663)

com.rti.dds.domain.DomainParticipant.add_peer (p. 729)

8.78.2.68 get_current_time()

```
void get_current_time (
            Time_t current_time )
```

Returns the current value of the time.

The current value of the time that RTI Connext uses to time-stamp **com.rti.dds.publication.DataWriter** (p. 553) and to set the reception-timestamp for the data updates that it receives.

Parameters

current_time	<<***inout***>> *(*p. *156)* Current time to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.78.2.69 get_discovered_participants()

```
void get_discovered_participants (
              InstanceHandleSeq participant_handles )
```

Returns a list of discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities.

This operation retrieves the list of **com.rti.dds.domain.DomainParticipant** (p. 670) entities that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723) operation. When using **com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind.SPDP2** (p. 645), this list only includes **com.rti.dds.domain.DomainParticipant** (p. 670) entities that the application has received configuration information from.

Parameters

participant_handles	<<***inout***>> *(p. 156)* **com.rti.dds.infrastructure.InstanceHandleSeq** (p. 1156) to be filled with handles of the discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.78.2.70 get_discovered_participants_from_subject_name()

```
void get_discovered_participants_from_subject_name (
              InstanceHandleSeq participant_handles,
             String subject_name )
```

<<***extension***>> *(p. 155)* Returns a list of discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities that have the given **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038).

This operation retrieves the same list as **com.rti.dds.domain.DomainParticipant.get_discovered_participants** (p. 733), except this list contains only the participants that have the given **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038).

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant_handles	*<<**inout**>> (p. 156)* **com.rti.dds.infrastructure.InstanceHandleSeq** (p. 1156) to be filled with handles of the discovered **com.rti.dds.domain.DomainParticipant** (p. 670) entities.
subject_name	*<<**in**>> (p. 156)* The **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038) by which to filter the list.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.78.2.71 get_discovered_participant_data()

```
void get_discovered_participant_data (
              ParticipantBuiltinTopicData participant_data,
              InstanceHandle_t participant_handle )
```

Returns **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) for the specified **com.rti.dds.domain.↩ DomainParticipant** (p. 670).

This operation retrieves information on a **com.rti.dds.domain.DomainParticipant** (p. 670) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 670). Otherwise, the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Use the operation **com.rti.dds.domain.DomainParticipant.get_discovered_participants** (p. 733) to find the **com.↩ rti.dds.domain.DomainParticipant** (p. 670) entities that are currently discovered.

MT Safety:

Safe.

Parameters

participant_data	*<<**inout**>> (p. 156)* **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) to be filled in with the data for the specified **com.rti.dds.domain.DomainParticipant** (p. 670).
participant_handle	*<<**in**>> (p. 156)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of **com.rti.dds.domain.DomainParticipant** (p. 670).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

com.rti.dds.domain.DomainParticipant.get_discovered_participants (p. 733)

8.78.2.72 get_discovered_participant_subject_name()

```
String get_discovered_participant_subject_name (
            InstanceHandle_t participant_handle )
```

<<***extension***>> *(p. 155)* Returns **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038) for the specified **com.rti.dds.domain.DomainParticipant** (p. 670).

This operation retrieves the **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038) of a **com.rti.dds.domain.DomainParticipant** (p. 670) that has been discovered on the network. The participant must be in the same domain as the participant on which this operation is invoked and must not have been "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 670). If the `participant_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) or is not a valid **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) for a **DomainParticipant** (p. 670), then the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If the `participant_handle` corresponds to a **DomainParticipant** (p. 670) that has not been discovered, then the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Use the operation **com.rti.dds.domain.DomainParticipant.get_discovered_participants** (p. 733) to find the **com.rti.dds.domain.DomainParticipant** (p. 670) entities that are currently discovered.

Note: this method has different functionality when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

MT Safety:

Safe.

Parameters

participant_handle	<<***in***>> *(p. 156)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of **com.rti.dds.domain.DomainParticipant** (p. 670).

Returns

The **com.rti.dds.infrastructure.EntityNameQosPolicy.name** (p. 1038) for the `participant_handle`

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.infrastructure.EntityNameQosPolicy.name (p. 1038)

com.rti.dds.domain.DomainParticipant.get_discovered_participants (p. 733)

8.78.2.73 get_discovered_topics()

```
void get_discovered_topics (
            InstanceHandleSeq topic_handles )
```

Returns list of discovered **com.rti.dds.topic.Topic** (p. 1807) objects.

This operation retrieves the list of **com.rti.dds.topic.Topic** (p. 1807) s that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.↩ ignore_topic** (p. 724) operation.

Parameters

topic_handles	*<<**inout**>> (p. 156)* **com.rti.dds.infrastructure.InstanceHandleSeq** (p. 1156) to be filled with handles of the discovered **com.rti.dds.topic.Topic** (p. 1807) objects

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

8.78.2.74 get_discovered_topic_data()

```
void get_discovered_topic_data (
            TopicBuiltinTopicData topic_data,
            InstanceHandle_t topic_handle )
```

Returns builtin.TopicBuiltinTopicData for the specified **com.rti.dds.topic.Topic** (p. 1807).

This operation retrieves information on a **com.rti.dds.topic.Topic** (p. 1807) that has been discovered by the local Participant and must not have been "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 724) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_UNSUPPORTED** (p. 1599).

Use the operation **com.rti.dds.domain.DomainParticipant.get_discovered_topics** (p. 736) to find the topics that are currently discovered.

Parameters

topic_data	<<***inout***>> *(*p. *156)* builtin.TopicBuiltinTopicData to be filled with the specified **com.rti.dds.topic.Topic** (p. 1807)'s data.
topic_handle	<<***in***>> *(*p. *156)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of **com.rti.dds.topic.Topic** (p. 1807).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

builtin.TopicBuiltinTopicData

com.rti.dds.domain.DomainParticipant.get_discovered_topics (p. 736)

8.78.2.75 contains_entity()

```
boolean contains_entity (
            InstanceHandle_t a_handle )
```

Completes successfully with com.rti.dds.infrastructure.true if the referenced **com.rti.dds.infrastructure.Entity** (p. 1029) is contained by the **com.rti.dds.domain.DomainParticipant** (p. 670).

This operation checks whether or not the given `a_handle` represents an **com.rti.dds.infrastructure.Entity** (p. 1029) that was created from the **com.rti.dds.domain.DomainParticipant** (p. 670). The containment applies recursively. That is, it applies both to entities (**com.rti.dds.topic.TopicDescription** (p. 1820), **com.rti.dds.publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730)) created directly using the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) as well as entities created using a contained **com.rti.dds.publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) as the factory, and so forth.

The `instance` handle for an **com.rti.dds.infrastructure.Entity** (p. 1029) may be obtained from built-in topic data, from various statuses, or from the operation **com.rti.dds.infrastructure.Entity.get_instance_handle** (p. 1034).

Parameters

a_handle	<<***in***>> *(p. 156)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) of the **com.rti.dds.infrastructure.Entity** (p. 1029) to be checked.

Returns

com.rti.dds.infrastructure.true if **com.rti.dds.infrastructure.Entity** (p. 1029) is contained by the **com.rti.dds.↩ domain.DomainParticipant** (p. 670), or com.rti.dds.infrastructure.false otherwise.

8.78.2.76 register_durable_subscription()

```
void register_durable_subscription (
              EndpointGroup_t group,
             String topic_name )
```

<<***extension***>> *(p. 155)* Registers a Durable Subscription on the specified **com.rti.dds.topic.Topic** (p. 1807) on all Persistence Services.

If you need to receive all samples published on a **com.rti.dds.topic.Topic** (p. 1807), including the ones published while a **com.rti.dds.subscription.DataReader** (p. 450) is inactive or before it may be created, create a Durable Subscription using this method.

In this way, the Persistence Service will ensure that all the samples on that **com.rti.dds.topic.Topic** (p. 1807) are retained until they are acknowledged by at least *N* DataReaders belonging to the Durable Subscription where *N* is the quorum count.

If the same Durable Subscription is created on a different **com.rti.dds.topic.Topic** (p. 1807), the Persistence Service will implicitly delete the previous Durable Subscription and create a new one on the new **com.rti.dds.topic.Topic** (p. 1807).

Parameters

group	<<***in***>> *(p. 156)* **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022) The Durable Subscription name and quorum.
topic_name	<<***in***>> *(p. 156)* The topic name for which the Durable Subscription is created.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.78.2.77 delete_durable_subscription()

```
void delete_durable_subscription (
```

```
            EndpointGroup_t group )
```

<<***extension***>> *(*p. *155)* Deletes an existing Durable Subscription on all Persistence Services.

The Persistence Service will delete the Durable Subscription and the quorum of the existing samples will be considered satisfied.

Parameters

group	<<***in***>> *(*p. *156)* **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022) specifying the Durable Subscription name. Quorum is not required for this operation.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.78.2.78 resume_endpoint_discovery()

```
void resume_endpoint_discovery (
            InstanceHandle_t remote_participant_handle )
```

<<***extension***>> *(*p. *155)* Initiates endpoint discovery with the specified remote **com.rti.dds.domain.Domain↩ Participant** (p. 670).

If the operation returns com.rti.dds.infrastructure.RETCODE_OK, the **com.rti.dds.domain.DomainParticipant** (p. 670) will initiate endpoint discovery with the remote **com.rti.dds.domain.DomainParticipant** (p. 670) provided as a parameter.

When **com.rti.dds.infrastructure.DiscoveryQosPolicy.enable_endpoint_discovery** (p. 668) is set to com.rti.dds.↩ infrastructure.false, this operation allows the RTI Connext application to select for which remote DomainParticipants endpoint discovery is performed. By disabling endpoint discovery, the **DomainParticipant** (p. 670) will not store any state about remote endpoints and will not send local endpoint information to remote DomainParticipants.

If **com.rti.dds.infrastructure.DiscoveryQosPolicy.enable_endpoint_discovery** (p. 668) is set to com.rti.dds.↩ infrastructure.true, endpoint discovery will automatically occur for every discovered **com.rti.dds.domain.Domain↩ Participant** (p. 670). In this case, invoking this operation will have no effect and will return com.rti.dds.infrastructure.↩ RETCODE_OK.

When **com.rti.dds.infrastructure.DiscoveryQosPolicy.enable_endpoint_discovery** (p. 668) is set to com.rti.↩ dds.infrastructure.false, you have two options after a remote **com.rti.dds.domain.DomainParticipant** (p. 670) is discovered:

- Call this operation to enable endpoint discovery. After invoking this operation, the **com.rti.dds.domain.Domain↩ Participant** (p. 670) will start to exchange endpoint information so that matching and communication can occur with the remote **com.rti.dds.domain.DomainParticipant** (p. 670).
- Call the **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723) operation to permanently ignore endpoint discovery with the remote **com.rti.dds.domain.DomainParticipant** (p. 670).

Setting **com.rti.dds.infrastructure.DiscoveryQosPolicy.enable_endpoint_discovery** (p. 668) to com.rti.dds.↩ infrastructure.false enables application-level authentication use cases, in which a **com.rti.dds.domain.Domain**↩ **Participant** (p. 670) will initiate endpoint discovery with a remote **com.rti.dds.domain.DomainParticipant** (p. 670) after successful authentication at the application level.

The `remote_participant_handle` paremeter is the one that appears in the **com.rti.dds.subscription.**↩ **SampleInfo** (p. 1634) retrieved when reading the data samples available for the built-in **builtin.ParticipantBuiltin**↩ **TopicDataDataReader** (p. 1354).

If the specified remote **com.rti.dds.domain.DomainParticipant** (p. 670) is not in the database of discovered Domain↩ Participants or has been previously ignored, this operation will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

This operation can be called multiple times on the same remote participant. If endpoint discovery has already been resumed, successive calls will have no effect and will return com.rti.dds.infrastructure.RETCODE_OK.

Parameters

remote_participant_handle	<<***in***>> *(*p. *156)* Handle of a discovered **com.rti.dds.domain.DomainParticipant** (p. 670) for which endpoint discovery is to be resumed. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261),or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.dds.infrastructure.DiscoveryQosPolicy (p. 665)

8.78.2.79 register_contentfilter()

```
void register_contentfilter (
             String filter_name,
              ContentFilter contentfilter )
```

<<***extension***>> *(*p. *155)* Register a content filter which can be used to create a **com.rti.dds.topic.ContentFiltered**↩ **Topic** (p. 436).

DDS specifies a SQL-like content filter for use by content filtered topics. If this filter does not meet your filtering requirements, you can register a custom filter.

To use a custom filter, it must be registered in the following places:

- In any application that uses the custom filter to create a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) and the corresponding **com.rti.dds.subscription.DataReader** (p. 450).
- In each application that writes the data to the applications mentioned above.

For example, suppose Application A on the subscription side creates a Topic named X and a ContentFilteredTopic named filteredX (and a corresponding DataReader), using a previously registered content filter, myFilter. With only that, you will have filtering at the subscription side. If you also want to perform filtering in any application that publishes Topic X, then you also need to register the same definition of the ContentFilter myFilter in that application.

Each `filter_name` can only be used to registered a content filter once with a **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Parameters

filter_name	*<<**in**>> (p. 156)* Name of the filter. The name must be unique within the **com.rti.dds.domain.DomainParticipant** (p. 670) and must not exceed 255 characters. Cannot be NULL.
contentfilter	*<<**in**>> (p. 156)* Content filter to be registered. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipant.unregister_contentfilter (p. 741)

8.78.2.80 lookup_contentfilter()

```
ContentFilter lookup_contentfilter (
            String filter_name )
```

*<<**extension**>> (p. 155)* Lookup a content filter previously registered with **com.rti.dds.domain.Domain↩ Participant.register_contentfilter** (p. 740).

Parameters

filter_name	*<<**in**>> (p. 156)* Name of the filter. Cannot be NULL.

Returns

NULL if the given filter_name has not been previously registered to the **com.rti.dds.domain.DomainParticipant** (p. 670) with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740). Otherwise, return the **com.rti.dds.topic.ContentFilter** (p. 433) that has been previosuly registered with the given filter_name.

See also

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 740)

8.78.2.81 unregister_contentfilter()

```
void unregister_contentfilter (
            String filter_name )
```

<<***extension***>> *(p. 155)* Unregister a content filter previously registered with **com.rti.dds.domain.Domain↩ Participant.register_contentfilter** (p. 740).

A `filter_name` can be unregistered only if it has been previously registered to the **com.rti.dds.domain.Domain↩ Participant** (p. 670) with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 740).

The unregistration of filter is not allowed if there are any existing **com.rti.dds.topic.ContentFilteredTopic** (p. 436) objects that are using the filter. If the operation is called on a filter with existing **com.rti.dds.topic.ContentFilteredTopic** (p. 436) objects attached to it, this operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_↩ NOT_MET** (p. 1598).

If there are still existing discovered **com.rti.dds.subscription.DataReader** (p. 450) s with the same `filter_name` and the filter's compile method of the filter have previously been called on the discovered **com.rti.dds.subscription.↩ DataReader** (p. 450) s, finalize method of the filter will be called on those discovered **com.rti.dds.subscription.Data↩ Reader** (p. 450) s before the content filter is unregistered. This means filtering will now be performed on the application that is creating the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

filter_name	<<***in***>> *(p. 156)* Name of the filter. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

See also

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 740)

8.78.2.82 get_typecode()

```
TypeCode get_typecode (
            String type_name )
```

<<***extension***>> *(p. 155)* Retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) of a type registered with the **com.rti.dds.domain.DomainParticipant** (p. 670) based on the registration name.

Every data type used in a **com.rti.dds.domain.DomainParticipant** (p. 670) has a registered type name which is the name specified at the time the type is registered with the **com.rti.dds.domain.DomainParticipant** (p. 670).

This operation retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) for the type given the registared type name. The `type_name` argument must correspond to a name used to register a type with the **com.rti.dds.domain.Domain↩ Participant** (p. 670). Otherwise, this method will return NULL.

Type registration in a **com.rti.dds.domain.DomainParticipant** (p. 670) is performed by a call to the **com.rti.ndds.↩ example.FooTypeSupport.register_type** (p. 1119) operation on a derived class of the **com.rti.dds.topic.Type↩ Support** (p. 1941) interface.

Type registration might occur directly via an application call to the **com.rti.ndds.example.FooTypeSupport.register↩ _type** (p. 1119) operation, or indirectly by the RTI Connext infrastructure as a result of parsing an XML configuration file that refers to that type.

If a type is registered by the RTI Connext infrastructure as a result of parsing an XML configuration file, the **com.↩ rti.dds.topic.TypeSupport** (p. 1941) can be created either from a type description found in the XML files, or else by calling the **com.rti.ndds.example.FooTypeSupport.register_type** (p. 1119) operation on a **com.rti.dds.topic.Type↩ Support** (p. 1941) that has been registered with **com.rti.dds.domain.DomainParticipantFactory** (p. 761). If a **com.↩ rti.dds.topic.TypeSupport** (p. 1941) is registered with the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) this mechanism takes precedence over the creation of a **com.rti.dds.topic.TypeSupport** (p. 1941) from a type decription in the XML file.

To register a **com.rti.dds.topic.TypeSupport** (p. 1941) with the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) use the operation **com.rti.dds.domain.DomainParticipantFactory.register_type_support** (p. 786).

If the **com.rti.dds.topic.TypeSupport** (p. 1941) is created from a type description found in the XML files, the resulting type support will be a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995). In this case the **com.rti.dds.↩ publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) uses to write and read data of that type will be the **com.rti.dds.dynamicdata.DynamicDataWriter** (p. 1003) and **com.rti.dds.dynamicdata.Dynamic↩ DataReader** (p. 959), respectively.

Parameters

type_name	<<***in***>> *(p. 156)* Name of the type whose TypeCode is retrieved.

Returns

The TypeCode of a registered type with the **com.rti.dds.domain.DomainParticipant** (p. 670) or NULL if no type was registered with the participant under that name.

8.78.2.83 get_participant_protocol_status()

```
void get_participant_protocol_status (
            DomainParticipantProtocolStatus status )
```

<<***extension***>> *(p. 155)* Get the domain participant protocol status for this participant.

This also resets the status so that it is no longer considered changed.

Parameters

status	<<***inout***>> *(p. 156)* **com.rti.dds.domain.DomainParticipantProtocolStatus** (p. 794) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.78.2.84 get_implicit_publisher()

```
Publisher get_implicit_publisher ( )
```

<<***extension***>> *(p. 155)* Returns the implicit **com.rti.dds.publication.Publisher** (p. 1466). If an implicit Publisher does not already exist, this creates one.

There can only be one implicit Publisher per **DomainParticipant** (p. 670).

The implicit Publisher is created with **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) and no Listener.

This implicit Publisher will be deleted automatically when the following methods are called: **com.rti.dds.domain.DomainParticipant.delete_contained_entities** (p. 728), or **com.rti.dds.domain.DomainParticipant.delete_publisher** (p. 696) with the implicit publisher as a parameter. Additionally, when a **DomainParticipant** (p. 670) is deleted, if there are no attached DataWriters that belong to the implicit Publisher, the implicit Publisher will be implicitly deleted.

Returns

The implicit publisher

See also

com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 46)

com.rti.dds.domain.DomainParticipant.create_publisher (p. 693)

8.78.2.85 get_implicit_subscriber()

```
Subscriber get_implicit_subscriber ( )
```

<<***extension***>> *(p. 155)* Returns the implicit **com.rti.dds.subscription.Subscriber** (p. 1730). If an implicit Subscriber does not already exist, this creates one.

There can only be one implicit Subscriber per **DomainParticipant** (p. 670).

The implicit Subscriber is created with **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) and no Listener.

This implicit Subscriber will be deleted automatically when the following methods are called: **com.rti.dds.domain.DomainParticipant.delete_contained_entities** (p. 728), or **com.rti.dds.domain.DomainParticipant.delete_subscriber** (p. 699) with the subscriber as a parameter. Additionally, when a **DomainParticipant** (p. 670) is deleted, if there are no attached DataReaders that belong to the implicit Subscriber, the implicit Subscriber will be implicitly deleted.

MT Safety:

UNSAFE. it is not safe to create the implicit subscriber while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 687).

Returns

The implicit subscriber

See also

com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 47)

com.rti.dds.domain.DomainParticipant.create_subscriber (p. 697)

8.78.2.86 lookup_publisher_by_name()

```
abstract  Publisher lookup_publisher_by_name (
             String publisher_name )  [abstract]
```

<<***extension***>> *(p. 155)* Looks up a **com.rti.dds.publication.Publisher** (p. 1466) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).

Every **com.rti.dds.publication.Publisher** (p. 1466) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves a **com.rti.dds.publication.Publisher** (p. 1466) within the **com.rti.dds.domain.DomainParticipant** (p. 670) given the entity's name. If there are several **com.rti.dds.publication.Publisher** (p. 1466) with the same name within the **com.rti.dds.domain.DomainParticipant** (p. 670), this function returns the first matching occurrence.

Parameters

publisher_name	<<***in***>> *(*p. *156)* Entity name of the **com.rti.dds.publication.Publisher** (p. 1466).

Returns

The first **com.rti.dds.publication.Publisher** (p. 1466) found with the specified name or NULL if it is not found.

See also

com.rti.dds.domain.DomainParticipant.lookup_datawriter_by_name (p. 746)

8.78.2.87 lookup_subscriber_by_name()

```
abstract  Subscriber lookup_subscriber_by_name (
            String subscriber_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Retrieves a **com.rti.dds.subscription.Subscriber** (p. 1730) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).

Every **com.rti.dds.subscription.Subscriber** (p. 1730) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves a **com.rti.dds.subscription.Subscriber** (p. 1730) within the **com.rti.dds.domain.Domain↩ Participant** (p. 670) given the entity's name. If there are several **com.rti.dds.subscription.Subscriber** (p. 1730) with the same name within the **com.rti.dds.domain.DomainParticipant** (p. 670), this function returns the first matching occurrence.

Parameters

subscriber_name	<<***in***>> *(*p. *156)* Entity name of the **com.rti.dds.subscription.Subscriber** (p. 1730).

Returns

The first **com.rti.dds.subscription.Subscriber** (p. 1730) found with the specified name or NULL if it is not found.

See also

com.rti.dds.domain.DomainParticipant.lookup_datareader_by_name (p. 747)

8.78.2.88 lookup_datawriter_by_name()

```
abstract  DataWriter lookup_datawriter_by_name (
            String datawriter_full_name )  [abstract]
```

<<***extension***>> *(p. 155)* Looks up a **com.rti.dds.publication.DataWriter** (p. 553) by its entity name within this **com.rti.dds.domain.DomainParticipant** (p. 670).

Every **com.rti.dds.publication.DataWriter** (p. 553) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

Every **com.rti.dds.publication.Publisher** (p. 1466) in the system has an entity name which is also configured and stored in the EntityName policy.

This operation retrieves a **com.rti.dds.publication.DataWriter** (p. 553) within a **com.rti.dds.publication.Publisher** (p. 1466) given the specified name which encodes both to the **com.rti.dds.publication.DataWriter** (p. 553) and the **com.rti.dds.publication.Publisher** (p. 1466) name.

If there are several **com.rti.dds.publication.DataWriter** (p. 553) with the same name within the corresponding **com.↩ rti.dds.publication.Publisher** (p. 1466) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **com.rti.dds.publication.Publisher** (p. 1466) to which the **com.rti.dds.publication.DataWriter** (p. 553) belongs and the entity name of of the **com.rti.dds.publication.Data↩ Writer** (p. 553) itself, separated by a double colon "::". For example: MyPublisherName::MyDataWriterName

The plain name contains the **com.rti.dds.publication.DataWriter** (p. 553) name only. In this situation it is implied that the **com.rti.dds.publication.DataWriter** (p. 553) belongs to the implicit **com.rti.dds.publication.Publisher** (p. 1466) so the use of a plain name is equivalent to specifying a fully qualified name with the **com.rti.dds.publication.Publisher** (p. 1466) name part being "implicit". For example: the plain name "MyDataWriterName" is equivalent to specifiying the fully qualified name "implicit::MyDataWriterName"

The **com.rti.dds.publication.DataWriter** (p. 553) is only looked up within the **com.rti.dds.publication.Publisher** (p. 1466) specified in the fully qualified name, or within the implicit **com.rti.dds.publication.Publisher** (p. 1466) if the name was not fully qualified.

Parameters

datawriter_full_name	<<***in***>> *(p. 156)* Entity name or fully-qualified entity name of the **com.rti.dds.publication.DataWriter** (p. 553).

Returns

The first **com.rti.dds.publication.DataWriter** (p. 553) found with the specified name or NULL if it is not found.

See also

com.rti.dds.publication.Publisher.lookup_datawriter_by_name (p. 1487)

com.rti.dds.domain.DomainParticipant.lookup_publisher_by_name (p. 745)

8.78.2.89 lookup_datareader_by_name()

```
abstract  DataReader lookup_datareader_by_name (
            String datareader_full_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Retrieves up a **com.rti.dds.subscription.DataReader** (p. 450) by its entity name in this **com.rti.dds.domain.DomainParticipant** (p. 670).

Every **com.rti.dds.subscription.DataReader** (p. 450) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

Every **com.rti.dds.subscription.Subscriber** (p. 1730) in the system has an entity name which is also configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves a **com.rti.dds.subscription.DataReader** (p. 450) within a **com.rti.dds.subscription.↩ Subscriber** (p. 1730) given the specified name which encodes both to the **com.rti.dds.subscription.DataReader** (p. 450) and the **com.rti.dds.subscription.Subscriber** (p. 1730) name.

If there are several **com.rti.dds.subscription.DataReader** (p. 450) with the same name within the corresponding **com.rti.dds.subscription.Subscriber** (p. 1730) this function returns the first matching occurrence.

The specified name might be given as a fully-qualified entity name or as a plain name.

The fully qualified entity name is a concatenation of the **com.rti.dds.subscription.Subscriber** (p. 1730) to which the **com.rti.dds.subscription.DataReader** (p. 450) belongs and the entity name of of the **com.rti.dds.subscription.↩ DataReader** (p. 450) itself, separated by a double colon "::". For example: MySubscriberName::MyDataReaderName

The plain name contains the **com.rti.dds.subscription.DataReader** (p. 450) name only. In this situation it is implied that the **com.rti.dds.subscription.DataReader** (p. 450) belongs to the implicit **com.rti.dds.subscription.Subscriber** (p. 1730) so the use of a plain name is equivalent to specifying a fully qualified name with the **com.rti.dds.↩ subscription.Subscriber** (p. 1730) name part being "implicit". For example: the plain name "MyDataReaderName" is equivalent to specifiying the fully qualified name "implicit::MyDataReaderName"

The **com.rti.dds.subscription.DataReader** (p. 450) is only looked up within the **com.rti.dds.subscription.Subscriber** (p. 1730) specified in the fully qualified name, or within the implicit **com.rti.dds.subscription.Subscriber** (p. 1730) if the name was not fully qualified.

Parameters

datareader_full_name	<<***in***>> *(*p. *156)* Full entity name of the **com.rti.dds.subscription.DataReader** (p. 450).

Returns

The first **com.rti.dds.subscription.DataReader** (p. 450) found with the specified name or NULL if it is not found.

See also

com.rti.dds.subscription.Subscriber.lookup_datareader_by_name (p. 1753)

com.rti.dds.domain.DomainParticipant.lookup_subscriber_by_name (p. 746)

8.78.2.90 take_discovery_snapshot() [1/2]

```
void take_discovery_snapshot ( )
```

Take a snapshot of the remote participants discovered by a local one.

The snapshot will be printed through the **com.rti.ndds.config.Logger** (p. 1267). A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"

role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

------------------------------------------------------------------------
```

Exceptions

One	of the **Standard Return Codes** (p. 261).

8.78.2.91 take_discovery_snapshot() [2/2]

```
void take_discovery_snapshot (
             String file_name )
```

Take a snapshot of the remote participants discovered by a local one.

The snapshot will be printed in the file specified by file_name. A possible output may be the following:

```
Remote participants that match the local participant domain=0

name="participantTestName" role="participantTestRole" id="1"

guid_prefix="0x0101D8D1,0x20B83C0D,0x4495246E"

------------------------------------------------------------------------

1. 0x0101542A,0x2C59B595,0xA1693BDF name="participantTestName"

role="participantTestRole"

unicastLocators="udpv4://192.168.1.170:7411"

------------------------------------------------------------------------
```

Parameters

file_name	<<***in***>> *(p. 156)* Name of the file where snapshot should be printed.

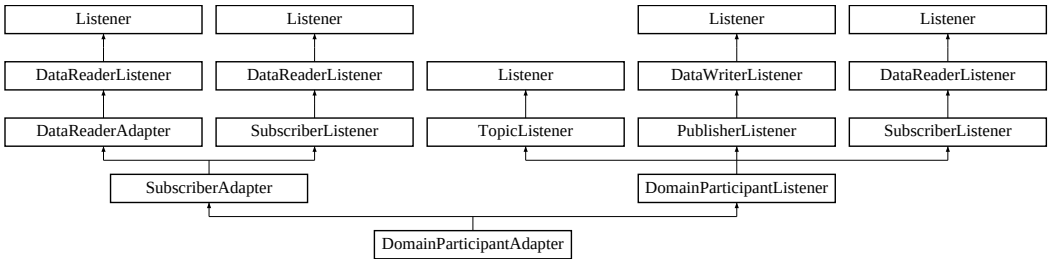
Exceptions

One	of the **Standard Return Codes** (p. 261).

8.79 DomainParticipantAdapter Class Reference

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Inheritance diagram for DomainParticipantAdapter:



Public Member Functions

- void **on_invalid_local_identity_status_advance_notice** (**DomainParticipant** participant, InvalidLocal↩ IdentityAdvanceNoticeStatus invalidLocalIdentityAdvanceNoticeStatus)

  *Notifies the user that the identity of the local **com.rti.dds.domain.DomainParticipant** (p. 670) is about to expire.*
- void **on_inconsistent_topic** (**Topic** topic, **InconsistentTopicStatus** status)

  Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.
- void **on_offered_deadline_missed** (**DataWriter** writer, **OfferedDeadlineMissedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.
- void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.
- void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.
- void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.
- void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  <<***extension***>> *(p. 155) A matched reliable reader has become active or become inactive.*
- void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  <<***extension***>> *(p. 155) A change has occurred in the writer's cache of unacknowledged samples.*
- void **on_sample_removed** (**DataWriter** writer, **Cookie_t** cookie)

  <<***extension***>> *(p. 155) Called when a sample is removed from the DataWriter queue.*
- void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

  <<***extension***>> *(p. 155) Notifies when an instance is replaced in DataWriter queue.*
- void **on_application_acknowledgment** (**DataWriter** writer, **AcknowledgmentInfo** ackInfo)

*<<**extension**>> (p. 155) Called when a sample is application-acknowledged*

- void **on_service_request_accepted** (**DataWriter** writer, **ServiceRequestAcceptedStatus** status)

  *<<**extension**>> (p. 155) Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.*

8.79.1 Detailed Description

*<<**extension**>> (p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.79.2 Member Function Documentation

8.79.2.1 on_invalid_local_identity_status_advance_notice()

```
void on_invalid_local_identity_status_advance_notice (
              DomainParticipant participant,
             InvalidLocalIdentityAdvanceNoticeStatus invalidLocalIdentityAdvanceNoticeStatus )
```

Notifies the user that the identity of the local **com.rti.dds.domain.DomainParticipant** (p. 670) is about to expire.

Implements **DomainParticipantListener** (p. 793).

8.79.2.2 on_inconsistent_topic()

```
void on_inconsistent_topic (
              Topic topic,
              InconsistentTopicStatus status )
```

Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1807) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1807) of the same topic name.

Parameters

topic	*<<**out**>> (p. 156)* Locally created **com.rti.dds.topic.Topic** (p. 1807) that triggers the listener callback
status	*<<**out**>> (p. 156)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1807)

Implements **TopicListener** (p. 1823).

8.79.2.3 on_offered_deadline_missed()

```
void on_offered_deadline_missed (
              DataWriter writer,
              OfferedDeadlineMissedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **DEADLINE** (p. 217) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **com.rti.dds.publication.DataWriter** (p. 553) failed to provide data for an instance.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 591).

8.79.2.4 on_offered_incompatible_qos()

```
void on_offered_incompatible_qos (
              DataWriter writer,
              OfferedIncompatibleQosStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 612) of the **com.rti.dds.publication.↩ DataWriter** (p. 553) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 450). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 553) has discovered a **com.rti.dds.↩ subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 591).

8.79.2.5 on_liveliness_lost()

```
void on_liveliness_lost (
            DataWriter writer,
            LivelinessLostStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **LIVELINESS** (p. 239) qos policy was not respected; this **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 553) simply remains not alive for another liveliness period.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 592).

8.79.2.6 on_publication_matched()

```
void on_publication_matched (
            DataWriter writer,
            PublicationMatchedStatus status )
```

Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 553) has found a **com.rti.dds.subscription.↩ DataReader** (p. 450) that matches the **com.rti.dds.topic.Topic** (p. 1807), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 450) that was previously considered to be matched.

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current publication match status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 592).

8.79.2.7 on_reliable_reader_activity_changed()

```
void on_reliable_reader_activity_changed (
            DataWriter writer,
            ReliableReaderActivityChangedStatus status )
```

<<***extension***>> *(*p. *155)* A matched reliable reader has become active or become inactive.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 593).

8.79.2.8 on_reliable_writer_cache_changed()

```
void on_reliable_writer_cache_changed (
            DataWriter writer,
            ReliableWriterCacheChangedStatus status )
```

<<***extension***>> *(*p. *155)* A change has occurred in the writer's cache of unacknowledged samples.

This listener callback is triggered when:

- The cache is empty (contains no unacknowledged samples).
- The cache is full (the number of unacknowledged samples has reached the value specified in **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)).
- The number of unacknowledged samples has reached **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1607) or **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark** (p. 1607).

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

Implements **DataWriterListener** (p. 593).

8.79.2.9 on_sample_removed()

```
void on_sample_removed (
             DataWriter writer,
             Cookie_t cookie )
```

<<***extension***>> *(p. 155)* Called when a sample is removed from the DataWriter queue.

This callback is called only if the sample was written with a **com.rti.dds.infrastructure.Cookie_t** (p. 442) with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), or if this writer uses **Zero Copy** (p. 56) transfer over shared memory" or **FlatData Topic-Types** (p. 55) "FlatData language binding".

Parameters

writer	<<***out***>> *(p. 156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
cookie	<<***out***>> *(p. 156)* • If this sample was written with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), this field contains a copy of the cookie set in **com.rti.dds.infrastructure.WriteParams_t** (p. 1994). • If this writer uses **Zero Copy transfer over shared memory** (p. 56) or **FlatData language binding** (p. 55), this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using com.rti.dds.infrastructure.Cookie_t.to_pointer

See also

com.rti.ndds.example.FooDataWriter.get_loan

Implements **DataWriterListener** (p. 594).

8.79.2.10 on_instance_replaced()

```
void on_instance_replaced (
             DataWriter writer,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Notifies when an instance is replaced in DataWriter queue.

This callback is called when an instance is replaced by the **com.rti.dds.publication.DataWriter** (p. 553) due to instance resource limits being reached. This callback returns to the user the handle of the replaced instance, which can be used to get the key of the replaced instance using the **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114) API.

Because this callback can be called within the context of an in-progress write, dispose, or unregister call, most APIs on the DataWriter must not be used. The only DataWriter APIs that are safe to call within this callback are:

- **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114)
- com.rti.ndds.example.FooDataWriter.create_data
- com.rti.ndds.example.FooDataWriter.delete_data
- **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566)
- **com.rti.dds.publication.DataWriter.is_matched_subscription_active** (p. 567)
- **com.rti.dds.publication.DataWriter.get_matched_subscription_participant_data** (p. 569)
- **com.rti.dds.publication.DataWriter.get_topic** (p. 569)
- **com.rti.dds.publication.DataWriter.get_publisher** (p. 570)
- **com.rti.dds.publication.DataWriter.is_sample_app_acknowledged** (p. 571)

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
handle	<<***out***>> *(*p. *156)* Handle of the replaced instance

Implements **DataWriterListener** (p. 594).

8.79.2.11 on_application_acknowledgment()

```
void on_application_acknowledgment (
              DataWriter writer,
              AcknowledgmentInfo info )
```

<<***extension***>> *(*p. *155)* Called when a sample is application-acknowledged

Applicable only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1531)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **com.rti.dds.subscription.DataReader** (p. 450). Also provides user-specified response data sent from the **com.rti.dds.subscription.DataReader** (p. 450) by the acknowledgment message.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
info	<<***out***>> *(*p. *156)* **com.rti.dds.publication.AcknowledgmentInfo** (p. 330) of the acknowledged sample

Implements **DataWriterListener** (p. 595).

8.79.2.12 on_service_request_accepted()

```
void on_service_request_accepted (
              DataWriter writer,
              ServiceRequestAcceptedStatus status )
```

<<***extension***>> *(*p. *155)* Called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) for the **com.rti.dds.↩ subscription.TopicQuery** (p. 1830) service is dispatched to this **com.rti.dds.publication.DataWriter** (p. 553) for processing.

Parameters

writer	<<***out***>> *(*p. *156)* Locally created **com.rti.dds.publication.DataWriter** (p. 553) that triggers the listener callback
status	<<***out***>> *(*p. *156)* Current service request accepted status of locally created **com.rti.dds.publication.DataWriter** (p. 553)

See also

Topic Queries (p. 84)

Implements **DataWriterListener** (p. 596).

8.80 DomainParticipantConfigParams_t Class Reference

<<***extension***>> *(*p. *155)* Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.

Inherits Struct.

Public Attributes

- int **domain_id** = **DOMAIN_ID_USE_XML_CONFIG**

  *Domain ID from which the **com.rti.dds.domain.DomainParticipant** (*p. *670) is created.*
- String **participant_name**

  *Name assigned to the **com.rti.dds.domain.DomainParticipant** (*p. *670).*
- String **participant_qos_library_name**

  *QoS library name containing the QoS profile from which the **com.rti.dds.domain.DomainParticipant** (*p. *670) is created.*
- String **participant_qos_profile_name**

  *QoS profile name from which the **com.rti.dds.domain.DomainParticipant** (*p. *670) is created.*
- String **domain_entity_qos_library_name**

  *QoS library name containing the QoS profile from which the all the **com.rti.dds.infrastructure.DomainEntity** (*p. *669) defined under the participant configuration are created.*
- String **domain_entity_qos_profile_name**

  *QoS profile name from which the all the **com.rti.dds.infrastructure.DomainEntity** (*p. *669) defined under the participant configuration are created.*

Static Public Attributes

- static final int **DOMAIN_ID_USE_XML_CONFIG** = -1

  *<<**extension**>> (p. 155) Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig**↩ **Params_t** (p. 757) to indicate that a participant is created using the domain ID specified in the participant configuration.*
- static final String **ENTITY_NAME_USE_XML_CONFIG**

  *<<**extension**>> (p. 155) Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig**↩ **Params_t** (p. 757) to indicate that a participant created is with an autogenerated entity name.*
- static final String **QOS_ELEMENT_NAME_USE_XML_CONFIG**

  *<<**extension**>> (p. 155) Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig**↩ **Params_t** (p. 757) to indicate that entities are created from the QoS profile specified in the participant configuration.*

8.80.1 Detailed Description

*<<**extension**>> (p. 155)* Input paramaters for creating a participant from configuration. It allows to modify or override some of the properties of the entities defined in the configuration.

8.80.2 Member Data Documentation

8.80.2.1 DOMAIN_ID_USE_XML_CONFIG

```
final int DOMAIN_ID_USE_XML_CONFIG = -1  [static]
```

*<<**extension**>> (p. 155)* Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig**↩ **Params_t** (p. 757) to indicate that a participant is created using the domain ID specified in the participant configuration.

This variable contains a constant sentinel value that is compared when creating the entities from configuration.

8.80.2.2 ENTITY_NAME_USE_XML_CONFIG

```
final String ENTITY_NAME_USE_XML_CONFIG  [static]
```

Initial value:

```
=
            "com.rti.dds.domain.entity_name_use_xml_config"
```

*<<**extension**>> (p. 155)* Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig**↩ **Params_t** (p. 757) to indicate that a participant created is with an autogenerated entity name.

This variable contains a constant sentinel value that is lexicographically compared when creating the entities from configuration.

8.80.2.3 QOS_ELEMENT_NAME_USE_XML_CONFIG

```
final String QOS_ELEMENT_NAME_USE_XML_CONFIG  [static]
```

Initial value:

```
=
            "com.rti.dds.domain.qos_element_name_use_xml_config"
```

<<***extension***>> *(p. 155)* Special value to be used with **com.rti.dds.infrastructure.DomainParticipantConfig↩ Params_t** (p. 757) to indicate that entities are created from the QoS profile specified in the participant configuration.

This variable contains a constant sentinel value that is lexicographically compared when creating the entities from configuration.

8.80.2.4 domain_id

```
int domain_id =  DOMAIN_ID_USE_XML_CONFIG
```

Domain ID from which the **com.rti.dds.domain.DomainParticipant** (p. 670) is created.

Allows overriding the domain ID defined in the configuration for the participant to be created. If the special value **com.rti.dds.infrastructure.DomainParticipantConfigParams_t.DOMAIN_ID_USE_XML_CONFIG** (p. 758) is specified then the ID in the configuration will be used.

8.80.2.5 participant_name

```
String participant_name
```

Initial value:

```
= new String(
            ENTITY_NAME_USE_XML_CONFIG)
```

Name assigned to the **com.rti.dds.domain.DomainParticipant** (p. 670).

This is the name the name that will be set in the **com.rti.dds.domain.DomainParticipantQos.participant_name** (p. 801). It allows overriding the participant name that is generated automatically.

When this member is lexicographically equal to the special value **com.rti.dds.infrastructure.DomainParticipant↩ ConfigParams_t.ENTITY_NAME_USE_XML_CONFIG** (p. 758) then an automatically generated name will be assigned.

8.80.2.6 participant_qos_library_name

```
String participant_qos_library_name
```

Initial value:

```
= new String(
            QOS_ELEMENT_NAME_USE_XML_CONFIG)
```

QoS library name containing the QoS profile from which the **com.rti.dds.domain.DomainParticipant** (p. 670) is created.

Allows overriding the QoS defined in the configuration for the participant to be created. This value only affects to the **com.rti.dds.domain.DomainParticipant** (p. 670).

When this member is lexicographically equal to the special value **com.rti.dds.infrastructure.DomainParticipant↩ ConfigParams_t.QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 758) then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **com.rti.dds.infrastructure.↩ DomainParticipantConfigParams_t.participant_qos_profile_name** (p. 759) will be ignored.

8.80.2.7 participant_qos_profile_name

```
String participant_qos_profile_name
```

Initial value:
```
= new String(
            QOS_ELEMENT_NAME_USE_XML_CONFIG)
```

QoS profile name from which the **com.rti.dds.domain.DomainParticipant** (p. 670) is created.

Allows overriding the QoS defined in the configuration for the participant to be created.This value only affects to the **com.rti.dds.domain.DomainParticipant** (p. 670).

When this member is lexicographically equal to the special value **com.rti.dds.infrastructure.DomainParticipant↩ ConfigParams_t.QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 758) then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **com.rti.dds.infrastructure.↩ DomainParticipantConfigParams_t.participant_qos_library_name** (p. 759) will be ignored.

8.80.2.8 domain_entity_qos_library_name

```
String domain_entity_qos_library_name
```

Initial value:
```
= new String(
            QOS_ELEMENT_NAME_USE_XML_CONFIG)
```

QoS library name containing the QoS profile from which the all the **com.rti.dds.infrastructure.DomainEntity** (p. 669) defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the **com.rti.dds.infrastructure.DomainEntity** (p. 669).

When this member is lexicographically equal to the special value **com.rti.dds.infrastructure.DomainParticipant↩ ConfigParams_t.QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 758) then the QoS library from the configuration will be applied. Also, the same action will apply for the QoS profile and the value in **com.rti.dds.infrastructure.↩ DomainParticipantConfigParams_t.domain_entity_qos_profile_name** (p. 760) will be ignored.

8.80.2.9 domain_entity_qos_profile_name

```
String domain_entity_qos_profile_name
```

Initial value:
```
= new String(
            QOS_ELEMENT_NAME_USE_XML_CONFIG)
```

QoS profile name from which the all the **com.rti.dds.infrastructure.DomainEntity** (p. 669) defined under the participant configuration are created.

Allows overriding the QoS defined in the configuration for the domain entities to be created. This value only affects to the **com.rti.dds.infrastructure.DomainEntity** (p. 669).

When this member is lexicographically equal to the special value **com.rti.dds.infrastructure.DomainParticipant↩ ConfigParams_t.QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 758) then the QoS profile from the configuration will be applied. Also, the same action will apply for the QoS library and the value in **com.rti.dds.infrastructure.↩ DomainParticipantConfigParams_t.domain_entity_qos_library_name** (p. 760) will be ignored.

8.81 DomainParticipantFactory Class Reference

<<***singleton***>> *(*p. *156)* <<***interface***>> *(*p. *156)* Allows creation and destruction of **com.rti.dds.domain.Domain↩ Participant** (p. 670) objects.

Public Member Functions

- abstract **DomainParticipant** **create_participant** (int domainId, **DomainParticipantQos** qos, **Domain↩ ParticipantListener** listener, int mask)

  Creates a new ***com.rti.dds.domain.DomainParticipant*** *(*p. *670) object.*
- abstract void **delete_participant** (**DomainParticipant** a_participant)

  Deletes an existing ***com.rti.dds.domain.DomainParticipant*** *(*p. *670).*
- abstract void **get_default_participant_qos** (**DomainParticipantQos** qos)

  Initializes the ***com.rti.dds.domain.DomainParticipantQos*** *(*p. *795) instance with default values.*
- abstract void **set_default_participant_qos** (**DomainParticipantQos** qos)

  Sets the default ***com.rti.dds.domain.DomainParticipantQos*** *(*p. *795) values for this domain participant factory.*
- abstract void **set_default_participant_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Sets the default* ***com.rti.dds.domain.DomainParticipantQos*** *(*p. *795) values for this domain participant factory based on the input XML QoS profile.*
- abstract **DomainParticipant** **lookup_participant** (int domainId)

  Locates an existing ***com.rti.dds.domain.DomainParticipant*** *(*p. *670).*
- abstract void **get_qos** (**DomainParticipantFactoryQos** qos)

  Gets the value for participant factory QoS.
- abstract void **set_qos** (**DomainParticipantFactoryQos** qos)

  Sets the value for a participant factory QoS.
- abstract void **load_profiles** ()

  <<***extension***>> *(*p. *155) Loads the XML QoS profiles.*
- abstract void **reload_profiles** ()

  <<***extension***>> *(*p. *155) Reloads the XML QoS profiles.*
- abstract void **unload_profiles** ()

  <<***extension***>> *(*p. *155) Unloads the XML QoS profiles.*
- abstract String **get_default_library** ()

  <<***extension***>> *(*p. *155) Gets the default XML library associated with a* ***com.rti.dds.domain.DomainParticipant↩ Factory*** *(*p. *761).*
- abstract void **set_default_library** (String library_name)

  <<***extension***>> *(*p. *155) Sets the default XML library for a* ***com.rti.dds.domain.DomainParticipantFactory*** *(*p. *761).*
- abstract String **get_default_profile** ()

  <<***extension***>> *(*p. *155) Gets the default XML profile associated with a* ***com.rti.dds.domain.DomainParticipant↩ Factory*** *(*p. *761).*
- abstract void **set_default_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Sets the default XML profile for a* ***com.rti.dds.domain.DomainParticipantFactory*** *(*p. *761).*
- abstract String **get_default_profile_library** ()

  <<***extension***>> *(*p. *155) Gets the library where the default XML profile is contained for a* ***com.rti.dds.domain.↩ DomainParticipantFactory*** *(*p. *761).*
- abstract void **get_participant_factory_qos_from_profile** (**DomainParticipantFactoryQos** qos, String library_name, String profile_name)

*<<**extension**>> (p. 155) Gets the **com.rti.dds.domain.DomainParticipantFactoryQos** (p. 787) values associated with the input XML QoS profile.*

- abstract void **get_participant_qos_from_profile** (**DomainParticipantQos** qos, String library_name, String profile_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.domain.DomainParticipantQos** (p. 795) values associated with the input XML QoS profile.*

- abstract void **get_publisher_qos_from_profile** (**PublisherQos** qos, String library_name, String profile_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.publication.PublisherQos** (p. 1490) values associated with the input XML QoS profile.*

- abstract void **get_subscriber_qos_from_profile** (**SubscriberQos** qos, String library_name, String profile_↩ name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.subscription.SubscriberQos** (p. 1756) values associated with the input XML QoS profile.*

- abstract void **get_datawriter_qos_from_profile** (**DataWriterQos** qos, String library_name, String profile_↩ name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.publication.DataWriterQos** (p. 612) values associated with the input XML QoS profile.*

- abstract void **get_datawriter_qos_from_profile_w_topic_name** (**DataWriterQos** qos, String library_name, String profile_name, String topic_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.publication.DataWriterQos** (p. 612) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

- abstract void **get_datareader_qos_from_profile** (**DataReaderQos** qos, String library_name, String profile_↩ name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.subscription.DataReaderQos** (p. 517) values associated with the input XML QoS profile.*

- abstract void **get_datareader_qos_from_profile_w_topic_name** (**DataReaderQos** qos, String library_name, String profile_name, String topic_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.subscription.DataReaderQos** (p. 517) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

- abstract void **get_topic_qos_from_profile** (**TopicQos** qos, String library_name, String profile_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.topic.TopicQos** (p. 1824) values associated with the input XML QoS profile.*

- abstract void **get_topic_qos_from_profile_w_topic_name** (**TopicQos** qos, String library_name, String profile_name, String topic_name)

  *<<**extension**>> (p. 155) Gets the **com.rti.dds.topic.TopicQos** (p. 1824) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

- abstract void **get_qos_profile_libraries** (**StringSeq** library_names)

  *<<**extension**>> (p. 155) Gets the names of all XML QoS profile libraries associated with the **com.rti.dds.domain.**↩ **DomainParticipantFactory** (p. 761)*

- abstract void **get_qos_profiles** (**StringSeq** profile_names, String library_name)

  *<<**extension**>> (p. 155) Gets the names of all XML QoS profiles associated with the input XML QoS profile library.*

- abstract **DomainParticipant** **create_participant_with_profile** (int domainId, String library_name, String profile_name, **DomainParticipantListener** listener, int mask)

  *<<**extension**>> (p. 155) Creates a new **com.rti.dds.domain.DomainParticipant** (p. 670) object using the **com.rti.**↩ **dds.domain.DomainParticipantQos** (p. 795) associated with the input XML QoS profile.*

- abstract void **unregister_thread** ()

  *<<**extension**>> (p. 155) Allows the user to release thread specific resources kept by the middleware.*

- abstract **DomainParticipant** **create_participant_from_config** (String configuration_name)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.domain.DomainParticipant** (p. 670) given its configuration name from a description provided in an XML configuration file.*

- abstract **DomainParticipant create_participant_from_config_w_params** (String configuration_name, **DomainParticipantConfigParams_t** params)

  *<<**extension**>> (p. 155) Creates a **com.rti.dds.domain.DomainParticipant** (p. 670) given its configuration name from a description provided in an XML configuration file and a set of parameters that allow changing some properties of such configuration.*
- abstract **DomainParticipant lookup_participant_by_name** (String participant_name)

  *<<**extension**>> (p. 155) Looks up a **com.rti.dds.domain.DomainParticipant** (p. 670) by its entity name in the **com.rti.dds.domain.DomainParticipantFactory** (p. 761).*
- abstract void **register_type_support** (**TypeSupport** type_support, String type_name)

  *<<**extension**>> (p. 155) Registers a **com.rti.dds.topic.TypeSupport** (p. 1941) with the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) to enable automatic registration if the corresponding type, should it be needed by a **com.rti.dds.domain.DomainParticipant** (p. 670).*

Static Public Member Functions

- static final **DomainParticipantFactory get_instance** ()

  Gets the singleton instance of this class.
- static final void **finalize_instance** ()

  *<<**extension**>> (p. 155) Destroys the singleton instance of this class.*

Static Public Attributes

- static final **DomainParticipantQos PARTICIPANT_QOS_DEFAULT**

  *Special value for creating a **DomainParticipant** (p. 670) with default QoS.*
- static **DomainParticipantFactory TheParticipantFactory** = create_singletonI()

  *Can be used as an alias for the singleton factory returned by the operation **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764).*
- static **DomainParticipantConfigParams_t PARTICIPANT_CONFIG_PARAMS_DEFAULT**

  *Special value for creating a **com.rti.dds.domain.DomainParticipant** (p. 670) from configuration using default parameters.*

8.81.1 Detailed Description

*<<**singleton**>> (p. 156) <<**interface**>> (p. 156)* Allows creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 670) objects.

The sole purpose of this class is to allow the creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 670) objects. This class itself is a *<<**singleton**>> (p. 156)*, and accessed via the **get_instance()** (p. 764) method, and destroyed with **finalize_instance()** (p. 764) method.

A single application can participate in multiple domains by instantiating multiple **com.rti.dds.domain.DomainParticipant** (p. 670) objects.

An application may even instantiate multiple participants in the same domain. Participants in the same domain exchange data in the same way regardless of whether they are in the same application or different applications or on the same node or different nodes; their location is transparent.

There are two important caveats:

- When there are multiple participants on the same node (in the same application or different applications) in the same domain, the application(s) must make sure that the participants do not try to bind to the same port numbers. You must disambiguate between the participants by setting a participant ID for each participant (**com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id** (p. 1990)). The port numbers used by a participant are calculated based on both the participant index and the domain ID, so if all participants on the same node have different participant indexes, they can coexist in the same domain.
- You cannot mix entities from different participants. For example, you cannot delete a topic on a different participant than you created it from, and you cannot ask a subscriber to create a reader for a topic created from a participant different than the subscriber's own participant. (Note that it is permissable for an application built on top of RTI Connext to know about entities from different participants. For example, an application could keep references to a reader from one domain and a writer from another and then bridge the domains by writing the data received in the reader callback.)

See also

com.rti.dds.domain.DomainParticipant (p. 670)

8.81.2 Member Function Documentation

8.81.2.1 get_instance()

```
static final DomainParticipantFactory get_instance ( ) [static]
```

Gets the singleton instance of this class.

com.rti.dds.domain.DomainParticipantFactory.TheParticipantFactory (p. 42) can be used as an alias for the singleton factory returned by this operation.

Returns

The singleton **com.rti.dds.domain.DomainParticipantFactory** (p. 761) instance.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

See also

com.rti.dds.domain.DomainParticipantFactory.TheParticipantFactory (p. 42)

References **DomainParticipantFactory.TheParticipantFactory**.

8.81.2.2 finalize_instance()

```
static final void finalize_instance ( )  [static]
```

<<***extension***>> *(p. 155)* Destroys the singleton instance of this class.

Only necessary to explicitly reclaim resources used by the participant factory singleton. Note that on many OSs, these resources are automatically reclaimed by the OS when the program terminates. However, some memory-check tools still flag these as unreclaimed. So this method provides a way to clean up memory used by the participant factory.

Precondition

All participants created from the factory have been deleted.

Postcondition

All resources belonging to the factory have been reclaimed. Another call to **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764) will return a new lifecycle of the singleton.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

See also

com.rti.dds.domain.DomainParticipantFactory.TheParticipantFactory (p. 42)

References **DomainParticipantFactory.TheParticipantFactory**.

8.81.2.3 create_participant()

```
abstract  DomainParticipant create_participant (
             int domainId,
              DomainParticipantQos qos,
              DomainParticipantListener listener,
             int mask )  [abstract]
```

Creates a new **com.rti.dds.domain.DomainParticipant** (p. 670) object.

Precondition

The specified QoS policies must be consistent or the operation will fail and no **com.rti.dds.domain.Domain↩ Participant** (p. 670) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

Safe.

Parameters

domain↩ Id	<<***in***>> *(*p. *156)* ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622).
qos	<<***in***>> *(*p. *156)* the **DomainParticipant** (p. 670)'s QoS. The special value **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42) can be used to indicate that the **com.rti.dds.domain.DomainParticipant** (p. 670) should be created with the default **com.rti.dds.domain.DomainParticipantQos** (p. 795) set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 761). Cannot be NULL.
listener	<<***in***>> *(*p. *156)* the domain participant's listener.
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See **com.rti.dds.infrastructure.StatusKind** (p. 1701).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.domain.DomainParticipantQos (p. 795) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

NDDS_DISCOVERY_PEERS (p. 222)

com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile (p. 782)

com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos (p. 767)

com.rti.dds.domain.DomainParticipant.set_listener (p. 718)

8.81.2.4 delete_participant()

```
abstract void delete_participant (
              DomainParticipant a_participant )  [abstract]
```

Deletes an existing **com.rti.dds.domain.DomainParticipant** (p. 670).

Precondition

All domain entities belonging to the participant must have already been deleted. Otherwise it fails with the error **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.domain.DomainParticipant** (p. 670) will not be called after this method returns successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_participant	*<<**in**>> (p. 156)* **com.rti.dds.domain.DomainParticipant** (p. 670) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.81.2.5 get_default_participant_qos()

```
abstract void get_default_participant_qos (
              DomainParticipantQos qos )  [abstract]
```

Initializes the **com.rti.dds.domain.DomainParticipantQos** (p. 795) instance with default values.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768), or **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 768), or else, if the call was never made, the default values listed in **com.rti.dds.domain.DomainParticipantQos** (p. 795).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***out***>> *(*p. *156)* the domain participant's QoS Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 765)

8.81.2.6 set_default_participant_qos()

```
abstract void set_default_participant_qos (
              DomainParticipantQos qos )  [abstract]
```

Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 795) values for this domain participant factory.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(*p. *156)* Qos to be filled up. The special value **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 765)

8.81.2.7 set_default_participant_qos_with_profile()

```
abstract void set_default_participant_qos_with_profile (
            String library_name,
            String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 795) values for this domain participant factory based on the input XML QoS profile.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

This default value will be used for newly created **com.rti.dds.domain.DomainParticipant** (p. 670) if **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 765) is called.

Precondition

The **com.rti.dds.domain.DomainParticipantQos** (p. 795) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a domain participant factory while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768)

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile (p. 782)

8.81.2.8 lookup_participant()

```
abstract  DomainParticipant lookup_participant (
            int domainId )  [abstract]
```

Locates an existing **com.rti.dds.domain.DomainParticipant** (p. 670).

If no such **com.rti.dds.domain.DomainParticipant** (p. 670) exists, the operation will return NULL value.

If multiple **com.rti.dds.domain.DomainParticipant** (p. 670) entities belonging to that domainId exist, then the operation will return one of them. It is not specified which one.

Parameters

domain↩ Id	<<***in***>> *(p. 156)* ID of the domain participant to lookup.

Returns

domain participant if it exists, or NULL

8.81.2.9 get_qos()

```
abstract void get_qos (
              DomainParticipantFactoryQos qos )  [abstract]
```

Gets the value for participant factory QoS.

Parameters

qos	<<***inout***>> *(p. 156)* QoS to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.10 set_qos()

```
abstract void set_qos (
              DomainParticipantFactoryQos qos )  [abstract]
```

Sets the value for a participant factory QoS.

The **com.rti.dds.domain.DomainParticipantFactoryQos.entity_factory** (p. 791) can be changed. The other policies are immutable.

Note that despite having QoS, the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) is not an **com.rti.dds.↩ infrastructure.Entity** (p. 1029).

Parameters

qos	<<***in***>> *(p. 156)* Set of policies to be applied to **com.rti.dds.domain.DomainParticipantFactory** (p. 761). Policies must be consistent. Immutable policies can only be changed before calling any other RTI Connext methods except for **com.rti.dds.domain.DomainParticipantFactory.get_qos** (p. 770) Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) if immutable policy is changed, or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) if policies are inconsistent

See also

com.rti.dds.domain.DomainParticipantFactoryQos (p. 787) for rules on consistency among QoS

8.81.2.11 load_profiles()

```
abstract void load_profiles ( )  [abstract]
```

<<***extension***>> *(p. 155)* Loads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **com.rti.dds.domain.DomainParticipant** (p. 670) is created or explicitly, after a call to this method.

This has the same effect as **com.rti.dds.domain.DomainParticipantFactory.reload_profiles()** (p. 771).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1393)

8.81.2.12 reload_profiles()

```
abstract void reload_profiles ( )  [abstract]
```

<<***extension***>> *(p. 155)* Reloads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **com.rti.dds.domain.DomainParticipant** (p. 670) is created or explicitly, after a call to this method.

This has the same effect as **com.rti.dds.domain.DomainParticipantFactory.load_profiles()** (p. 771).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1393)

8.81.2.13 unload_profiles()

```
abstract void unload_profiles ( )  [abstract]
```

<<***extension***>> *(p. 155)* Unloads the XML QoS profiles.

The resources associated with the XML QoS profiles are freed. Any reference to the profiles after calling this method will fail with an error.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1393)

8.81.2.14 get_default_library()

```
abstract String get_default_library ( )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the default XML library associated with a **com.rti.dds.domain.DomainParticipant↩ Factory** (p. 761).

Returns

The returned library name is determined as follows:

- If it was previously set with **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773), this function returns that library name
- Otherwise, if one or more profiles have the XML attribute `is_default_qos="true"`, this function returns the library where the last one is contained.
- Otherwise, this function returns null.

See also

com.rti.dds.domain.DomainParticipantFactory.set_default_library (p. 773)

8.81.2.15 set_default_library()

```
abstract void set_default_library (
            String library_name )  [abstract]
```

<<***extension***>> *(p. 155)* Sets the default XML library for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

Any API requiring a library_name as a parameter can use null to refer to the default library set with this function.

Note: if the library set with this function no longer exists after reloading the QoS profiles (for example, by changing **com.rti.dds.domain.DomainParticipantFactoryQos.profile** (p. 791)) the default library will be set to the last library containing a profile with the attribute `is_default_qos=true` or null no such library exists.

See also

com.rti.dds.domain.DomainParticipantFactory.set_default_profile (p. 774) for more information.

Parameters

library_name	<<***in***>> *(p. 156)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipantFactory.get_default_library (p. 772)

8.81.2.16 get_default_profile()

```
abstract String get_default_profile ( )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the default XML profile associated with a **com.rti.dds.domain.DomainParticipant↩ Factory** (p. 761).

Returns

The returned profile name is determined as follows:

- If it was previously set with **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774), this function returns that profile name
- Otherwise, if one or more profiles have the XML attribute `is_default_qos="true"`, this function returns the name of one of them
- Otherwise, this function returns null.

See also

com.rti.dds.domain.DomainParticipantFactory.set_default_profile (p. 774)

8.81.2.17 set_default_profile()

```
abstract void set_default_profile (
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Sets the default XML profile for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

This method specifies the profile that will be used as the default the next time a default **DomainParticipantFactory** (p. 761) profile is needed during a call to a **DomainParticipantFactory** (p. 761) method. When calling a **com.rti.dds.↩ domain.DomainParticipantFactory** (p. 761) method that requires a `profile_name` parameter, you can use null to refer to the default profile. (This same information applies to setting a default library.)

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **com.↩ rti.dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 768) (you may pass in null after having called **set_default_profile()** (p. 774)).

Note: if the profile set with this function no longer exists after reloading the QoS profiles (for example, by changing **com.rti.dds.domain.DomainParticipantFactoryQos.profile** (p. 791)) the default profile will be set to the last one marked with the attribute `is_default_qos=true` or null no such profile exists.

Parameters

library_name	<<***in***>> *(p. 156)* The library name containing the profile.
profile_name	<<***in***>> *(p. 156)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.domain.DomainParticipantFactory.get_default_profile (p. 773)

com.rti.dds.domain.DomainParticipantFactory.get_default_profile_library (p. 775)

8.81.2.18 get_default_profile_library()

```
abstract String get_default_profile_library ( )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the library where the default XML profile is contained for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

The default profile library is automatically set when **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774) is called.

This library can be different than the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) default library (see **com.rti.dds.domain.DomainParticipantFactory.get_default_library** (p. 772)).

Returns

The default profile library for the profile returned by **com.rti.dds.domain.DomainParticipantFactory.get_default_profile** (p. 773), or null if that profile is null.

See also

com.rti.dds.domain.DomainParticipantFactory.get_default_profile (p. 773)

8.81.2.19 get_participant_factory_qos_from_profile()

```
abstract void get_participant_factory_qos_from_profile (
              DomainParticipantFactoryQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.domain.DomainParticipantFactoryQos** (p. 787) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.20 get_participant_qos_from_profile()

```
abstract void get_participant_qos_from_profile (
              DomainParticipantQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Gets the **com.rti.dds.domain.DomainParticipantQos** (p. 795) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.21 get_publisher_qos_from_profile()

```
abstract void get_publisher_qos_from_profile (
              PublisherQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.publication.PublisherQos** (p. 1490) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.22 get_subscriber_qos_from_profile()

```
abstract void get_subscriber_qos_from_profile (
              SubscriberQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.subscription.SubscriberQos** (p. 1756) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.23 get_datawriter_qos_from_profile()

```
abstract void get_datawriter_qos_from_profile (
              DataWriterQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.publication.DataWriterQos** (p. 612) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(p. 156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.24 get_datawriter_qos_from_profile_w_topic_name()

```
abstract void get_datawriter_qos_from_profile_w_topic_name (
              DataWriterQos qos,
             String library_name,
             String profile_name,
             String topic_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.publication.DataWriterQos** (p. 612) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).
topic_name	<<***in***>> *(*p. *156)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.25 get_datareader_qos_from_profile()

```
abstract void get_datareader_qos_from_profile (
              DataReaderQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Gets the **com.rti.dds.subscription.DataReaderQos** (p. 517) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.26 get_datareader_qos_from_profile_w_topic_name()

```
abstract void get_datareader_qos_from_profile_w_topic_name (
              DataReaderQos qos,
             String library_name,
             String profile_name,
             String topic_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Gets the **com.rti.dds.subscription.DataReaderQos** (p. 517) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).
topic_name	<<***in***>> *(*p. *156)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.27 get_topic_qos_from_profile()

```
abstract void get_topic_qos_from_profile (
              TopicQos qos,
             String library_name,
             String profile_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Gets the **com.rti.dds.topic.TopicQos** (p. 1824) values associated with the input XML QoS profile.

Parameters

qos	<<***out***>> *(*p. *156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.28 get_topic_qos_from_profile_w_topic_name()

```
abstract void get_topic_qos_from_profile_w_topic_name (
              TopicQos qos,
             String library_name,
             String profile_name,
             String topic_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the **com.rti.dds.topic.TopicQos** (p. 1824) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

qos	<<***out***>> *(p. 156)* Qos to be filled up. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).
topic_name	<<***in***>> *(p. 156)* Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connext will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.29 get_qos_profile_libraries()

```
abstract void get_qos_profile_libraries (
              StringSeq library_names )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the names of all XML QoS profile libraries associated with the **com.rti.dds.domain.↩ DomainParticipantFactory** (p. 761)

Parameters

library_names	<<***out***>> *(p. 156)* **com.rti.dds.infrastructure.StringSeq** (p. 1718) to be filled with names of XML QoS profile libraries. Cannot be NULL.

8.81.2.30 get_qos_profiles()

```
abstract void get_qos_profiles (
              StringSeq profile_names,
             String library_name )  [abstract]
```

<<***extension***>> *(p. 155)* Gets the names of all XML QoS profiles associated with the input XML QoS profile library.

Parameters

profile_names	<<***out***>> *(p. 156)* **com.rti.dds.infrastructure.StringSeq** (p. 1718) to be filled with names of XML QoS profiles. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).

8.81.2.31 create_participant_with_profile()

```
abstract  DomainParticipant create_participant_with_profile (
             int domainId,
             String library_name,
             String profile_name,
              DomainParticipantListener listener,
             int mask )  [abstract]
```

<<***extension***>> *(p. 155)* Creates a new **com.rti.dds.domain.DomainParticipant** (p. 670) object using the **com.rti.dds.domain.DomainParticipantQos** (p. 795) associated with the input XML QoS profile.

Precondition

The **com.rti.dds.domain.DomainParticipantQos** (p. 795) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.domain.DomainParticipant** (p. 670) will be created.

If you want to create multiple participants on a given host in the same domain, make sure each one has a different participant index (set in the **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

Safe.

Parameters

domainId	<<***in***>> *(*p. *156)* ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622).
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 773)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 774)).
listener	<<***in***>> *(*p. *156)* the **DomainParticipant** (p. 670)'s listener.
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See **com.rti.dds.infrastructure.StatusKind** (p. 1701).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.domain.DomainParticipantQos (p. 795) for rules on consistency among QoS

com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 42)

NDDS_DISCOVERY_PEERS (p. 222)

com.rti.dds.domain.DomainParticipantFactory.create_participant() (p. 765)

com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos() (p. 767)

com.rti.dds.domain.DomainParticipant.set_listener() (p. 718)

8.81.2.32 unregister_thread()

```
abstract void unregister_thread ( )  [abstract]
```

<<***extension***>> *(*p. *155)* Allows the user to release thread specific resources kept by the middleware.

This function should be called by the user right before exiting a thread where DDS API were used. In this way the middleware will be able to free all the resources related to this specific thread. The best approach is to call the function during the thread deletion after all the DDS related API have have been called.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.81.2.33 create_participant_from_config()

```
abstract  DomainParticipant create_participant_from_config (
            String configuration_name )  [abstract]
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.domain.DomainParticipant** (p. 670) given its configuration name from a description provided in an XML configuration file.

This operation creates a **com.rti.dds.domain.DomainParticipant** (p. 670) registering all the necessary data types and creating all the contained entities (**com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.DataReader** (p. 450)) from a description given in an XML configuration file.

The configuration name is the fully qualified name of the XML participant object, consisting of the name of the participant library plus the name of participant configuration.

For example the name "MyParticipantLibrary::PublicationParticipant" can be used to create the domain participant from the description in an XML file with contents shown in the snippet below:

```
<participant_library name="MyParticipantLibrary">

    <domain_participant name="PublicationParticipant" domain_ref="MyDomainLibrary::HelloWorldDomain">

        <publisher name="MyPublisher">

            <data_writer name="HelloWorldWriter" topic_ref="HelloWorldTopic"/>

        </publisher>

    </domain_participant>

</participant_library>
```

The entities belonging to the newly created **com.rti.dds.domain.DomainParticipant** (p. 670) can be retrieved with the help of lookup operations such as: **com.rti.dds.domain.DomainParticipant.lookup_datareader_by_name** (p. 747).

This operation is equivalent to call **com.rti.dds.domain.DomainParticipantFactory.create_participant_from_config_w_params** (p. 785) passing **com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_CONFIG_PARAMS_DEFAULT** (p. 43) as parameters.

MT Safety:

Safe.

Parameters

configuration_name	<<***in***>> *(p. 156)* Name of the participant configuration in the XML file.

Returns

The created **com.rti.dds.domain.DomainParticipant** (p. 670) or NULL on error.

See also

com.rti.dds.domain.DomainParticipantFactory.lookup_participant_by_name (p. 785)

com.rti.dds.domain.DomainParticipant.lookup_topicdescription (p. 722)

com.rti.dds.domain.DomainParticipant.lookup_publisher_by_name (p. 745)

com.rti.dds.domain.DomainParticipant.lookup_subscriber_by_name (p. 746)

com.rti.dds.domain.DomainParticipant.lookup_datareader_by_name (p. 747)

com.rti.dds.domain.DomainParticipant.lookup_datawriter_by_name (p. 746)

com.rti.dds.publication.Publisher.lookup_datawriter_by_name (p. 1487)

com.rti.dds.subscription.Subscriber.lookup_datareader_by_name (p. 1753)

8.81.2.34 create_participant_from_config_w_params()

```
abstract  DomainParticipant create_participant_from_config_w_params (
             String configuration_name,
              DomainParticipantConfigParams_t params )  [abstract]
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.domain.DomainParticipant** (p. 670) given its configuration name from a description provided in an XML configuration file and a set of parameters that allow changing some properties of such configuration.

The operation will create a **com.rti.dds.domain.DomainParticipant** (p. 670) the same way specified in **com.rti.dds.domain.DomainParticipantFactory.create_participant_from_config** (p. 784).

In addition, this operation allows overriding the domain ID, participant name, and entities QoS specified in the configuration.

MT Safety:

Safe.

Parameters

configuration_name	<<***in***>> *(p. 156)* Name of the participant configuration in the XML file.
params	<<***in***>> *(p. 156)* input parameters that allow changing some properties of the configuration referred to by `configuration_name`.

Returns

The created **com.rti.dds.domain.DomainParticipant** (p. 670) or NULL on error.

8.81.2.35 lookup_participant_by_name()

```
abstract  DomainParticipant lookup_participant_by_name (
            String participant_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Looks up a **com.rti.dds.domain.DomainParticipant** (p. 670) by its entity name in the **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

Every **com.rti.dds.domain.DomainParticipant** (p. 670) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves a **com.rti.dds.domain.DomainParticipant** (p. 670) within the **com.rti.dds.domain.Domain↩ ParticipantFactory** (p. 761) given the entity's name. If there are several **com.rti.dds.domain.DomainParticipant** (p. 670) with the same name within the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) this function returns the first matching occurrence.

Parameters

participant_name	<<***in***>> *(*p. *156)* Entity name of the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be null.

Returns

The first **com.rti.dds.domain.DomainParticipant** (p. 670) found with the specified name or NULL if it is not found.

8.81.2.36 register_type_support()

```
abstract void register_type_support (
              TypeSupport type_support,
             String type_name )  [abstract]
```

<<***extension***>> *(*p. *155)* Registers a **com.rti.dds.topic.TypeSupport** (p. 1941) with the **com.rti.dds.domain.↩ DomainParticipantFactory** (p. 761) to enable automatic registration if the corresponding type, should it be needed by a **com.rti.dds.domain.DomainParticipant** (p. 670).

Types refered by the **com.rti.dds.topic.Topic** (p. 1807) entities within a **com.rti.dds.domain.DomainParticipant** (p. 670) must be registered with the **com.rti.dds.domain.DomainParticipant** (p. 670).

Type registration in a **com.rti.dds.domain.DomainParticipant** (p. 670) is performed by a call to the **com.rti.ndds.↩ example.FooTypeSupport.register_type** (p. 1119) operation. This can be done directly from the application code or indirectly by the RTI Connext infrastructure as a result of parsing an XML configuration file that refers to that type.

The **com.rti.dds.domain.DomainParticipantFactory.register_type_support** (p. 786) operation provides the **com.↩ rti.dds.domain.DomainParticipantFactory** (p. 761) with the information it needs to automatically call the **com.rti.↩ ndds.example.FooTypeSupport.register_type** (p. 1119) operation and register the corresponding type if the type is needed as a result of parsing an XML configuration file.

Automatic type registration while parsing XML files can also be done by the RTI Connext infrastructure based on the type description provided in the XML files. If the **com.rti.dds.topic.TypeSupport** (p. 1941) has been registered with the **com.rti.dds.domain.DomainParticipantFactory** (p. 761) this definition takes precedence over the description of the type given in the XML file.

The **com.rti.dds.domain.DomainParticipantFactory.register_type_support** (p. 786) operation receives a **com.rti.↩ ndds.example.FooTypeSupport** (p. 1118) object as a parameter. This object's class is normally generated using rtiddsgen from a description of the corresponsing type in IDL, XML, or XSD.

The typical workflow when using this function is as follows: Define the data-type in IDL (or XML, or XSD) in a file. E.g. Foo.idl Run rtiddsgen in that file to generate the TypeSupport files, for the desired programming language. E.g. in C++ FooTypeSupport.h FooTypeSupport.cxx Include the proper header file (e.g. FooTypeSupport.h) in your program and call **com.rti.dds.domain.DomainParticipantFactory.register_type_support** (p. 786) passing the function that was generated by rtiddsgen. E.g. FooTypeSupport::register_type Include the TypeSupport source file in your project such that it is compiled and linked. E.g. FooTypeSupport.cxx

You may refer to the `Getting Started Guide` for additional details in this process.

Note that only one register function is allowed per registered type name.

Parameters

type_support	<<***in***>> *(p. 156)* **com.rti.dds.topic.TypeSupport** (p. 1941) object to be used for registering the type with a **com.rti.dds.domain.DomainParticipant** (p. 670).

Parameters

type_name	<<***in***>> *(p. 156)* Name the type is registered with.

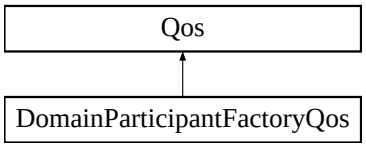
Exceptions

One	of the **Standard Return Codes** (p. 261)

8.82 DomainParticipantFactoryQos Class Reference

QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

Inheritance diagram for DomainParticipantFactoryQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**DomainParticipantFactoryQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.*
- String **toString** (**DomainParticipantFactoryQos** baseQos)

  *Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.*

Public Attributes

- final **EntityFactoryQosPolicy entity_factory**

  *Entity factory policy, **ENTITY_FACTORY** (p. 234).*
- final **SystemResourceLimitsQosPolicy resource_limits**

  *<<**extension**>> (p. 155) System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 266).*
- final **ProfileQosPolicy profile**

  *<<**extension**>> (p. 155) Qos profile policy, **PROFILE** (p. 247).*
- final **LoggingQosPolicy logging**

  *<<**extension**>> (p. 155) Logging qos policy, **LOGGING** (p. 241).*
- final **MonitoringQosPolicy monitoring**

  *<<**extension**>> (p. 155) Monitoring qos policy, **MONITORING** (p. 241).*

8.82.1 Detailed Description

QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 761)

See also

QoS Policies (p. 250) and allowed ranges within each Qos.

8.82.2 Member Function Documentation

8.82.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 788) overloads allow formatting the output and printing only the differences with respect to another **DomainParticipantFactoryQos** (p. 787) object.

```
DomainParticipantFactoryQos qos = new DomainParticipantFactoryQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
DomainParticipantFactoryQos baseQos = new DomainParticipantFactoryQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **DomainParticipant↩ FactoryQos** (p. 787) and the documented default.

Returns

The string representation of the Qos.

References **DomainParticipantFactoryQos.toString()**.

Referenced by **DomainParticipantFactoryQos.toString()**.

8.82.2.2 toString() [2/4]

```
String toString (
              DomainParticipantFactoryQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 49) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipant.DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL**.

8.82.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipantFactoryQos.toString()**.

8.82.2.4 toString() [4/4]

```
String toString (
              DomainParticipantFactoryQos baseQos )
```

Obtains a string representation of a **DomainParticipantFactoryQos** (p. 787) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 49) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipantFactoryQos.toString()**.

8.82.3 Member Data Documentation

8.82.3.1 entity_factory

```
final EntityFactoryQosPolicy entity_factory
```

Entity factory policy, **ENTITY_FACTORY** (p. 234).

8.82.3.2 resource_limits

```
final SystemResourceLimitsQosPolicy resource_limits
```

<<***extension***>> *(p. 155)* System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 266).

Note: This QoS policy cannot be configured from XML configuration files.

8.82.3.3 profile

```
final ProfileQosPolicy profile
```

<<***extension***>> *(p. 155)* Qos profile policy, **PROFILE** (p. 247).

Note: This QoS policy cannot be configured from XML configuration files.

8.82.3.4 logging

```
final LoggingQosPolicy logging
```

<<***extension***>> *(p. 155)* Logging qos policy, **LOGGING** (p. 241).

8.82.3.5 monitoring

```
final MonitoringQosPolicy monitoring
```

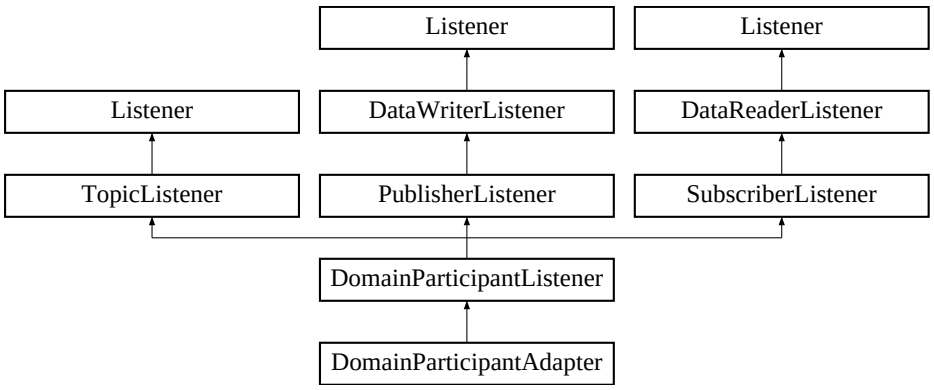
<<***extension***>> *(*p. *155)* Monitoring qos policy, **MONITORING** (p. 241).

This QoS policy is used to collect and emit telemetry data of a Connext application using RTI Monitoring Library 2.0.

8.83 DomainParticipantListener Interface Reference

<<***interface***>> *(*p. *156)* Listener for participant status.

Inheritance diagram for DomainParticipantListener:



Public Member Functions

- void **on_invalid_local_identity_status_advance_notice** (**DomainParticipant** participant, InvalidLocal↩ IdentityAdvanceNoticeStatus invalidLocalIdentityAdvanceNoticeStatus)

  *Notifies the user that the identity of the local **com.rti.dds.domain.DomainParticipant** (*p. *670) is about to expire.*

8.83.1 Detailed Description

<<***interface***>> *(*p. *156)* Listener for participant status.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Status:

Status Kinds (p. 262)

This is the interface that can be implemented by an application-provided class and then registered with the **com.rti.dds.domain.DomainParticipant** (p. 670) such that the application can be notified by RTI Connext of relevant status changes.

The **com.rti.dds.domain.DomainParticipantListener** (p. 792) interface extends all other Listener interfaces and has no additional operation beyond the ones defined by the more general listeners.

The purpose of the **com.rti.dds.domain.DomainParticipantListener** (p. 792) is to be the listener of last resort that is notified of all status changes not captured by more specific listeners attached to the **com.rti.dds.infrastructure.DomainEntity** (p. 669) objects. When a relevant status change occurs, RTI Connext will first attempt to notify the listener attached to the concerned **com.rti.dds.infrastructure.DomainEntity** (p. 669) if one is installed. Otherwise, RTI Connext will notify the Listener attached to the **com.rti.dds.domain.DomainParticipant** (p. 670).

Important: Because a **com.rti.dds.domain.DomainParticipantListener** (p. 792) may receive callbacks pertaining to many different entities, it is possible for the same listener to receive multiple callbacks simultaneously in different threads. (Such is not the case for listeners of other types.) It is therefore critical that users of this listener provide their own protection for any thread-unsafe activities undertaken in a **com.rti.dds.domain.DomainParticipantListener** (p. 792) callback.

Note: Due to a thread-safety issue, the destruction of a **DomainParticipantListener** (p. 792) from an enabled **DomainParticipant** (p. 670) should be avoided – even if the **DomainParticipantListener** (p. 792) has been removed from the **DomainParticipant** (p. 670). (This limitation does not affect the Java API.)

See also

com.rti.dds.infrastructure.Listener (p. 1236)

com.rti.dds.domain.DomainParticipant.set_listener (p. 718)

8.83.2 Member Function Documentation

8.83.2.1 on_invalid_local_identity_status_advance_notice()

```
void on_invalid_local_identity_status_advance_notice (
              DomainParticipant participant,
             InvalidLocalIdentityAdvanceNoticeStatus invalidLocalIdentityAdvanceNoticeStatus )
```

Notifies the user that the identity of the local **com.rti.dds.domain.DomainParticipant** (p. 670) is about to expire.

Implemented in **DomainParticipantAdapter** (p. 751).

8.84 DomainParticipantProtocolStatus Class Reference

<<***extension***>> *(*p. *155)* The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.

Inherits Struct.

Public Attributes

- long **corrupted_rtps_message_count**

  The number of corrupted RTPS messages detected by the domain participant.
- long **corrupted_rtps_message_count_change**

  The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.
- **Time_t last_corrupted_message_timestamp**

  The timestamp when the last corrupted RTPS message was detected by the domain participant.

8.84.1 Detailed Description

<<***extension***>> *(*p. *155)* The status of a participant's protocol related metrics, like the number of corrupted messages, change in corrupted messages and timestamp of the last corrupted message.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670) The corrupted messages are detected by validating the received CRC. The participant protocol status can be obtained by enabling **com.rti.dds.infrastructure.WireProtocolQosPolicy.compute_crc** (p. 1993) and **com.rti.dds.infrastructure.WireProtocolQosPolicy.check_crc** (p. 1993) at the publishing and subscribing application respectively.

8.84.2 Member Data Documentation

8.84.2.1 corrupted_rtps_message_count

```
long corrupted_rtps_message_count
```

The number of corrupted RTPS messages detected by the domain participant.

Counts the corrupted RTPS messages received by the participant. It includes messages belonging to discovery and user traffic.

8.84.2.2 corrupted_rtps_message_count_change

```
long corrupted_rtps_message_count_change
```

The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.

8.84.2.3 last_corrupted_message_timestamp

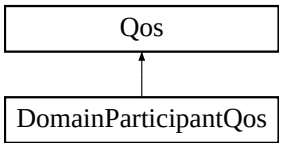
```
Time_t last_corrupted_message_timestamp
```

The timestamp when the last corrupted RTPS message was detected by the domain participant.

8.85 DomainParticipantQos Class Reference

QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 670) entity.

Inheritance diagram for DomainParticipantQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**DomainParticipantQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **DomainParticipantQos** (p. 795) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **DomainParticipantQos** (p. 795) object.*
- String **toString** (**DomainParticipantQos** baseQos)

  *Obtains a string representation of a **DomainParticipantQos** (p. 795) object.*

Public Attributes

- final **UserDataQosPolicy user_data**

  *User data policy, **USER_DATA** (p. 278).*
- final **EntityFactoryQosPolicy entity_factory**

  *Entity factory policy, **ENTITY_FACTORY** (p. 234).*
- final **WireProtocolQosPolicy wire_protocol**

  *<<**extension**>> (p. 155) Wire Protocol policy, **WIRE_PROTOCOL** (p. 280).*
- final **TransportBuiltinQosPolicy transport_builtin**

  *<<**extension**>> (p. 155) Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 269).*
- final **TransportUnicastQosPolicy default_unicast**

  *<<**extension**>> (p. 155) Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 276).*
- final **DiscoveryQosPolicy discovery**

  *<<**extension**>> (p. 155) Discovery policy, **DISCOVERY** (p. 221).*
- final **DomainParticipantResourceLimitsQosPolicy resource_limits**

  *<<**extension**>> (p. 155) Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 229).*
- final **EventQosPolicy event**

  *<<**extension**>> (p. 155) Event policy, **EVENT** (p. 235).*
- final **ReceiverPoolQosPolicy receiver_pool**

  *<<**extension**>> (p. 155) Receiver pool policy, **RECEIVER_POOL** (p. 257).*
- final **DatabaseQosPolicy database**

  *<<**extension**>> (p. 155) Database policy, **DATABASE** (p. 210).*
- final **DiscoveryConfigQosPolicy discovery_config**

  *<<**extension**>> (p. 155) Discovery config policy, **DISCOVERY_CONFIG** (p. 218).*
- final **PropertyQosPolicy property**

  *<<**extension**>> (p. 155) Property policy, **PROPERTY** (p. 248). See also `Property Reference Guide`.*
- final **EntityNameQosPolicy participant_name**

  *<<**extension**>> (p. 155) The participant name. **ENTITY_NAME** (p. 234)*
- final **ServiceQosPolicy service**

  *<<**extension**>> (p. 155) The service qos policy. **SERVICE** (p. 261)*
- final **TypeSupportQosPolicy type_support**

  *<<**extension**>> (p. 155) Type support data, **TYPESUPPORT** (p. 277).*
- final **TransportMulticastMappingQosPolicy multicast_mapping**

  *<<**extension**>> (p. 155) The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 272)*
- final **PartitionQosPolicy partition**

  *<<**extension**>> (p. 155) The partition qos policy. **PARTITION** (p. 246)*

8.85.1 Detailed Description

QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 670) entity.

Certain members must be set in a consistent manner:

Length of **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 799) .value <= **com.rti.dds.domain.Domain↩ ParticipantQos.resource_limits** (p. 800) .participant_user_data_max_length

For **com.rti.dds.domain.DomainParticipantQos.discovery_config** (p. 801) .publication_writer high_watermark <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 800) .local_writer_allocation .max_count heartbeats_per_max_samples <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 800) .local_writer_allocation.max_count

For **com.rti.dds.domain.DomainParticipantQos.discovery_config** (p. 801) .suscription_writer high_watermark <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 800) .local_reader_allocation.max_count heartbeats_per_max_samples <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 800) .local_reader_allocation.max_count

If any of the above are not true, **com.rti.dds.domain.DomainParticipant.set_qos** (p. 714) and **com.rti.dds.domain.DomainParticipant.set_qos_with_profile** (p. 714) and **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 768) will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596), and **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 765) will fail.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

See also

QoS Policies (p. 250) and allowed ranges within each Qos.

NDDS_DISCOVERY_PEERS (p. 222)

8.85.2 Member Function Documentation

8.85.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 797) overloads allow formatting the output and printing only the differences with respect to another **DomainParticipantQos** (p. 795) object.

```
DomainParticipantQos qos = new DomainParticipantQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
DomainParticipantQos baseQos = new DomainParticipantQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value DOMAINPARTICIPANT_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(DOMAINPARTICIPANT_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **DomainParticipantQos** (p. 795) and the documented default.

Returns

The string representation of the Qos.

References **DomainParticipantQos.toString()**.

Referenced by **DomainParticipantQos.toString()**.

8.85.2.2 toString() [2/4]

```
String toString (
              DomainParticipantQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **DomainParticipantQos** (p. 795) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 48) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipant.DOMAINPARTICIPANT_QOS_PRINT_ALL**.

8.85.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **DomainParticipantQos** (p. 795) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipantQos.toString()**.

8.85.2.4 toString() [4/4]

```
String toString (
              DomainParticipantQos baseQos )
```

Obtains a string representation of a **DomainParticipantQos** (p. 795) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 48) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipantQos.toString()**.

8.85.3 Member Data Documentation

8.85.3.1 user_data

```
final  UserDataQosPolicy user_data
```

User data policy, **USER_DATA** (p. 278).

8.85.3.2 entity_factory

```
final  EntityFactoryQosPolicy entity_factory
```

Entity factory policy, **ENTITY_FACTORY** (p. 234).

8.85.3.3 wire_protocol

```
final  WireProtocolQosPolicy wire_protocol
```

<<***extension***>> *(*p. *155)* Wire Protocol policy, **WIRE_PROTOCOL** (p. 280).

The wire protocol (RTPS) attributes associated with the participant.

8.85.3.4 transport_builtin

```
final  TransportBuiltinQosPolicy transport_builtin
```

<<***extension***>> *(*p. *155)* Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 269).

8.85.3.5 default_unicast

```
final  TransportUnicastQosPolicy default_unicast
```

<<***extension***>> *(*p. *155)* Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 276).

8.85.3.6 discovery

```
final  DiscoveryQosPolicy discovery
```

<<***extension***>> *(*p. *155)* Discovery policy, **DISCOVERY** (p. 221).

8.85.3.7 resource_limits

```
final  DomainParticipantResourceLimitsQosPolicy resource_limits
```

<<***extension***>> *(*p. *155)* Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 229).

8.85.3.8 event

```
final  EventQosPolicy event
```

<<***extension***>> *(p. 155)* Event policy, **EVENT** (p. 235).

8.85.3.9 receiver_pool

```
final  ReceiverPoolQosPolicy receiver_pool
```

<<***extension***>> *(p. 155)* Receiver pool policy, **RECEIVER_POOL** (p. 257).

8.85.3.10 database

```
final  DatabaseQosPolicy database
```

<<***extension***>> *(p. 155)* Database policy, **DATABASE** (p. 210).

8.85.3.11 discovery_config

```
final  DiscoveryConfigQosPolicy discovery_config
```

<<***extension***>> *(p. 155)* Discovery config policy, **DISCOVERY_CONFIG** (p. 218).

8.85.3.12 property

```
final  PropertyQosPolicy property
```

<<***extension***>> *(p. 155)* Property policy, **PROPERTY** (p. 248). See also `Property Reference Guide`.

8.85.3.13 participant_name

```
final  EntityNameQosPolicy participant_name
```

<<***extension***>> *(p. 155)* The participant name. **ENTITY_NAME** (p. 234)

8.85.3.14 service

```
final ServiceQosPolicy service
```

<<***extension***>> *(p. 155)* The service qos policy. **SERVICE** (p. 261)

8.85.3.15 type_support

```
final TypeSupportQosPolicy type_support
```

<<***extension***>> *(p. 155)* Type support data, **TYPESUPPORT** (p. 277).

Optional value that is passed to a type plugin's on_participant_attached function.

8.85.3.16 multicast_mapping

```
final TransportMulticastMappingQosPolicy multicast_mapping
```

<<***extension***>> *(p. 155)* The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 272)

8.85.3.17 partition

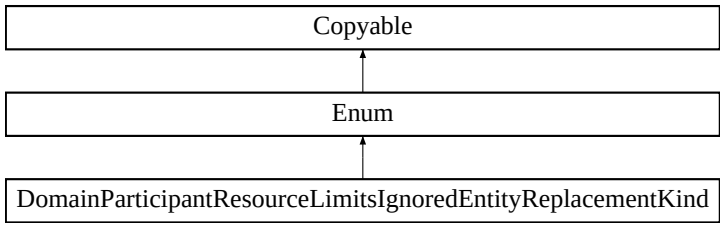
```
final PartitionQosPolicy partition
```

<<***extension***>> *(p. 155)* The partition qos policy. **PARTITION** (p. 246)

8.86 DomainParticipantResourceLimitsIgnoredEntityReplacementKind Class Reference

Available replacement policies for the ignored entities.

Inheritance diagram for DomainParticipantResourceLimitsIgnoredEntityReplacementKind:



Static Public Attributes

- static final **DomainParticipantResourceLimitsIgnoredEntityReplacementKind NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT** = new **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**("NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT", 0)

  Available replacement policies for the ignored entities.
- static final **DomainParticipantResourceLimitsIgnoredEntityReplacementKind NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT** = new **DomainParticipantResourceLimitsIgnoredEntityReplacementKind**("NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT", 1)

  Available replacement policies for the ignored entities.

Additional Inherited Members

8.86.1 Detailed Description

Available replacement policies for the ignored entities.

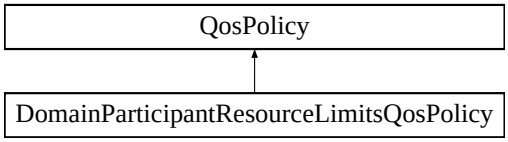
QoS:

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy (p. 803)

8.87 DomainParticipantResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Inheritance diagram for DomainParticipantResourceLimitsQosPolicy:



Public Attributes

- final **AllocationSettings_t local_writer_allocation**

  Allocation settings applied to local DataWriters.
- final **AllocationSettings_t local_reader_allocation**

  Allocation settings applied to local DataReaders.
- final **AllocationSettings_t local_publisher_allocation**

  Allocation settings applied to local Publisher.
- final **AllocationSettings_t local_subscriber_allocation**

  Allocation settings applied to local Subscriber.
- final **AllocationSettings_t local_topic_allocation**

Allocation settings applied to local Topic.

- final **AllocationSettings_t remote_writer_allocation**

  Allocation settings applied to remote DataWriters.

- final **AllocationSettings_t remote_reader_allocation**

  Allocation settings applied to remote DataReaders.

- final **AllocationSettings_t remote_participant_allocation**

  Allocation settings applied to remote DomainParticipants.

- final **AllocationSettings_t matching_writer_reader_pair_allocation**

  Allocation settings applied to matching local writer and remote/local reader pairs.

- final **AllocationSettings_t matching_reader_writer_pair_allocation**

  Allocation settings applied to matching local reader and remote/local writer pairs.

- final **AllocationSettings_t ignored_entity_allocation**

  Allocation settings applied to ignored entities.

- final **AllocationSettings_t content_filtered_topic_allocation**

  Allocation settings applied to content filtered topic.

- final **AllocationSettings_t content_filter_allocation**

  Allocation settings applied to content filter.

- final **AllocationSettings_t read_condition_allocation**

  Allocation settings applied to read condition pool.

- final **AllocationSettings_t query_condition_allocation**

  Allocation settings applied to query condition pool.

- final **AllocationSettings_t outstanding_asynchronous_sample_allocation**

  Allocation settings applied to the maximum number of samples (from all ***com.rti.dds.publication.DataWriter*** *(*p. 553*)) waiting to be asynchronously written.*

- final **AllocationSettings_t flow_controller_allocation**

  Allocation settings applied to flow controllers.

- int **local_writer_hash_buckets**

  Hash_Buckets settings applied to local DataWriters.

- int **local_reader_hash_buckets**

  Number of hash buckets for local DataReaders.

- int **local_publisher_hash_buckets**

  Number of hash buckets for local Publisher.

- int **local_subscriber_hash_buckets**

  Number of hash buckets for local Subscriber.

- int **local_topic_hash_buckets**

  Number of hash buckets for local Topic.

- int **remote_writer_hash_buckets**

  Number of hash buckets for remote DataWriters.

- int **remote_reader_hash_buckets**

  Number of hash buckets for remote DataReaders.

- int **remote_participant_hash_buckets**

  Number of hash buckets for remote DomainParticipants.

- int **matching_writer_reader_pair_hash_buckets**

  Number of hash buckets for matching local writer and remote/local reader pairs.

- int **matching_reader_writer_pair_hash_buckets**

  Number of hash buckets for matching local reader and remote/local writer pairs.

- int **ignored_entity_hash_buckets**

Number of hash buckets for ignored entities.

- int **content_filtered_topic_hash_buckets**

  Number of hash buckets for content filtered topics.

- int **content_filter_hash_buckets**

  Number of hash buckets for content filters.

- int **flow_controller_hash_buckets**

  Number of hash buckets for flow controllers.

- int **max_gather_destinations**

  Maximum number of destinations per RTI Connext send.

- int **participant_user_data_max_length**

  *Maximum length of user data in **com.rti.dds.domain.DomainParticipantQos** (p. 795) and **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349).*

- int **topic_data_max_length**

  *Maximum length of topic data in **com.rti.dds.topic.TopicQos** (p. 1824), builtin.TopicBuiltinTopicData, **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762).*

- int **publisher_group_data_max_length**

  *Maximum length of group data in **com.rti.dds.publication.PublisherQos** (p. 1490) and **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).*

- int **subscriber_group_data_max_length**

  *Maximum length of group data in **com.rti.dds.subscription.SubscriberQos** (p. 1756) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762).*

- int **writer_user_data_max_length**

  *Maximum length of user data in **com.rti.dds.publication.DataWriterQos** (p. 612) and **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).*

- int **reader_user_data_max_length**

  *Maximum length of user data in **com.rti.dds.subscription.DataReaderQos** (p. 517) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762).*

- int **max_partitions**

  *Maximum number of partition name strings allowable in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365).*

- int **max_partition_cumulative_characters**

  *Maximum number of combined characters allowable in all partition names in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365).*

- int **type_code_max_serialized_length**

  Maximum size of serialized string for type code.

- int **type_object_max_serialized_length**

  The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.

- int **serialized_type_object_dynamic_allocation_threshold**

  A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.

- int **type_object_max_deserialized_length**

  The maximum number of bytes that a deserialized TypeObject can consume.

- int **deserialized_type_object_dynamic_allocation_threshold**

  A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject.

- int **contentfilter_property_max_length**

  This field is the maximum length of all data related to a Content-filtered topic.

- int **channel_seq_max_length**

  *Maximum number of channels that can be specified in **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) for MultiChannel DataWriters.*

- int **channel_filter_expression_max_length**

Maximum length of a channel ***com.rti.dds.infrastructure.ChannelSettings_t.filter_expression*** *(*p. *414) in a Multi↩ Channel DataWriter.*

- int **participant_property_list_max_length**

  Maximum number of properties associated with the ***com.rti.dds.domain.DomainParticipant*** *(*p. *670).*

- int **participant_property_string_max_length**

  Maximum string length of the properties associated with the ***com.rti.dds.domain.DomainParticipant*** *(*p. *670).*

- int **writer_property_list_max_length**

  Maximum number of properties associated with a ***com.rti.dds.publication.DataWriter*** *(*p. *553).*

- int **writer_property_string_max_length**

  Maximum string length of the properties associated with a ***com.rti.dds.publication.DataWriter*** *(*p. *553).*

- int **reader_property_list_max_length**

  Maximum number of properties associated with a ***com.rti.dds.subscription.DataReader*** *(*p. *450).*

- int **reader_property_string_max_length**

  Maximum string length of the properties associated with a ***com.rti.dds.subscription.DataReader*** *(*p. *450).*

- int **max_endpoint_groups**

  Maximum number of ***com.rti.dds.infrastructure.EndpointGroup_t*** *(*p. *1022) allowable in a* ***com.rti.dds.↩ infrastructure.AvailabilityQosPolicy*** *(*p. *343).*

- int **max_endpoint_group_cumulative_characters**

  Maximum number of combined role_name characters allowed in all ***com.rti.dds.infrastructure.EndpointGroup_↩ t*** *(*p. *1022) in a* ***com.rti.dds.infrastructure.AvailabilityQosPolicy*** *(*p. *343).*

- int **transport_info_list_max_length**

  Maximum number of installed transports to send and receive information about in builtin.ParticipantBuiltinTopicData.↩ transport_info.

- **DomainParticipantResourceLimitsIgnoredEntityReplacementKind ignored_entity_replacement_kind**

  Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in ***com.rti.dds.↩ infrastructure.DomainParticipantResourceLimitsQosPolicy.ignored_entity_allocation*** *(*p. *810) are reached.*

- final **AllocationSettings_t remote_topic_query_allocation**

  Allocation settings applied to remote TopicQueries.

- int **remote_topic_query_hash_buckets**

  Number of hash buckets for remote TopicQueries.

- int **writer_data_tag_list_max_length**

  Maximum number of data tags associated with a ***com.rti.dds.publication.DataWriter*** *(*p. *553).*

- int **writer_data_tag_string_max_length**

  Maximum string length of the data tags associated with a ***com.rti.dds.publication.DataWriter*** *(*p. *553).*

- int **reader_data_tag_list_max_length**

  Maximum number of data tags associated with a ***com.rti.dds.subscription.DataReader*** *(*p. *450).*

- int **reader_data_tag_string_max_length**

  Maximum string length of the data tags associated with a ***com.rti.dds.subscription.DataReader*** *(*p. *450).*

8.87.1 Detailed Description

Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 670) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

This QoS policy sets maximum size limits on variable-length parameters used by the participant and its contained Entities. It also controls the initial and maximum sizes of data structures used by the participant to store information about locally-created and remotely-discovered entities (such as DataWriters/DataReaders), as well as parameters used by the internal database to size the hash tables it uses.

By default, a **com.rti.dds.domain.DomainParticipant** (p. 670) is allowed to dynamically allocate memory as needed as users create local Entities such as **com.rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.Data↩ Reader** (p. 450) or as the participant discovers new applications to store their information. By setting fixed values for the maximum parameters in this **QosPolicy** (p. 1501), you can bound the memory that can be allocated by a Domain↩ Participant. In addition, by setting the initial values to the maximum values, you can reduce the amount of memory allocated by DomainParticipants after the initialization period. Notice that memory can still be allocated dynamically after the initialization period. For example, when a new local **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.↩ dds.subscription.DataReader** (p. 450) is created, the initial memory required for its queue is allocated dynamically.

The maximum sizes of different variable-length parameters such as the number of partitions that can be stored in the **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365), the maximum length of data store in the **com.rti.dds.↩ infrastructure.UserDataQosPolicy** (p. 1959) and **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1127), and many others can be changed from their defaults using this QoS policy. However, it is important that all Domain↩ Participants that need to communicate with each other use the *same set* of maximum values. Otherwise, when these parameters are propagated from one **com.rti.dds.domain.DomainParticipant** (p. 670) to another, a **com.rti.dds.↩ domain.DomainParticipant** (p. 670) with a smaller maximum length may reject the parameter, resulting in an error.

An important parameter in this QoS policy that is often changed by users is **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.type_object_max_serialized_length** (p. 817).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.87.2 Member Data Documentation

8.87.2.1 local_writer_allocation

```
final AllocationSettings_t local_writer_allocation
```

Allocation settings applied to local DataWriters.

[default] initial_count = 16; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.2 local_reader_allocation

```
final AllocationSettings_t local_reader_allocation
```

Allocation settings applied to local DataReaders.

[default] initial_count = 16; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.3 local_publisher_allocation

```
final AllocationSettings_t local_publisher_allocation
```

Allocation settings applied to local Publisher.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.4 local_subscriber_allocation

```
final AllocationSettings_t local_subscriber_allocation
```

Allocation settings applied to local Subscriber.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.5 local_topic_allocation

```
final AllocationSettings_t local_topic_allocation
```

Allocation settings applied to local Topic.

[default] initial_count = 16; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.6 remote_writer_allocation

```
final AllocationSettings_t remote_writer_allocation
```

Allocation settings applied to remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] initial_count = 64; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.7 remote_reader_allocation

```
final AllocationSettings_t remote_reader_allocation
```

Allocation settings applied to remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] initial_count = 64; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.8 remote_participant_allocation

```
final AllocationSettings_t remote_participant_allocation
```

Allocation settings applied to remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] initial_count = 16; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.9 matching_writer_reader_pair_allocation

```
final AllocationSettings_t matching_writer_reader_pair_allocation
```

Allocation settings applied to matching local writer and remote/local reader pairs.

[default] initial_count = 32; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.10 matching_reader_writer_pair_allocation

```
final AllocationSettings_t matching_reader_writer_pair_allocation
```

Allocation settings applied to matching local reader and remote/local writer pairs.

[default] initial_count = 32; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.11 ignored_entity_allocation

```
final AllocationSettings_t ignored_entity_allocation
```

Allocation settings applied to ignored entities.

[default] initial_count = 8; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.12 content_filtered_topic_allocation

```
final AllocationSettings_t content_filtered_topic_allocation
```

Allocation settings applied to content filtered topic.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.13 content_filter_allocation

```
final AllocationSettings_t content_filter_allocation
```

Allocation settings applied to content filter.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.14 read_condition_allocation

```
final AllocationSettings_t read_condition_allocation
```

Allocation settings applied to read condition pool.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.15 query_condition_allocation

```
final AllocationSettings_t query_condition_allocation
```

Allocation settings applied to query condition pool.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.16 outstanding_asynchronous_sample_allocation

```
final AllocationSettings_t outstanding_asynchronous_sample_allocation
```

Allocation settings applied to the maximum number of samples (from all **com.rti.dds.publication.DataWriter** (p. 553)) waiting to be asynchronously written.

[default] initial_count = 64; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.17 flow_controller_allocation

```
final AllocationSettings_t flow_controller_allocation
```

Allocation settings applied to flow controllers.

[default] initial_count = 4; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.18 local_writer_hash_buckets

```
int local_writer_hash_buckets
```

Hash_Buckets settings applied to local DataWriters.

[default] 4

[range] [1, 10000]

8.87.2.19 local_reader_hash_buckets

```
int local_reader_hash_buckets
```

Number of hash buckets for local DataReaders.

[default] 4

[range] [1, 10000]

8.87.2.20 local_publisher_hash_buckets

```
int local_publisher_hash_buckets
```

Number of hash buckets for local Publisher.

[default] 1

[range] [1, 10000]

8.87.2.21 local_subscriber_hash_buckets

```
int local_subscriber_hash_buckets
```

Number of hash buckets for local Subscriber.

[default] 1

[range] [1, 10000]

8.87.2.22 local_topic_hash_buckets

```
int local_topic_hash_buckets
```

Number of hash buckets for local Topic.

[default] 4

[range] [1, 10000]

8.87.2.23 remote_writer_hash_buckets

```
int remote_writer_hash_buckets
```

Number of hash buckets for remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] 16

[range] [1, 10000]

8.87.2.24 remote_reader_hash_buckets

```
int remote_reader_hash_buckets
```

Number of hash buckets for remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] 16

[range] [1, 10000]

8.87.2.25 remote_participant_hash_buckets

```
int remote_participant_hash_buckets
```

Number of hash buckets for remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] 4

[range] [1, 10000]

8.87.2.26 matching_writer_reader_pair_hash_buckets

```
int matching_writer_reader_pair_hash_buckets
```

Number of hash buckets for matching local writer and remote/local reader pairs.

[default] 32

[range] [1, 10000]

8.87.2.27 matching_reader_writer_pair_hash_buckets

```
int matching_reader_writer_pair_hash_buckets
```

Number of hash buckets for matching local reader and remote/local writer pairs.

[default] 32

[range] [1, 10000]

8.87.2.28 ignored_entity_hash_buckets

```
int ignored_entity_hash_buckets
```

Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

8.87.2.29 content_filtered_topic_hash_buckets

```
int content_filtered_topic_hash_buckets
```

Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

8.87.2.30 content_filter_hash_buckets

```
int content_filter_hash_buckets
```

Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

8.87.2.31 flow_controller_hash_buckets

```
int flow_controller_hash_buckets
```

Number of hash buckets for flow controllers.

[default] 1

[range] [1, 10000]

8.87.2.32 max_gather_destinations

```
int max_gather_destinations
```

Maximum number of destinations per RTI Connext send.

When RTI Connext sends out a message, it has the capability to send to multiple destinations to be more efficient. The maximum number of destinations per RTI Connext send is specified by max_gather_destinations.

[default] 16

[range] [16, 1 million]

8.87.2.33 participant_user_data_max_length

```
int participant_user_data_max_length
```

Maximum length of user data in **com.rti.dds.domain.DomainParticipantQos** (p. 795) and **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349).

[default] 256

[range] [0,0x7fffffff]

8.87.2.34 topic_data_max_length

```
int topic_data_max_length
```

Maximum length of topic data in **com.rti.dds.topic.TopicQos** (p. 1824), builtin.TopicBuiltinTopicData, **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762).

[default] 256

[range] [0,0x7fffffff]

8.87.2.35 publisher_group_data_max_length

```
int publisher_group_data_max_length
```

Maximum length of group data in **com.rti.dds.publication.PublisherQos** (p. 1490) and **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).

[default] 256

[range] [0,0x7fffffff]

8.87.2.36 subscriber_group_data_max_length

```
int subscriber_group_data_max_length
```

Maximum length of group data in **com.rti.dds.subscription.SubscriberQos** (p. 1756) and **com.rti.dds.↩ subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762).

[default] 256

[range] [0,0x7fffffff]

8.87.2.37 writer_user_data_max_length

```
int writer_user_data_max_length
```

Maximum length of user data in **com.rti.dds.publication.DataWriterQos** (p. 612) and **com.rti.dds.publication.↩ builtin.PublicationBuiltinTopicData** (p. 1452).

[default] 256

[range] [0,0x7fffffff]

8.87.2.38 reader_user_data_max_length

```
int reader_user_data_max_length
```

Maximum length of user data in **com.rti.dds.subscription.DataReaderQos** (p. 517) and **com.rti.dds.subscription.↩ builtin.SubscriptionBuiltinTopicData** (p. 1762).

[default] 256

[range] [0,0x7fffffff]

8.87.2.39 max_partitions

```
int max_partitions
```

Maximum number of partition name strings allowable in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365).

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 64.

[default] 64

[range] [0,64]

8.87.2.40 max_partition_cumulative_characters

```
int max_partition_cumulative_characters
```

Maximum number of combined characters allowable in all partition names in a **com.rti.dds.infrastructure.Partition↩ QosPolicy** (p. 1365).

The maximum number of combined characters should account for a terminating NULL ('\0') character for each partition name string.

This setting is made on a per DomainParticipant basis; it cannot be set individually on a per Publisher/Subscriber basis. However, the limit is enforced and applies per Publisher/Subscriber.

This value cannot exceed 256.

[default] 256

[range] [0,256]

8.87.2.41 type_code_max_serialized_length

```
int type_code_max_serialized_length
```

Maximum size of serialized string for type code.

This parameter is an alternative to **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type↩ _object_max_serialized_length** (p. 817) for limiting the size of the type code that a **com.rti.dds.domain.Domain↩ Participant** (p. 670) is able to store and propagate for user data types. Type codes can be used by external applications to understand user data types without having the data type predefined in compiled form. However, since type codes contain all of the information of a data structure, including the strings that define the names of the members of a structure, complex data structures can result in large type codes. So it is common for users to set this parameter to a large value, if used (by default, it is set to 0). However, as with all parameters in this QoS policy defining maximum sizes for variable-length elements, all DomainParticipants in the same domain should use the same value for this parameter. Note: TypeObject is now the standard method of exchanging type information in RTI Connext. It is recommended to use **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_serialized_length** (p. 817) to configure the maximum serialized type object string.

[default] 0

[range] [0,0xffff]

8.87.2.42 type_object_max_serialized_length

```
int type_object_max_serialized_length
```

The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.

This parameter limits the size of the TypeObject that a DomainParticipant is able to propagate. Since TypeObjects contain all of the information of a data structure, including the strings that define the names of the members of a structure, complex data structures can result in TypeObjects larger than the default maximum of 8192 bytes. This field allows you to specify a larger value. The desired size for a given **com.rti.dds.typecode.TypeCode** (p. 1873) can be obtained using **com.rti.dds.typecode.TypeCode.get_type_object_serialized_size** (p. 1913).

[default] 8192

[range] [0,0x7fffffff]

8.87.2.43 serialized_type_object_dynamic_allocation_threshold

```
int serialized_type_object_dynamic_allocation_threshold
```

A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.

Above this threshold, the memory for a TypeObject is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

In case **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_serialized_length** (p. 817) is different than **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) and is smaller than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.serialized_type_object_dynamic_allocation_threshold** (p. 817): **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.serialized_type_object_dynamic_allocation_threshold** (p. 817) will be adjusted to **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_serialized_length** (p. 817).

[default] 8192

[range] [0,0x7fffffff] <= **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_serialized_length** (p. 817)

8.87.2.44 type_object_max_deserialized_length

```
int type_object_max_deserialized_length
```

The maximum number of bytes that a deserialized TypeObject can consume.

This parameter limits the size of the TypeObject that a DomainParticipant is able to store.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [0,0x7fffffff] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.87.2.45 deserialized_type_object_dynamic_allocation_threshold

```
int deserialized_type_object_dynamic_allocation_threshold
```

A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject.

Above this threshold, the memory for a TypeObject is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers. The size of the buffers is equal to this threshold.

In case **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_deserialized_length** (p. 818) is different than **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) and is smaller than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.deserialized_type_object_dynamic_allocation_threshold** (p. 818): **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.deserialized_type_object_dynamic_allocation_threshold** (p. 818) will be adjusted to **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_deserialized_length** (p. 818).

[default] 4096

[range] [0,0x7fffffff] <= **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_object_max_deserialized_length** (p. 818)

8.87.2.46 contentfilter_property_max_length

```
int contentfilter_property_max_length
```

This field is the maximum length of all data related to a Content-filtered topic.

This is the sum of the length of the ContentFilteredTopic name, the length of the related topic name, the length of the filter expression, the length of the filter parameters, and the length of the filter name. The maximum number of combined characters should account for a terminating NULL ('\0') character for each string.

[default] 256

[range] [0,0xffff]

8.87.2.47 channel_seq_max_length

```
int channel_seq_max_length
```

Maximum number of channels that can be specified in **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) for MultiChannel DataWriters.

[default] 32

[range] [0,0xffff]

8.87.2.48 channel_filter_expression_max_length

```
int channel_filter_expression_max_length
```

Maximum length of a channel **com.rti.dds.infrastructure.ChannelSettings_t.filter_expression** (p. 414) in a Multi↩ Channel DataWriter.

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

8.87.2.49 participant_property_list_max_length

```
int participant_property_list_max_length
```

Maximum number of properties associated with the **com.rti.dds.domain.DomainParticipant** (p. 670).

[default] 32

[range] [0,0xffff]

8.87.2.50 participant_property_string_max_length

```
int participant_property_string_max_length
```

Maximum string length of the properties associated with the **com.rti.dds.domain.DomainParticipant** (p. 670).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **com.rti.dds.domain.DomainParticipant** (p. 670) properties.

[default] 4096

[range] [0,0xffff]

8.87.2.51 writer_property_list_max_length

```
int writer_property_list_max_length
```

Maximum number of properties associated with a **com.rti.dds.publication.DataWriter** (p. 553).

[range] [0,0xffff]

[default] 32

8.87.2.52 writer_property_string_max_length

```
int writer_property_string_max_length
```

Maximum string length of the properties associated with a **com.rti.dds.publication.DataWriter** (p. 553).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **com.rti.dds.publication.DataWriter** (p. 553) properties.

[default] 1024

[range] [0,0xffff]

8.87.2.53 reader_property_list_max_length

```
int reader_property_list_max_length
```

Maximum number of properties associated with a **com.rti.dds.subscription.DataReader** (p. 450).

[default] 32

[range] [0,0xffff]

8.87.2.54 reader_property_string_max_length

```
int reader_property_string_max_length
```

Maximum string length of the properties associated with a **com.rti.dds.subscription.DataReader** (p. 450).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with a **com.rti.dds.subscription.DataReader** (p. 450) properties.

[default] 1024

[range] [0,0xffff]

8.87.2.55 max_endpoint_groups

```
int max_endpoint_groups
```

Maximum number of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022) allowable in a **com.rti.dds.↩ infrastructure.AvailabilityQosPolicy** (p. 343).

[default] 32

[range] [0,65535]

8.87.2.56 max_endpoint_group_cumulative_characters

```
int max_endpoint_group_cumulative_characters
```

Maximum number of combined role_name characters allowed in all **com.rti.dds.infrastructure.EndpointGroup_↩ t** (p. 1022) in a **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343).

The maximum number of combined characters should account for a terminating NULL character for each role_name string.

[default] 1024

[range] [0,65535]

8.87.2.57 transport_info_list_max_length

```
int transport_info_list_max_length
```

Maximum number of installed transports to send and receive information about in builtin.ParticipantBuiltinTopicData.↩ transport_info.

[default] 12

[range] [0,100]

8.87.2.58 ignored_entity_replacement_kind

`DomainParticipantResourceLimitsIgnoredEntityReplacementKind ignored_entity_replacement_kind`

Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.ignored_entity_allocation** (p. 810) are reached.

When a **com.rti.dds.domain.DomainParticipant** (p. 670)'s number of ignored entities is greater than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.ignored_entity_allocation** (p. 810), the **com.rti.dds.domain.DomainParticipant** (p. 670) will try to make room by replacing an existing ignored participant entry. This field specifies what entity is allowed to be replaced.

If a replaceable participant entry is not available, an out-of-resources exception will be returned.

[default] com.rti.dds.infrastructure.DomainParticipantResourceLimitsIgnoredEntityReplacementKind.DomainParticipantResourceLimitsIgnoredEntityReplacementKind.NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT

See also

com.rti.dds.infrastructure.DomainParticipantResourceLimitsIgnoredEntityReplacementKind (p. 802)

8.87.2.59 remote_topic_query_allocation

`final AllocationSettings_t remote_topic_query_allocation`

Allocation settings applied to remote TopicQueries.

Settings applied to the allocation of information about **com.rti.dds.subscription.TopicQuery** (p. 1830) objects created by other participants and discovered by this participant.

When the participant receives a new topic query that would make the current count go above `max_count`, it is not processed until the current count drops (i.e. another topic query is cancelled). The topic query stays in the Built-in ServiceRequest DataReader queue until it can be processed or it is cancelled.

[default] initial_count = 1; max_count = **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259); incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 336)

8.87.2.60 remote_topic_query_hash_buckets

`int remote_topic_query_hash_buckets`

Number of hash buckets for remote TopicQueries.

[default] 1

[range] [1, 10000]

See also

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.remote_topic_query_allocation (p. 822)

8.87.2.61 writer_data_tag_list_max_length

```
int writer_data_tag_list_max_length
```

Maximum number of data tags associated with a **com.rti.dds.publication.DataWriter** (p. 553).

[default] 0

[range] [0,0xffff]

8.87.2.62 writer_data_tag_string_max_length

```
int writer_data_tag_string_max_length
```

Maximum string length of the data tags associated with a **com.rti.dds.publication.DataWriter** (p. 553).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **com.rti.dds.publication.DataWriter** (p. 553) data tags.

[default] 0

[range] [0,0xffff]

8.87.2.63 reader_data_tag_list_max_length

```
int reader_data_tag_list_max_length
```

Maximum number of data tags associated with a **com.rti.dds.subscription.DataReader** (p. 450).

[default] 0

[range] [0,0xffff]

8.87.2.64 reader_data_tag_string_max_length

```
int reader_data_tag_string_max_length
```

Maximum string length of the data tags associated with a **com.rti.dds.subscription.DataReader** (p. 450).

The string length is defined as the cumulative length in bytes, including the null terminating characters, of all the pairs (name, value) associated with the **com.rti.dds.subscription.DataReader** (p. 450) data tags.

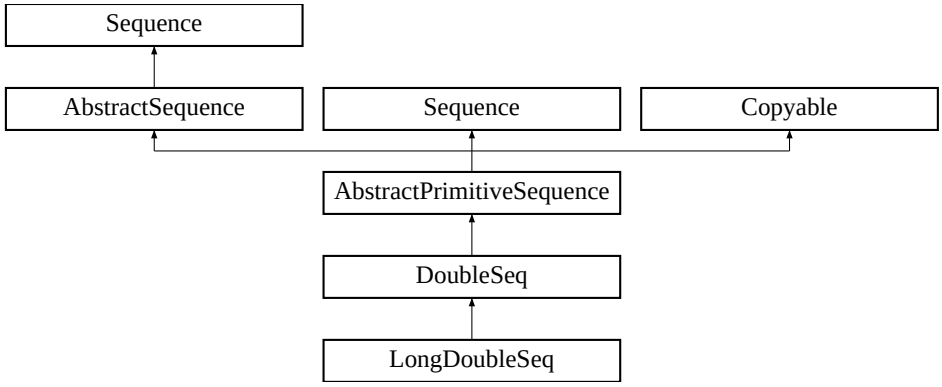
[default] 0

[range] [0,0xffff]

8.88 DoubleSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < double >

Inheritance diagram for DoubleSeq:



Public Member Functions

- **DoubleSeq** ()

  Constructs an empty sequence of doubles with an initial maximum of zero.
- **DoubleSeq** (int initialMaximum)

  Constructs an empty sequence of doubles with the given initial maximum.
- **DoubleSeq** (double[ ] doubles)

  Constructs a new sequence containing the given doubles.
- final boolean **addAllDouble** (double[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- final boolean **addAllDouble** (double[ ] elements)
- final void **addDouble** (double element)

  Append the element to the end of the sequence.
- final void **addDouble** (int index, double element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- final double **getDouble** (int index)

  Returns the double at the given index.
- final double **setDouble** (int index, double element)

  Set the new double at the given index and return the old double.
- final void **setDouble** (int dstIndex, double[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- final double[ ] **toArrayDouble** (double[ ] array)

  Return an array containing copy of the contents of this sequence.
- final int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- final Object **get** (int index)

  *A wrapper for **getDouble(int)** (p. 827) that returns a java.lang.Double.*
- final Object **set** (int index, Object element)

  *A wrapper for **setDouble()** (p. 827).*
- final void **add** (int index, Object element)

  A wrapper for addDouble(int, int).

8.88.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < double >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

double

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.88.2 Constructor & Destructor Documentation

8.88.2.1 DoubleSeq() [1/3]

```
DoubleSeq ( )
```

Constructs an empty sequence of doubles with an initial maximum of zero.

8.88.2.2 DoubleSeq() [2/3]

```
DoubleSeq (
            int initialMaximum )
```

Constructs an empty sequence of doubles with the given initial maximum.

8.88.2.3 DoubleSeq() [3/3]

```
DoubleSeq (
            double[] doubles )
```

Constructs a new sequence containing the given doubles.

Parameters

doubles	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **DoubleSeq.addAllDouble()**.

8.88.3 Member Function Documentation

8.88.3.1 addAllDouble() [1/2]

```
final boolean addAllDouble (
            double[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **DoubleSeq.addAllDouble()**, and **DoubleSeq.DoubleSeq()**.

8.88.3.2 addAllDouble() [2/2]

```
final boolean addAllDouble (
            double[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **DoubleSeq.addAllDouble()**.

8.88.3.3 addDouble() [1/2]

```
final void addDouble (
            double element )
```

Append the element to the end of the sequence.

Referenced by **DoubleSeq.add()**.

8.88.3.4 addDouble() [2/2]

```
final void addDouble (
             int index,
             double element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.88.3.5 getDouble()

```
final double getDouble (
             int index )
```

Returns the double at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **DoubleSeq.get()**.

8.88.3.6 setDouble() [1/2]

```
final double setDouble (
             int index,
             double element )
```

Set the new double at the given index and return the old double.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **DoubleSeq.set()**.

8.88.3.7 setDouble() [2/2]

```
final void setDouble (
            int dstIndex,
            double[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.88.3.8 toArrayDouble()

```
final double[] toArrayDouble (
            double[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.88.3.9 getMaximum()

```
final int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 830), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_double_seq()**, and **DynamicData.set_double_seq()**.

8.88.3.10 get()

```
final Object get (
             int index )
```

A wrapper for **getDouble(int)** (p. 827) that returns a java.lang.Double.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **DoubleSeq.getDouble()**.

8.88.3.11 set()

```
final Object set (
             int index,
             Object element )
```

A wrapper for **setDouble()** (p. 827).

Exceptions

ClassCastException	if the element is not of type Double.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **DoubleSeq.setDouble()**.

8.88.3.12 add()

```
final void add (
             int index,
             Object element )
```

A wrapper for addDouble(int, int).

Exceptions

ClassCastException	if the element is not of type Double.

See also

java.util.List::add(int, java.lang.Object)

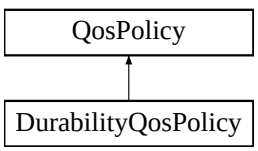
Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **DoubleSeq.addDouble()**.

8.89 DurabilityQosPolicy Class Reference

This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.

Inheritance diagram for DurabilityQosPolicy:



Public Attributes

- **DurabilityQosPolicyKind kind**

  The kind of durability.
- boolean **direct_communication**

  *<<**extension**>> (p. 155) Indicates whether or not a TRANSIENT or PERSISTENT **com.rti.dds.subscription.Data↩ Reader** (p. 450) should receive samples directly from a TRANSIENT or PERSISTENT **com.rti.dds.publication.Data↩ Writer** (p. 553)*
- int **writer_depth**

  *<<**extension**>> (p. 155) Indicates the number of samples per instance that a durable **com.rti.dds.publication.Data↩ Writer** (p. 553) will send to a late-joining **com.rti.dds.subscription.DataReader** (p. 450).*
- **PersistentStorageSettings storage_settings**

  Configures durable writer history and durable reader state.

Static Public Attributes

- static final int **AUTO_WRITER_DEPTH**

  *A special value used as the default value for **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834).*

8.89.1 Detailed Description

This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **com.rti.dds.subscription.DataReader** (p. 450) entities that join the network later.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

See also

DURABILITY_SERVICE (p. 232)

8.89.2 Usage

It is possible for a **com.rti.dds.publication.DataWriter** (p. 553) to start publishing data before all (or any) **com.rti.↩ dds.subscription.DataReader** (p. 450) entities have joined the network.

Moreover, a **com.rti.dds.subscription.DataReader** (p. 450) that joins the network after some data has been written could potentially be interested in accessing the most current values of the data, as well as potentially some history.

This policy makes it possible for a late-joining **com.rti.dds.subscription.DataReader** (p. 450) to obtain previously published samples.

By helping to ensure that DataReaders get all data that was sent by DataWriters, regardless of when it was sent, using this QoS policy can increase system tolerance to failure conditions.

Note that although related, this does not strictly control what data RTI Connext will maintain internally. That is, RTI Connext may choose to maintain some data for its own purposes (e.g., flow control) and yet not make it available to late-joining readers if the **DURABILITY** (p. 230) policy is set to com.rti.dds.infrastructure.DurabilityQosPolicyKind.↩ DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS.

8.89.2.1 Transient and Persistent Durability

For the purpose of implementing the DURABILITY QoS kind TRANSIENT or PERSISTENT, RTI Connext behaves *as if* for each Topic that has **com.rti.dds.infrastructure.DurabilityQosPolicy.kind** (p. 833) of com.rti.dds.infrastructure.↩ DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS or com.rti.dds.infrastructure.↩ DurabilityQosPolicyKind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS there is a corresponding "built-in" **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.publication.DataWriter** (p. 553) configured with the same DURABILITY kind. In other words, it is *as if* somewhere in the system, independent of the original **com.rti.dds.↩ publication.DataWriter** (p. 553), there is a built-in durable **com.rti.dds.subscription.DataReader** (p. 450) subscribing to that Topic and a built-in durable DataWriter re-publishing it as needed for the new subscribers that join the system. This functionality is provided by the *RTI Persistence Service*.

The Persistence Service can configure itself based on the QoS of your application's **com.rti.dds.publication.Data↩ Writer** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities. For each transient or persistent **com.↩ rti.dds.topic.Topic** (p. 1807), the built-in fictitious Persistence Service **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.publication.DataWriter** (p. 553) have their QoS configured from the QoS of your application's **com.↩ rti.dds.publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities that communicate on that **com.rti.dds.topic.Topic** (p. 1807).

For a given **com.rti.dds.topic.Topic** (p. 1807), the usual request/offered semantics apply to the matching between any **com.rti.dds.publication.DataWriter** (p. 553) in the domain that writes the **com.rti.dds.topic.Topic** (p. 1807) and the built-in transient/persistent **com.rti.dds.subscription.DataReader** (p. 450) for that **com.rti.dds.topic.Topic** (p. 1807); similarly for the built-in transient/persistent **com.rti.dds.publication.DataWriter** (p. 553) for a **com.rti.dds.topic.Topic** (p. 1807) and any **com.rti.dds.subscription.DataReader** (p. 450) for the **com.rti.dds.topic.Topic** (p. 1807). As a consequence, a **com.rti.dds.publication.DataWriter** (p. 553) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **com.rti.dds.subscription.DataReader** (p. 450) that has an incompatible QoS will not get data from it.

Incompatibilities between local **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.publication.Data↩ Writer** (p. 553) entities and the corresponding fictitious built-in transient/persistent entities cause the com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS and com.rti.dds.infrastructure.↩ StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS to change and the corresponding **Listener** (p. 1236) invocations and/or signaling of **com.rti.dds.infrastructure.Condition** (p. 429) objects as they would with your application's own entities.

The value of **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 839) controls when *RTI Persistence Service* is able to remove all information regarding a data instances.

Information on a data instance is maintained until the following conditions are met:

1. The instance has been explicitly disposed (instance_state = NOT_ALIVE_DISPOSED),

and

1. All samples for the disposed instance have been acknowledged, including the dispose sample itself,

and

1. A time interval longer that **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 839) has elapsed since the moment RTI Connext detected that the previous two conditions were met. (Note: Only values of zero or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) are currently supported for the service_cleanup_delay)

The utility of **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 839) is apparent in the situation where an application disposes an instance and it crashes before it has a chance to complete additional tasks related to the disposition. Upon restart, the application may ask for initial data to regain its state and the delay introduced by the service_cleanup_delay will allow the restarted application to receive the information on the disposed instance and complete the interrupted tasks.

8.89.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind* >= *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of DURABILITY kind are considered ordered such that com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.↩ VOLATILE_DURABILITY_QOS < com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.↩ TRANSIENT_LOCAL_DURABILITY_QOS < com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicy↩ Kind.TRANSIENT_DURABILITY_QOS < com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicy↩ Kind.PERSISTENT_DURABILITY_QOS.

8.89.4 Member Data Documentation

8.89.4.1 kind

DurabilityQosPolicyKind kind

The kind of durability.

[default] com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS

8.89.4.2 direct_communication

```
boolean direct_communication
```

<<***extension***>> *(p. 155)* Indicates whether or not a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 450) should receive samples directly from a TRANSIENT or PERSISTENT **com.rti.dds.publication.DataWriter** (p. 553)

When direct_communication is set to com.rti.dds.infrastructure.true, a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 450) will receive samples from both the original **com.rti.dds.publication.DataWriter** (p. 553) configured with TRANSIENT or PERSISTENT durability and the **com.rti.dds.publication.DataWriter** (p. 553) created by the persistence service. This peer-to-peer communication pattern provides low latency between end-points.

If the same sample is received from the original **com.rti.dds.publication.DataWriter** (p. 553) and the persistence service, the middleware will discard the duplicate.

When direct_communication is set to com.rti.dds.infrastructure.false, a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 450) will only receive samples from the **com.rti.dds.publication.DataWriter** (p. 553) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[default] com.rti.dds.infrastructure.true

8.89.4.3 writer_depth

```
int writer_depth
```

<<***extension***>> *(p. 155)* Indicates the number of samples per instance that a durable **com.rti.dds.publication.DataWriter** (p. 553) will send to a late-joining **com.rti.dds.subscription.DataReader** (p. 450).

The default value, **com.rti.dds.infrastructure.DurabilityQosPolicy.AUTO_WRITER_DEPTH** (p. 231), makes this parameter equal to the following:

- **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) if the history kind is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS.
- **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) in the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) if the history kind is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS.

The `writer_depth` must be <= **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147).

`writer_depth` applies only to non-volatile DataWriters (those for which the kind is TRANSIENT_LOCAL, TRANSIENT, or PERSISTENT).

When **com.rti.dds.infrastructure.BatchQosPolicy.enable** (p. 356) is set to true, `writer_depth` acts as a minimum number of samples per instance that will be sent to late joiners, as opposed to the maximum. As long as a batch contains a single sample that falls within the `writer_depth` for the instance to which it belongs, then the entire batch will be sent. This means that batches sent to late-joining DataReaders may contain more samples per instance than is specified by the `writer_depth` QoS setting.

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the `writer_depth`, not the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147).

Setting `writer_depth` on the DataReader side will be ignored.

[default] com.rti.dds.infrastructure.DurabilityQosPolicy.AUTO_WRITER_DEPTH (p. 231)

8.89.4.4 storage_settings

```
PersistentStorageSettings storage_settings
```

Initial value:

```
=
            new PersistentStorageSettings()
```

Configures durable writer history and durable reader state.

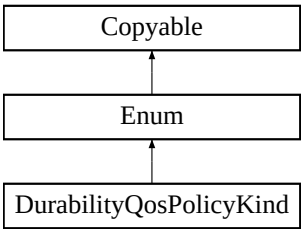
By default, durable writer history and durable reader state are disabled. This means that a DataWriter will not persist its historical cache and a DataReader will not persist its state.

To enable durable writer history and durable reader state set **com.rti.dds.infrastructure.PersistentStorage↩ Settings.enable** (p. 1373) to com.rti.dds.infrastructure.true.

8.90 DurabilityQosPolicyKind Class Reference

Kinds of durability.

Inheritance diagram for DurabilityQosPolicyKind:



Static Public Attributes

- static final **DurabilityQosPolicyKind VOLATILE_DURABILITY_QOS**

  [default] *RTI Connext does not need to keep any samples of data instances on behalf of any* ***com.rti.dds.subscription.↩ DataReader*** *(p. 450) that is unknown by the* ***com.rti.dds.publication.DataWriter*** *(p. 553) at the time the instance is written.*
- static final **DurabilityQosPolicyKind TRANSIENT_LOCAL_DURABILITY_QOS**

  RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining ***com.rti.dds.↩ subscription.DataReader*** *(p. 450).*
- static final **DurabilityQosPolicyKind TRANSIENT_DURABILITY_QOS**

  RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining ***com.rti.dds.↩ subscription.DataReader*** *(p. 450).*
- static final **DurabilityQosPolicyKind PERSISTENT_DURABILITY_QOS**

  Data is kept on permanent storage, so that they can outlive a system session.

Additional Inherited Members

8.90.1 Detailed Description

Kinds of durability.

QoS:

com.rti.dds.infrastructure.DurabilityQosPolicy (p. 830)

8.90.2 Member Data Documentation

8.90.2.1 VOLATILE_DURABILITY_QOS

```
final DurabilityQosPolicyKind VOLATILE_DURABILITY_QOS [static]
```

[default] RTI Connext does not need to keep any samples of data instances on behalf of any **com.rti.dds.↩ subscription.DataReader** (p. 450) that is unknown by the **com.rti.dds.publication.DataWriter** (p. 553) at the time the instance is written.

In other words, RTI Connext will only attempt to provide the data to existing subscribers.

This option does not require RTI Persistence Service.

8.90.2.2 TRANSIENT_LOCAL_DURABILITY_QOS

```
final DurabilityQosPolicyKind TRANSIENT_LOCAL_DURABILITY_QOS [static]
```

RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **com.rti.↩ dds.subscription.DataReader** (p. 450).

Which particular samples are kept depends on other QoS such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590). RTI Connext is only required to keep the data in memory of the **com.rti.dds.publication.DataWriter** (p. 553) that wrote the data.

Data is not required to survive the **com.rti.dds.publication.DataWriter** (p. 553).

For this setting to be effective, you must also set the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).

This option does not require RTI Persistence Service.

8.90.2.3 TRANSIENT_DURABILITY_QOS

```
final DurabilityQosPolicyKind TRANSIENT_DURABILITY_QOS [static]
```

RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining **com.rti.dds.subscription.DataReader** (p. 450).

Which particular samples are kept depends on other QoS such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590). RTI Connext is only required to keep the data in memory and not in permanent storage.

Data is not tied to the lifecycle of the **com.rti.dds.publication.DataWriter** (p. 553).

Data will survive the **com.rti.dds.publication.DataWriter** (p. 553).

This option requires RTI Persistence Service.

8.90.2.4 PERSISTENT_DURABILITY_QOS

```
final DurabilityQosPolicyKind PERSISTENT_DURABILITY_QOS [static]
```

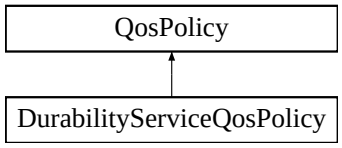
Data is kept on permanent storage, so that they can outlive a system session.

This option requires RTI Persistence Service.

8.91 DurabilityServiceQosPolicy Class Reference

Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.

Inheritance diagram for DurabilityServiceQosPolicy:



Public Attributes

- final **Duration_t service_cleanup_delay**

  Controls when the service is able to remove all information regarding a data instances.
- **HistoryQosPolicyKind history_kind**

  The kind of history to apply in recouping durable data.
- int **history_depth**

  *Setting to use for the **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834) when recouping durable data.*
- int **max_samples**

  Part of resource limits QoS policy to apply when feeding a late joiner.
- int **max_instances**

  Part of resource limits QoS policy to apply when feeding a late joiner.
- int **max_samples_per_instance**

  Part of resource limits QoS policy to apply when feeding a late joiner.

8.91.1 Detailed Description

Various settings to configure the external *RTI Persistence Service* used by RTI Connext for DataWriters with a **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830) setting of com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.DurabilityQosPolicy↩ Kind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

See also

DURABILITY (p. 230)

HISTORY (p. 237)

RESOURCE_LIMITS (p. 259)

8.91.2 Usage

When a DataWriter's **com.rti.dds.infrastructure.DurabilityQosPolicy.kind** (p. 833) is com.rti.dds.infrastructure.↩ DurabilityQosPolicyKind.DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or com.rti.dds.infrastructure.↩ DurabilityQosPolicyKind.DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS, an external service, the *RTI Persistence Service*, is used to store and possibly forward the data sent by the **com.rti.dds.publication.DataWriter** (p. 553) to **com.rti.dds.subscription.DataReader** (p. 450) objects that are created *after* the data was initially sent.

This QoS policy is used to configure certain parameters of the Persistence Service when it operates on the behalf of the **com.rti.dds.publication.DataWriter** (p. 553), such as how much data to store. For example, it configures the **DURABILITY** (p. 230), **HISTORY** (p. 237), and the **RESOURCE_LIMITS** (p. 259) used by the fictitious DataReader and DataWriter used by the Persistence Service. Note, however, that the Persistence Service itself may be configured to ignore these values and instead use values from its own configuration file.

8.91.3 Member Data Documentation

8.91.3.1 service_cleanup_delay

```
final  Duration_t service_cleanup_delay
```

Initial value:

```
= new Duration_t(
        Duration_t.DURATION_INFINITE_SEC, Duration_t.DURATION_INFINITE_NSEC)
```

Controls when the service is able to remove all information regarding a data instances.

When the service cleanup delay is set to 0, disposed instances will be completely removed from the service. Only values of 0 and **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) are currently supported.

[default] 0

8.91.3.2 history_kind

```
 HistoryQosPolicyKind history_kind
```

The kind of history to apply in recouping durable data.

[default] com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS

8.91.3.3 history_depth

```
int history_depth
```

Setting to use for the **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834) when recouping durable data.

If the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) is set to a value lower than this value, **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) will be set equal to the value of this field.

[default] com.rti.dds.infrastructure.DurabilityQosPolicy.AUTO_WRITER_DEPTH (p. 231) (1)

8.91.3.4 max_samples

```
int max_samples
```

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

8.91.3.5 max_instances

```
int max_instances
```

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

8.91.3.6 max_samples_per_instance

```
int max_samples_per_instance
```

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

8.92 Duration_t Class Reference

Type for *duration* representation.

Inherits Struct, and Externalizable.

Public Member Functions

- **Duration_t** (**Duration_t** duration)
  Copy constructor.
- **Duration_t** (int **sec**, int **nanosec**)
- boolean **is_zero** ()
- boolean **is_infinite** ()
- boolean **is_auto** ()
- **Duration_t** **add** (**Duration_t** other)
- **Duration_t** **subtract** (**Duration_t** other)

Static Public Member Functions

- static **Duration_t** **from_micros** (long micros)
  Creates a new duration object from a duration expressed in microseconds.
- static **Duration_t** **from_nanos** (long nanos)
  Creates a new duration object from a duration expressed in nanoseconds.
- static **Duration_t** **from_millis** (long millis)
  Creates a new duration object from a duration expressed in milliseconds.
- static **Duration_t** **from_seconds** (int seconds)
  Creates a new duration object from a duration expressed in seconds.

Public Attributes

- int **sec**

  seconds
- int **nanosec**

  nanoseconds

Static Public Attributes

- static final int **DURATION_ZERO_SEC**

  A zero-length second period of time.
- static final int **DURATION_ZERO_NSEC**

  A zero-length nano-second period of time.
- static final int **DURATION_INFINITE_SEC**

  An infinite second period of time.
- static final int **DURATION_INFINITE_NSEC**

  An infinite nano-second period of time.
- static final int **DURATION_AUTO_SEC**

  An auto second period of time.
- static final int **DURATION_AUTO_NSEC**

  An auto nano-second period of time.
- static final **Duration_t DURATION_ZERO**

  A zero-length period of time.
- static final **Duration_t DURATION_INFINITE**

  An infinite period of time.
- static final **Duration_t DURATION_AUTO**

  Duration is automatically assigned.

8.92.1 Detailed Description

Type for *duration* representation.

Represents a time interval.

8.92.2 Constructor & Destructor Documentation

8.92.2.1 Duration_t() [1/2]

```
Duration_t (
            Duration_t duration )
```

Copy constructor.

Parameters

duration	The duration instance to copy. It must not be null.

References **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.2.2 Duration_t() [2/2]

```
Duration_t (
            int sec,
            int nanosec )
```

Parameters

sec	must be >=0
nanosec	must be >=0 and < 1000000000

References **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.3 Member Function Documentation

8.92.3.1 from_micros()

```
static Duration_t from_micros (
            long micros )  [static]
```

Creates a new duration object from a duration expressed in microseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

References **Duration_t.DURATION_INFINITE**.

8.92.3.2 from_nanos()

```
static Duration_t from_nanos (
            long nanos )  [static]
```

Creates a new duration object from a duration expressed in nanoseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

References **Duration_t.DURATION_INFINITE**.

8.92.3.3 from_millis()

```
static Duration_t from_millis (
            long millis ) [static]
```

Creates a new duration object from a duration expressed in milliseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

References **Duration_t.DURATION_INFINITE**.

8.92.3.4 from_seconds()

```
static Duration_t from_seconds (
            int seconds ) [static]
```

Creates a new duration object from a duration expressed in seconds.

8.92.3.5 is_zero()

```
boolean is_zero ( )
```

Returns

com.rti.dds.infrastructure.true if the given duration is of zero length.

References **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.3.6 is_infinite()

```
boolean is_infinite ( )
```

Returns

com.rti.dds.infrastructure.true if the given duration is of infinite length.

References **Duration_t.DURATION_INFINITE_NSEC**, **Duration_t.DURATION_INFINITE_SEC**, **Duration_t.↩ nanosec**, and **Duration_t.sec**.

Referenced by **Duration_t.add()**, and **Duration_t.subtract()**.

8.92.3.7 is_auto()

```
boolean is_auto ( )
```

Returns

com.rti.dds.infrastructure.true if the given duration has auto value.

References **Duration_t.DURATION_AUTO_NSEC**, **Duration_t.DURATION_AUTO_SEC**, **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.3.8 add()

```
Duration_t add (
             Duration_t other )
```

Calculates a duration as the result of adding other and this.

Special case:

- d.add(**Duration_t.DURATION_INFINITE** (p. 846)).**is_infinite()** (p. 843)

References **Duration_t.DURATION_INFINITE**, **Duration_t.DURATION_INFINITE_SEC**, **Duration_t.is_infinite()**, **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.3.9 subtract()

```
Duration_t subtract (
             Duration_t other )
```

Calculates a duration as the result of substracting other to this.

Special cases:

- d.substract(**Duration_t.DURATION_INFINITE** (p. 846)).**is_zero()** (p. 843)
- Duration_t.DURATION_INFINITE.substract(d).**is_infinite()** (p. 843)
- Duration_t.DURATION_INFINITE.substract(**Duration_t.DURATION_INFINITE** (p. 846)).**is_infinite()** (p. 843)
- d1.substract(d2).**is_zero()** (p. 843) if d2 is greater or equal than d1

References **Duration_t.DURATION_ZERO**, **Duration_t.is_infinite()**, **Duration_t.nanosec**, and **Duration_t.sec**.

8.92.4 Member Data Documentation

8.92.4.1 DURATION_ZERO_SEC

```
final int DURATION_ZERO_SEC  [static]
```

A zero-length second period of time.

8.92.4.2 DURATION_ZERO_NSEC

```
final int DURATION_ZERO_NSEC  [static]
```

A zero-length nano-second period of time.

8.92.4.3 DURATION_INFINITE_SEC

```
final int DURATION_INFINITE_SEC  [static]
```

An infinite second period of time.

Referenced by **Duration_t.add()**, and **Duration_t.is_infinite()**.

8.92.4.4 DURATION_INFINITE_NSEC

```
final int DURATION_INFINITE_NSEC  [static]
```

An infinite nano-second period of time.

Referenced by **Duration_t.is_infinite()**.

8.92.4.5 DURATION_AUTO_SEC

```
final int DURATION_AUTO_SEC  [static]
```

An auto second period of time.

Referenced by **Duration_t.is_auto()**.

8.92.4.6 DURATION_AUTO_NSEC

```
final int DURATION_AUTO_NSEC [static]
```

An auto nano-second period of time.

Referenced by **Duration_t.is_auto()**.

8.92.4.7 DURATION_ZERO

```
final Duration_t DURATION_ZERO [static]
```

A zero-length period of time.

Referenced by **Duration_t.subtract()**.

8.92.4.8 DURATION_INFINITE

```
final Duration_t DURATION_INFINITE [static]
```

An infinite period of time.

Referenced by **Duration_t.add()**, **Duration_t.from_micros()**, **Duration_t.from_millis()**, and **Duration_t.from_↩ nanos()**.

8.92.4.9 DURATION_AUTO

```
final Duration_t DURATION_AUTO [static]
```

Duration is automatically assigned.

8.92.4.10 sec

```
int sec
```

seconds

Referenced by **Duration_t.add()**, **Duration_t.Duration_t()**, **Duration_t.is_auto()**, **Duration_t.is_infinite()**, **Duration_t.is_zero()**, **Duration_t.subtract()**, and **WaitSet.wait()**.

8.92.4.11 nanosec

```
int nanosec
```

nanoseconds

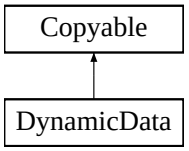
[range] [0,1000000000)

Referenced by **Duration_t.add()**, **Duration_t.Duration_t()**, **Duration_t.is_auto()**, **Duration_t.is_infinite()**, **Duration_t.is_zero()**, **Duration_t.subtract()**, and **WaitSet.wait()**.

8.93 DynamicData Class Reference

A sample of any complex data type, which can be inspected and manipulated reflectively.

Inheritance diagram for DynamicData:



Public Member Functions

- void **delete** ()

  *Finalize and deallocate this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*
- boolean **equals** (Object o)

  *Indicate whether the contents of another **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample are the same as those of this one.*
- Object **copy_from** (Object src)

  Deeply copy from the given object to this object.
- void **clear_all_members** ()

  Clear the contents of all data members of this object.
- void **clear_member** (String member_name, int member_id)

  Clear the contents of a single data member of this object.
- void **clear_optional_member** (String member_name, int member_id)

  Clear the contents of a single optional data member of this object.
- void **print** (File fp, int indent)

  Output a textual representation of this object and its contents to the given file.
- void **get_info** (**DynamicDataInfo** info_out)

  *Fill in the given descriptor with information about this **com.rti.dds.dynamicdata.DynamicData** (p. 847).*
- void **bind_type** (**TypeCode** type)

  *If this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object is not yet associated with a data type, set that type now to the given **com.rti.dds.typecode.TypeCode** (p. 1873).*
- void **unbind_type** ()

*Dissociate this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object from any particular data type.*

- void **bind_complex_member** (**DynamicData** value_out, String member_name, int member_id)

  *Use another **com.rti.dds.dynamicdata.DynamicData** (p. 847) object to provide access to a complex field of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*

- void **unbind_complex_member** (**DynamicData** value)

  *Tear down the association created by a **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) operation, committing any changes to the outer object since then.*

- synchronized **TypeCode get_type** ()

  *Get the data type, of which this **com.rti.dds.dynamicdata.DynamicData** (p. 847) represents an instance.*

- **TCKind get_type_kind** ()

  Get the kind of this object's data type.

- int **get_member_count** ()

  Get the number of members in the type.

- boolean **member_exists** (String member_name, int member_id)

  Indicates whether a member exists in this sample.

- boolean **member_exists_in_type** (String member_name, int member_id)

  Indicates whether a member of a particular name/ID exists in this data sample's type.

- void **get_member_info** (**DynamicDataMemberInfo** info, String member_name, int member_id)

  *Fill in the given descriptor with information about the identified member of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*

- void **get_member_info_by_index** (**DynamicDataMemberInfo** info, int index)

  *Fill in the given descriptor with information about the identified member of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*

- **TypeCode get_member_type** (String member_name, int member_id)

  Get the type of the given member of this sample.

- boolean **is_member_key** (String member_name, int member_id)

  Indicates whether a given member forms part of the key of this sample's data type.

- int **get_int** (String member_name, int member_id)

  *Get the value of the given field, which is of type com.rti.dds.infrastructure.int or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, or **com.rti.dds.util.Enum** (p. 1038)).*

- long **get_uint** (String member_name, int member_id)

  *Get the value of the given field, which is of type com.rti.dds.infrastructure.int or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, or **com.rti.dds.util.Enum** (p. 1038)).*

- int **get_int_array** (int[ ] array, String member_name, int member_id)

  *Get a copy of the given array member. The array may contain members of type com.rti.dds.infrastructure.int or **com.rti.dds.util.Enum** (p. 1038).*

- int **get_uint_array** (long[ ] array, String member_name, int member_id)

  *Get a copy of the given array member. The array may contain members of type com.rti.dds.infrastructure.long or **com.rti.dds.util.Enum** (p. 1038).*

- void **get_int_seq** (**IntSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.

- void **get_uint_seq** (**LongSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.

- short **get_short** (String member_name, int member_id)

  Get the value of the given field, which is of type short or another type implicitly convertible to it (byte or char).

- int **get_ushort** (String member_name, int member_id)

  Get the value of the given field, which is of type com.rti.dds.infrastructure.short or another type implicitly convertible to it (byte or char).

- int **get_short_array** (short[ ] array, String member_name, int member_id)

Get a copy of the given array member.

- int **get_ushort_array** (int[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_short_seq** (**ShortSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- void **get_ushort_seq** (**IntSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- float **get_float** (String member_name, int member_id)

  Get the value of the given field, which is of type float.
- int **get_float_array** (float[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_float_seq** (**FloatSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- double **get_double** (String member_name, int member_id)

  Get the value of the given field, which is of type double or another type implicitly convertible to it (float).
- int **get_double_array** (double[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_double_seq** (**DoubleSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- boolean **get_boolean** (String member_name, int member_id)

  Get the value of the given field, which is of type boolean.
- int **get_boolean_array** (boolean[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_boolean_seq** (**BooleanSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- char **get_char** (String member_name, int member_id)

  Get the value of the given field, which is of type char.
- int **get_char_array** (char[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_char_seq** (**CharSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- byte **get_byte** (String member_name, int member_id)

  Get the value of the given field, which is of type byte.
- short **get_uint8** (String member_name, int member_id)

  Get the value of the given field, which is of type com.rti.dds.infrastructure.byte.
- byte **get_int8** (String member_name, int member_id)

  Get the value of the given field, which is of type com.rti.dds.infrastructure.byte.
- int **get_byte_array** (byte[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- int **get_int8_array** (byte[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- int **get_uint8_array** (short[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_byte_seq** (**ByteSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- void **get_int8_seq** (**ByteSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.

- void **get_uint8_seq** (**ShortSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- long **get_long** (String member_name, int member_id)

  *Get the value of the given field, which is of type long or another type implicitly convertible to it (byte, char, short, com.rti.↩ dds.infrastructure.short, com.rti.dds.infrastructure.int, com.rti.dds.infrastructure.long, or **com.rti.dds.util.Enum** (*p. *1038)).*
- BigInteger **get_ulong** (String member_name, int member_id)

  *Get the value of the given field, which is of type com.rti.dds.infrastructure.long or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, com.rti.dds.infrastructure.int, com.rti.dds.infrastructure.long, or **com.↩ rti.dds.util.Enum** (*p. *1038)).*
- int **get_long_array** (long[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- int **get_ulong_array** (BigInteger[ ] array, String member_name, int member_id)

  Get a copy of the given array member.
- void **get_long_seq** (**LongSeq** seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- void **get_ulong_seq** (List< BigInteger > seq, String member_name, int member_id)

  Get a copy of the given sequence member.
- String **get_string** (String member_name, int member_id)

  Get the value of the given field, which is of type com.rti.dds.infrastructure.String.
- void **get_complex_member** (**DynamicData** value_out, String member_name, int member_id)

  Get a copy of the value of the given field, which is of some composed type.
- void **set_int** (String member_name, int member_id, int value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.int.
- void **set_uint** (String member_name, int member_id, long value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.long.
- void **set_int_array** (String member_name, int member_id, int[ ] array)

  *Set the contents of the given array member. The array may contain members of type com.rti.dds.infrastructure.int or **com.rti.dds.util.Enum** (*p. *1038).*
- void **set_uint_array** (String member_name, int member_id, long[ ] array)

  *Set the contents of the given array member. The array may contain members of type com.rti.dds.infrastructure.long or **com.rti.dds.util.Enum** (*p. *1038).*
- void **set_int_seq** (String member_name, int member_id, **IntSeq** value)

  Set the contents of the given sequence member.
- void **set_uint_seq** (String member_name, int member_id, **LongSeq** value)

  Set the contents of the given sequence member.
- void **set_short** (String member_name, int member_id, short value)

  Set the value of the given field, which is of type short.
- void **set_ushort** (String member_name, int member_id, int value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.short.
- void **set_short_array** (String member_name, int member_id, short[ ] array)

  Set the contents of the given array member.
- void **set_ushort_array** (String member_name, int member_id, int[ ] array)

  Set the contents of the given array member.
- void **set_short_seq** (String member_name, int member_id, **ShortSeq** value)

  Set the contents of the given sequence member.
- void **set_ushort_seq** (String member_name, int member_id, **IntSeq** value)

  Set the contents of the given sequence member.
- void **set_float** (String member_name, int member_id, float value)

Set the value of the given field, which is of type float.

- void **set_float_array** (String member_name, int member_id, float[ ] array)

  Set the contents of the given array member.

- void **set_float_seq** (String member_name, int member_id, **FloatSeq** value)

  Set the contents of the given sequence member.

- void **set_double** (String member_name, int member_id, double value)

  Set the value of the given field, which is of type double.

- void **set_double_array** (String member_name, int member_id, double[ ] array)

  Set the contents of the given array member.

- void **set_double_seq** (String member_name, int member_id, **DoubleSeq** value)

  Set the contents of the given sequence member.

- void **set_boolean** (String member_name, int member_id, boolean value)

  Set the value of the given field, which is of type boolean.

- void **set_boolean_array** (String member_name, int member_id, boolean[ ] array)

  Set the contents of the given array member.

- void **set_boolean_seq** (String member_name, int member_id, **BooleanSeq** value)

  Set the contents of the given sequence member.

- void **set_char** (String member_name, int member_id, char value)

  Set the value of the given field, which is of type char.

- void **set_char_array** (String member_name, int member_id, char[ ] array)

  Set the contents of the given array member.

- void **set_char_seq** (String member_name, int member_id, **CharSeq** value)

  Set the contents of the given sequence member.

- void **set_byte** (String member_name, int member_id, byte value)

  Set the value of the given field, which is of type byte.

- void **set_uint8** (String member_name, int member_id, short value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.byte.

- void **set_int8** (String member_name, int member_id, byte value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.byte.

- void **set_byte_array** (String member_name, int member_id, byte[ ] array)

  Set the contents of the given array member.

- void **set_int8_array** (String member_name, int member_id, byte[ ] array)

  Set the contents of the given array member.

- void **set_uint8_array** (String member_name, int member_id, short[ ] array)

  Set the contents of the given array member.

- void **set_byte_seq** (String member_name, int member_id, **ByteSeq** value)

  Set the contents of the given sequence member.

- void **set_int8_seq** (String member_name, int member_id, **ByteSeq** value)

  Set the contents of the given sequence member.

- void **set_uint8_seq** (String member_name, int member_id, **ShortSeq** value)

  Set the contents of the given sequence member.

- void **set_long** (String member_name, int member_id, long value)

  Set the value of the given field, which is of type long.

- void **set_ulong** (String member_name, int member_id, BigInteger value)

  Set the value of the given field, which is of type com.rti.dds.infrastructure.long.

- void **set_long_array** (String member_name, int member_id, long[ ] array)

  Set the contents of the given array member.

- void **set_ulong_array** (String member_name, int member_id, BigInteger[ ] array)

  Set the contents of the given array member.
- void **set_long_seq** (String member_name, int member_id, **LongSeq** value)

  Set the contents of the given sequence member.
- void **set_ulong_seq** (String member_name, int member_id, List< BigInteger > value)

  Set the contents of the given sequence member.
- void **set_string** (String member_name, int member_id, String value)

  Set the value of the given field of type com.rti.dds.infrastructure.String.
- void **set_complex_member** (String member_name, int member_id, **DynamicData** value)

  *Copy the state of the given **com.rti.dds.dynamicdata.DynamicData** (p. 847) object into a member of this object.*
- int **to_cdr_buffer** (byte[ ] buffer, short representation)

  *Serializes a **DynamicData** (p. 847) object into a buffer of octets.*
- int **to_cdr_buffer** (byte[ ] buffer)

  *Serializes a **DynamicData** (p. 847) object into a CDR buffer of octets.*
- void **from_cdr_buffer** (byte[ ] buffer)

  *Deserializes a **DynamicData** (p. 847) object from a buffer of octets.*
- void **to_cdr_buffer** (**ByteSeq** sequence, short representation)

  *Serializes a **DynamicData** (p. 847) object into a sequence of octets.*
- void **to_cdr_buffer** (**ByteSeq** sequence)

  *Serializes a **DynamicData** (p. 847) object into a CDR sequence of octets.*
- void **from_cdr_buffer** (**ByteSeq** sequence)

  *Deserializes a **DynamicData** (p. 847) object from a sequence octets.*
- String **to_string** (**PrintFormatProperty** property)

  *Get a string representation of a **DynamicData** (p. 847) object.*
- String **to_string** ()

  *Get a string representation of a **DynamicData** (p. 847) object using the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387).*
- void **from_string** (String str, **PrintFormatKind** format_kind)

  *Populates a **DynamicData** (p. 847) object from a JSON string representation.*
- void **from_string** (String str)

  *Populates a **DynamicData** (p. 847) object from a JSON string representation.*
- **DynamicData** (**TypeCode** type, **DynamicDataProperty_t** property)

  *The constructor for new **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.*

Static Public Attributes

- static final int **MEMBER_ID_UNSPECIFIED**

  A sentinel value that indicates that no member ID is needed in order to perform some operation.
- static final **DynamicDataProperty_t** **PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 955).*

8.93.1 Detailed Description

A sample of any complex data type, which can be inspected and manipulated reflectively.

Objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847) represent corresponding objects of the type identified by their **com.rti.dds.typecode.TypeCode** (p. 1873). Because the definition of these types may not have existed at compile time on the system on which the application is running, you will interact with the data using an API of reflective getters and setters.

For example, if you had access to your data types at compile time, you could do this:
```
theValue = theObject.theField;
```

Instead, you will do something like this:
```
theValue = get(theObject, "theField");
```

com.rti.dds.dynamicdata.DynamicData (p. 847) objects can represent any complex data type, including those of type kinds **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792). They cannot represent objects of basic types (e.g., integers and strings). Since those type definitions always exist on every system, you can examine their objects directly.

8.93.2 Member Names and IDs

The members of a data type can be identified in one of two ways: by their name or by their numeric ID. The former is often more transparent to human users; the latter is typically faster.

You define the name and ID of a type member when you add that member to that type. If you define your type in IDL or XML, the name will be the field name that appears in the type definition; the ID will be the one-based index of the field in declaration order. For example, in the following IDL structure, the ID of `theLong` is 2.
```
struct MyNestedType {

    char theChar;

    octet theOctetArray[10];

    long long theMultidimensionalArray[4][6][12];

    sequence<long> myArrayOfSeq[8];

};


struct MyType {

    short theShort;

    long theLong;

    MyNestedType theNestedType;

};
```

For unions (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)), the ID of a member is the discriminator value corresponding to that member. To access the current discriminator of a union, you must use the **com.rti.dds.dynamicdata.DynamicData.get_member_info_by_index** (p. 873) operation on the **DynamicData** (p. 847) object using an index value of 0. This operation fills in a **com.rti.dds.dynamicdata.DynamicDataMemberInfo** (p. 953), then you can access the populated **com.rti.dds.dynamicdata.DynamicDataMemberInfo.member_id** (p. 954) field to get the current discriminator. Once you know the value of the discriminator, you can use it in the proper get/set_xxx() operations to access and set the member's value. Here is an example of accessing the discriminator:
```
DynamicDataMemberInfo memberInfo;

myDynamicData.get_member_info_by_index(memberInfo, 0);

DynamicDataMemberId discriminatorValue = memberInfo.member_id;

DDS_Long myMemberValue = myDynamicData.get_long(NULL, discriminatorValue);
```

8.93.2.1 Hierarchical Member Names

It is possible to refer to a nested member in a type without first having to use the **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) API. You can do this by using a hierarchical name. A hierarchical member name is a concatenation of member names separated by the '.' character. The hierarchical name describes the complete path from a top-level type to the nested member. For example, in the above type, any **DynamicData** (p. 847) API that receives a member name will accept "theNestedType.theChar" to refer to the char member in MyNestedType:

```
char myChar = myDynamicData.get_char("theNestedType.theChar", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

In order to access the value of `theChar` without using a hierarchical name, you would have to first bind to theNestedType and then get the value:

```
myDynamicData.bind_complex_member(myBoundData, "theNestedType", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

DDS_Char myChar = myBoundData.get_char("theChar", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

As you can see, using a hierarchical member name removes the need to call the **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) and **com.rti.dds.dynamicdata.DynamicData.unbind_complex_member** (p. 869) APIs, and allows for access to nested members at any depth directly from the top-level type.

The member name can also contain indexes to address members in arrays and sequences. For example, to set the third member in the array `theOctetArray`, you can pass in "theNestedType.theOctetArray[2]" as the member name to the com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte API. The index values when used as part of the member name are 0-based.

For multi-dimensional arrays, the indexes for each dimension should be listed comma-separated in between brackets. For example, to address a member of `theMultidimensionalArray`, the member name should be something like "theNestedType.theMultidimensionalArray[3,2,5]".

In complex types with arrays and sequences that contain other arrays and sequences, the hierarchical name may include multiple index values, one right after another. For example, in MyNestedType, `myArrayOfSeq` is an array of sequences. In order to set the third member of the sequence in the fourth member of the array, the member name would be "myNestedType.myArrayOfSeq[3][2]".

8.93.3 Arrays and Sequences

The "members" of array and sequence types, unlike those of structure and union types, don't have names or explicit member IDs. However, they may nevertheless be accessed by "ID": the ID is one more than the index. (The first element has ID 1, the second 2, etc.)

Multi-dimensional arrays are effectively flattened by this pattern. For example, for an array `theArray`[4][5], accessing ID 7 is equivalent to index 6, or the second element of the second group of 5.

To determine the length of a collection-typed member of a structure or union, you have two choices:

1. Get the length along with the data: call the appropriate array accessor (see **Getters and Setters** (p. 855)) and check the resulting length.
2. Get the length without getting the data itself: call **com.rti.dds.dynamicdata.DynamicData.get_member_info** (p. 871) and check the resulting **com.rti.dds.dynamicdata.DynamicDataMemberInfo.element_count** (p. 955).

8.93.4 Available Functionality

The Dynamic Data API is large when measured by the number of methods it contains. But each method falls into one of a very small number of categories. You will find it easier to navigate this documentation if you understand these categories.

8.93.4.1 Lifecycle and Utility Methods

Managing the lifecycle of **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects is simple. You have two choices:

1. Usually, you will go through a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) factory object, which will ensure that the type and property information for the new **com.rti.dds.dynamicdata.DynamicData** (p. 847) object corresponds to a registered type in your system.
2. In certain advanced cases, such as when you're navigating a nested structure, you will want to have a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object that is not bound up front to any particular type, or you will want to initialize the object in a custom way. In that case, you can call the constructor directly.

Table 8.451 Lifecycle

com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. **995**)	**com.rti.dds.dynamicdata.DynamicData** (p. **847**)
com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data (p. 998)	com.rti.dds.dynamicdata.DynamicData.DynamicData.DynamicData

You can also copy **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects:

- com.rti.dds.dynamicdata.DynamicData.copy

You can test them for equality:

- com.rti.dds.dynamicdata.DynamicData.DynamicData.equals

And you can print their contents:

- **com.rti.dds.dynamicdata.DynamicData.print** (p. 864)
- **com.rti.dds.dynamicdata.DynamicDataTypeSupport.print_data** (p. 999)

8.93.4.2 Getters and Setters

Most methods get or set the value of some field. These methods are named according to the type of the field they access.

The names of types vary across languages. The programming API for each language reflects that programming language. However, if your chosen language does not use the same names as the language that you used to define your types (e.g., IDL), or if you need to interoperate among programming languages, you will need to understand these differences. They are explained the following table. (Note: for modern C++, see the RTI Connext Modern C++ API reference.)

Table 8.452 Type Names Across Languages

Type	IDL	C, Traditional C++	Java	Ada
16-bit integer	short	DDS_Short	short	Standard.DDS.↩ Short
32-bit integer	long	DDS_Long	int	Standard.DDS.Long
64-bit integer	long long	DDS_LongLong	long	Standard.DDS.↩ Long_Long
Unsigned 16-bit integer	unsigned short	DDS_Unsigned↩ Short	short	Standard.DDS.↩ Unsigned_Short
Unsigned 32-bit integer	unsigned long	DDS_Unsigned↩ Long	int	Standard.DDS.Long
Unsigned 64-bit integer	unsigned long long	DDS_Unsigned↩ LongLong	long	Standard.DDS.↩ Unsigned_Long_↩ Long
float	float	DDS_Float	float	Standard.DDS.Float
double	double	DDS_Double	double	Standard.DDS.↩ Double
long double	long double	DDS_LongDouble	N/A (see CORE-14091 known issue)	Standard.DDS.↩ Long_Double
character	char	DDS_Char	char	Standard.DDS.Char
wide character	wchar	DDS_Wchar	char	Standard.DDS.↩ Wchar
octet	octet	DDS_Octet	byte	Standard.DDS.Octet
boolean	boolean	DDS_Boolean	boolean	Standard.DDS.↩ Boolean
string	string	DDS_Char∗	String	Standard.DDS.↩ String
wstring	wstring	DDS_Wchar∗	String	Standard.DDS.↩ Wide_String

When working with a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object representing an array or sequence, calling one of the "get" methods below for an index that is out of bounds will result in **com.rti.dds.infrastructure.↩ RETCODE_NO_DATA** (p. 1597). Calling "set" for an index that is past the end of a sequence will cause that sequence to automatically lengthen (filling with default contents).

When working with a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object whose type contains optional members, calling one of the "get" methods below on an unset optional member or any member that is part of an unset complex optional member will result in **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Table 8.453 Basic Types

Get	Set
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_int	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_int
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_uint	**com.rti.dds.dynamicdata.DynamicData.set_ulong** (p. 939)
com.rti.dds.dynamicdata.DynamicData.get_short (p. 881)	**com.rti.dds.dynamicdata.DynamicData.set_short** (p. 916)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_ushort	**com.rti.dds.dynamicdata.DynamicData.set_ushort** (p. 916)

Get	Set
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_long	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_long
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_ulong	com.rti.dds.dynamicdata.DynamicData.set_ulonglong
com.rti.dds.dynamicdata.DynamicData.get_float (p. 886)	**com.rti.dds.dynamicdata.DynamicData.set_float** (p. 921)
com.rti.dds.dynamicdata.DynamicData.get_double (p. 888)	**com.rti.dds.dynamicdata.DynamicData.set_double** (p. 923)
com.rti.dds.dynamicdata.DynamicData.get_longdouble	com.rti.dds.dynamicdata.DynamicData.set_longdouble
com.rti.dds.dynamicdata.DynamicData.get_boolean (p. 891)	**com.rti.dds.dynamicdata.DynamicData.set_boolean** (p. 925)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_byte	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_byte
com.rti.dds.dynamicdata.DynamicData.get_char (p. 893)	**com.rti.dds.dynamicdata.DynamicData.set_char** (p. 928)
com.rti.dds.dynamicdata.DynamicData.get_string (p. 908)	**com.rti.dds.dynamicdata.DynamicData.set_string** (p. 943)

Table 8.454 Structures, Arrays, and Other Complex Types

Get	Set
com.rti.dds.dynamicdata.DynamicData.get_↩ complex_member (p. 909)	**com.rti.dds.dynamicdata.DynamicData.set_↩ complex_member** (p. 944)

Table 8.455 Arrays of Basic Types

Get	Set
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_int_array	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_int_array
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_int_array	**com.rti.dds.dynamicdata.DynamicData.set_ulong↩ _array** (p. 940)
com.rti.dds.dynamicdata.DynamicData.get_short_↩ array (p. 882)	**com.rti.dds.dynamicdata.DynamicData.set_short_↩ array** (p. 917)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_ushort_array	**com.rti.dds.dynamicdata.DynamicData.set_ushort↩ _array** (p. 918)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_long_array	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_long_array
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_ulong_array	com.rti.dds.dynamicdata.DynamicData.set_ulonglong↩ _array
com.rti.dds.dynamicdata.DynamicData.get_float_↩ array (p. 886)	**com.rti.dds.dynamicdata.DynamicData.set_float_↩ array** (p. 921)
com.rti.dds.dynamicdata.DynamicData.get_↩ double_array (p. 889)	**com.rti.dds.dynamicdata.DynamicData.set_↩ double_array** (p. 924)

Get	Set
com.rti.dds.dynamicdata.DynamicData.get_↩ boolean_array (p. 891)	**com.rti.dds.dynamicdata.DynamicData.set_boolean** (p. 925)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_byte_array	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_byte_array
com.rti.dds.dynamicdata.DynamicData.get_char_↩ array (p. 894)	**com.rti.dds.dynamicdata.DynamicData.set_char_↩ array** (p. 929)

Table 8.456 Sequences of Basic Types

Get	Set
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_int_seq	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_int_seq
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_uint_seq	**com.rti.dds.dynamicdata.DynamicData.set_ulong↩ _seq** (p. 942)
com.rti.dds.dynamicdata.DynamicData.get_short_↩ seq (p. 884)	**com.rti.dds.dynamicdata.DynamicData.set_short_↩ seq** (p. 919)
com.rti.dds.dynamicdata.DynamicData.get_↩ ushort_seq (p. 885)	**com.rti.dds.dynamicdata.DynamicData.set_ushort↩ _seq** (p. 920)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_long_seq	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_long_seq
com.rti.dds.dynamicdata.DynamicData.get_ulonglong↩ _seq	com.rti.dds.dynamicdata.DynamicData.set_ulonglong↩ _seq
com.rti.dds.dynamicdata.DynamicData.get_float_↩ seq (p. 887)	**com.rti.dds.dynamicdata.DynamicData.set_float_↩ seq** (p. 922)
com.rti.dds.dynamicdata.DynamicData.get_↩ double_seq (p. 890)	**com.rti.dds.dynamicdata.DynamicData.set_↩ double_seq** (p. 925)
com.rti.dds.dynamicdata.DynamicData.get_↩ boolean_seq (p. 892)	**com.rti.dds.dynamicdata.DynamicData.set_↩ boolean_seq** (p. 927)
com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ get_byte_seq	com.rti.dds.dynamicdata.DynamicData.DynamicData.↩ set_byte_seq
com.rti.dds.dynamicdata.DynamicData.get_char_↩ seq (p. 895)	**com.rti.dds.dynamicdata.DynamicData.set_char_↩ seq** (p. 930)

In addition to getting or setting a field, you can "clear" its value; that is, set it to a default zero value.

- **com.rti.dds.dynamicdata.DynamicData.clear_all_members** (p. 862)
- **com.rti.dds.dynamicdata.DynamicData.clear_optional_member** (p. 863)

8.93.4.3 Query and Iteration

Not all components of your application will have static knowledge of all of the fields of your type. Sometimes, you will want to query meta-data about the fields that appear in a given data sample.

- **com.rti.dds.dynamicdata.DynamicData.get_type** (p. 869)
- **com.rti.dds.dynamicdata.DynamicData.get_type_kind** (p. 869)
- **com.rti.dds.dynamicdata.DynamicData.get_member_type** (p. 874)
- **com.rti.dds.dynamicdata.DynamicData.get_member_info** (p. 871)
- **com.rti.dds.dynamicdata.DynamicData.get_member_count** (p. 870)
- **com.rti.dds.dynamicdata.DynamicData.get_member_info_by_index** (p. 873)
- **com.rti.dds.dynamicdata.DynamicData.member_exists** (p. 870)
- **com.rti.dds.dynamicdata.DynamicData.member_exists_in_type** (p. 871)
- **com.rti.dds.dynamicdata.DynamicData.is_member_key** (p. 875)

8.93.4.4 Type/Object Association

Sometimes, you may want to change the association between a data object and its type. This is not something you can do with a typical object, but with **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects, it is a powerful capability. It allows you to, for example, examine nested structures without copying them by using a "bound" **com.rti.↩ dds.dynamicdata.DynamicData** (p. 847) object as a view into an enclosing **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

- **com.rti.dds.dynamicdata.DynamicData.bind_type** (p. 865)
- **com.rti.dds.dynamicdata.DynamicData.unbind_type** (p. 866)
- **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867)
- **com.rti.dds.dynamicdata.DynamicData.unbind_complex_member** (p. 869)

MT Safety:

UNSAFE. In general, using a single **com.rti.dds.dynamicdata.DynamicData** (p. 847) object concurrently from multiple threads is *unsafe*.

8.93.5 Constructor & Destructor Documentation

8.93.5.1 DynamicData()

```
DynamicData (
            TypeCode type,
            DynamicDataProperty_t property )
```

The constructor for new **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

The type parameter may be null. In that case, this **com.rti.dds.dynamicdata.DynamicData** (p. 847) must be *bound* with **com.rti.dds.dynamicdata.DynamicData.bind_type** (p. 865) or **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) before it can be used.

If the **com.rti.dds.typecode.TypeCode** (p. 1873) is not null, the newly constructed **com.rti.dds.dynamicdata.DynamicData** (p. 847) object will retain a reference to it. It is *not* safe to delete the **com.rti.dds.typecode.TypeCode** (p. 1873) until all samples that use it have themselves been deleted. You have two options:

- Keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873) object yourself, and delete it with **com.rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934) after you've deleted all of the objects that use it.
- Do not keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873). The garbage collector will delete it when it's eligible for collection.

In most cases, it is not necessary to call this constructor explicitly. Instead, use **com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data** (p. 998), and the **com.rti.dds.typecode.TypeCode** (p. 1873) and properties will be specified for you. Using the factory method also ensures that the memory management contract documented above is followed correctly, because the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) object maintains the **com.rti.dds.typecode.TypeCode** (p. 1873) used by the samples it creates.

Parameters

type	<<***in***>> *(*p. *156)* The type of which the new object will represent an object.
property	<<***in***>> *(*p. *156)* Properties that configure the behavior of the new object. Most users can simply use com.rti.dds.dynamicdata.DYNAMIC_DATA_PROPERTY_DEFAULT.

See also

com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data (p. 998)

8.93.6 Member Function Documentation

8.93.6.1 delete()

```
void delete ( )
```

Finalize and deallocate this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

MT Safety:

UNSAFE.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.DynamicData

Referenced by **DynamicDataTypeSupport.destroy_data()**.

8.93.6.2 equals()

```
boolean equals (
             Object o )
```

Indicate whether the contents of another **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample are the same as those of this one.

This operation compares the data and type of existing members.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with com.rti.dds.↩ infrastructure.false. See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

See also

http://java.sun.com/javase/6/docs/api/java/lang/Object.html#equals(java.↩ lang.Object)

8.93.6.3 copy_from()

```
Object copy_from (
             Object src )
```

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

Implements **Copyable** (p. 445).

8.93.6.4 clear_all_members()

```
void clear_all_members ( )
```

Clear the contents of all data members of this object.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 261)

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

See also

com.rti.dds.dynamicdata.DynamicData.clear_optional_member (p. 863)

8.93.6.5 clear_member()

```
void clear_member (
             String member_name,
             int member_id )
```

Clear the contents of a single data member of this object.

This method can be used to clear both optional and non-optional members.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member to clear or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member to clear or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 862)

8.93.6.6 clear_optional_member()

```
void clear_optional_member (
             String member_name,
             int member_id )
```

Clear the contents of a single optional data member of this object.

This method is only applicable to optional members. Members of unions, sequences, and arrays are not considered optional.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member (to look up the member by name), or null (to look up the member by its ID).
member_id	<<***in***>> *(p. 156)* The ID of the member (to look up the member by its ID), or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) (to look up the member by name). See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 862)

8.93.6.7 print()

```
void print (
            File fp,
            int indent )
```

Output a textual representation of this object and its contents to the given file.

This method is equivalent to **com.rti.dds.dynamicdata.DynamicDataTypeSupport.print_data** (p. 999).

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

Precondition

The type must be an aggregation or collection. That is, its **com.rti.dds.typecode.TCKind** (p. 1786) must be one of: **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790) or **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), otherwise this function fails with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

MT Safety:

UNSAFE.

Parameters

fp	<<***in***>> *(*p. *156)* The file into which the object should be printed (to print to standard output, provide the stream pointer 'stdout')
indent	<<***in***>> *(*p. *156)* The output of this method will be pretty-printed. This argument indicates the amount of initial indentation of the output.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicDataTypeSupport.print_data (p. 999)

8.93.6.8 get_info()

```
void get_info (
                DynamicDataInfo info_out )
```

Fill in the given descriptor with information about this **com.rti.dds.dynamicdata.DynamicData** (p. 847).

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will print an error log. See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

info_out	<<***out***>> *(p. 156)* The descriptor object whose contents will be overwritten by this operation.

8.93.6.9 bind_type()

```
void bind_type (
                TypeCode type )
```

If this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object is not yet associated with a data type, set that type now to the given **com.rti.dds.typecode.TypeCode** (p. 1873).

This advanced operation allows you to reuse a single **com.rti.dds.dynamicdata.DynamicData** (p. 847) object with multiple data types.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

```
DynamicData myData = new DynamicData(null, myProperties);

TypeCode myType = ...;

myData.bind_type(myType);

try {

    // Do something...

} finally {
```

```
    myData.unbind_type();
}
myData.delete();
```

Note that the **com.rti.dds.dynamicdata.DynamicData** (p. 847) object will retain a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873) object you provide. It is *not* safe to delete the **com.rti.dds.typecode.TypeCode** (p. 1873) until after it is unbound. You have two options:

- Keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873) object yourself, and delete it with **com.rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934) after you've finished using it.
- Do not keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873). The garbage collector will delete it when it's eligible for collection.

MT Safety:

UNSAFE.

Parameters

type	<<***in***>> *(*p. *156)* The type to associate with this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.unbind_type (p. 866)

8.93.6.10 unbind_type()

```
void unbind_type ( )
```

Dissociate this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object from any particular data type.

This step is necessary before the object can be associated with a new data type.

This operation clears all members as a side effect.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.bind_type (p. 865)

com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 862)

8.93.6.11 bind_complex_member()

```
void bind_complex_member (
              DynamicData value_out,
             String member_name,
             int member_id )
```

Use another **com.rti.dds.dynamicdata.DynamicData** (p. 847) object to provide access to a complex field of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

For example, consider the following data types:

```
struct MyFieldType {

    float theFloat;

};


struct MyOuterType {

    MyFieldType complexMember;

};
```

Suppose you have an instance of `MyOuterType`, and you would like to examine the contents of its member `complexMember`. To do this, you must *bind* another **com.rti.dds.dynamicdata.DynamicData** (p. 847) object to that member. This operation will bind the type code of the member to the provided **com.rti.dds.dynamicdata.DynamicData** (p. 847) object and perform additional initialization.

The following example demonstrates the usage pattern. Note that error handling has been omitted for brevity.

```
DynamicData outer = ...;

DynamicData toBeBound = new DynamicData(null, myProperties);

outer.bind_complex_member(

    toBeBound,

    "complexMember",

    DynamicData.MEMBER_ID_UNSPECIFIED);

try {
```

```
    float theFloatValue = toBeBound.get_float(
        "theFloat"
        DynamicData.MEMBER_ID_UNSPECIFIED);
} finally {
    outer.unbind_complex_member(toBeBound);
}
toBeBound.delete();
```

This operation is only permitted when the object toBeBound (named as in the example above) is not currently associated with any type, including already being bound to another member. You can see in the example that this object is created directly with the constructer and is not provided with a **com.rti.dds.typecode.TypeCode** (p. 1873).

Only a single member of a given **com.rti.dds.dynamicdata.DynamicData** (p. 847) object may be bound at one time – however, members *of* members may be recursively bound to any depth. Furthermore, while the outer object has a bound member, it may only be modified through that bound member. That is, after calling this member, all "set" operations on the outer object will be disabled until **com.rti.dds.dynamicdata.DynamicData.unbind_complex_member** (p. 869) has been called. Furthermore, any bound member must be unbound before a sample can be written or deleted.

This method is logically related to **com.rti.dds.dynamicdata.DynamicData.get_complex_member** (p. 909) in that both allow you to examine the state of nested objects. They are different in an important way: this method provides a view into an outer object, such that any change made to the inner object will be reflected in the outer. But the **com.rti.dds.dynamicdata.DynamicData.get_complex_member** (p. 909) operation *copies* the state of the nested object; changes to it will not be reflected in the source object.

Note that you can bind to a member of a sequence at an index that is past the current length of that sequence. In that case, this method behaves like a "set" method: it automatically lengthens the sequence (filling in default elements) to allow the bind to take place. See **Getters and Setters** (p. 855).

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(p. 156)* The object that you wish to bind to the field.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.unbind_complex_member (p. 869)

com.rti.dds.dynamicdata.DynamicData.get_complex_member (p. 909)

8.93.6.12 unbind_complex_member()

```
void unbind_complex_member (
              DynamicData value )
```

Tear down the association created by a **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) operation, committing any changes to the outer object since then.

Some changes to the outer object will not be observable until after you have performed this operation.

If you have called **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) on a data sample, you must unbind before writing or deleting the sample.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

value	<<***in***>> *(p. 156)* The same object you passed to **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867). This argument is used for error checking purposes.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 867)

8.93.6.13 get_type()

```
synchronized TypeCode get_type ( )
```

Get the data type, of which this **com.rti.dds.dynamicdata.DynamicData** (p. 847) represents an instance.

MT Safety:

UNSAFE.

8.93.6.14 get_type_kind()

```
TCKind get_type_kind ( )
```

Get the kind of this object's data type.

This is a convenience method. It's equivalent to calling **com.rti.dds.dynamicdata.DynamicData.get_type** (p. 869) followed by **com.rti.dds.typecode.TypeCode.kind** (p. 1876).

MT Safety:

UNSAFE.

8.93.6.15 get_member_count()

```
int get_member_count ( )
```

Get the number of members in the type.

For objects of type kind **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790) or **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), this method returns the number of elements in the collection.

For objects of type kind **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), it returns the number of fields in the type.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info_by_index (p. 873)

8.93.6.16 member_exists()

```
boolean member_exists (
            String member_name,
            int member_id )
```

Indicates whether a member exists in this sample.

You only need to specify the name OR the ID (not both).

If the member doesn't exist in the type, this function returns false. In all other cases, it provides the same result as **com.rti.dds.dynamicdata.DynamicDataMemberInfo.member_exists** (p. 954), which is retrieved with idref_DynamicDataMember_get_member_info.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with com.rti.dds.infrastructure.false. See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

See also

com.rti.dds.dynamicdata.DynamicData.member_exists_in_type (p. 871)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 951)

8.93.6.17 member_exists_in_type()

```
boolean member_exists_in_type (
            String member_name,
            int member_id )
```

Indicates whether a member of a particular name/ID exists in this data sample's type.

You only need to specify the name OR the ID (not both).

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with com.rti.dds.↩ infrastructure.false. See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

See also

com.rti.dds.dynamicdata.DynamicData.member_exists (p. 870)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 951)

8.93.6.18 get_member_info()

```
void get_member_info (
               DynamicDataMemberInfo info,
              String member_name,
              int member_id )
```

Fill in the given descriptor with information about the identified member of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

This operation is valid for objects of **com.rti.dds.typecode.TCKind** (p. 1786) **com.rti.dds.typecode.TCKind.TK_↩ ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_↩ STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), and **com.rti.dds.typecode.TCKind.TK_↩ VALUE** (p. 1792).

MT Safety:

UNSAFE.

When this sample represents a struct, a value type, or a union:

- If the specified member is not defined in the type, this function fails with **com.rti.dds.infrastructure.RETCODE↩ _NO_DATA** (p. 1597).
- If the specified member is defined in the type but doesn't exist in this data sample, this function returns an object with **com.rti.dds.dynamicdata.DynamicDataMemberInfo.member_exists** (p. 954) set to false.
- If the specified member is defined in the type and exists in this data sample, **com.rti.dds.dynamicdata.↩ DynamicDataMemberInfo.member_exists** (p. 954) is true.

When this sample represents a sequence and `member_id` is the 1-based element index:

- If `member_id` is greater than the sequence's maximum length, this function fails with **com.rti.dds.↩ infrastructure.RETCODE_NO_DATA** (p. 1597).
- If `member_id` is greater than the sequence's current length but smaller than or equal to its maximum length, this function returns an object with **com.rti.dds.dynamicdata.DynamicDataMemberInfo.member_exists** (p. 954) set to false.
- If `member_id` is smaller than or equal to the current length, **com.rti.dds.dynamicdata.DynamicData↩ MemberInfo.member_exists** (p. 954) is true.

When this sample represents an array, this function either fails with with **com.rti.dds.infrastructure.RETCODE_NO↩ _DATA** (p. 1597) when the index is out of bounds or else returns an object with **com.rti.dds.dynamicdata.Dynamic↩ DataMemberInfo.member_exists** (p. 954) set to true.

Parameters

info	*<<**out**>> (p. 156)* The descriptor object whose contents will be overwritten by this operations.
member_name	*<<**in**>> (p. 156)* The name of the member for which to get the info or null to look up the member by its ID. Only one of the name and the ID may be unspecified.
member_id	*<<**in**>> (p. 156)* The ID of the member for which to get the info, or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info_by_index (p. 873)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 951)

8.93.6.19 get_member_info_by_index()

```
void get_member_info_by_index (
              DynamicDataMemberInfo info,
             int index )
```

Fill in the given descriptor with information about the identified member of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

This operation is valid for objects of **com.rti.dds.typecode.TCKind** (p. 1786) **com.rti.dds.typecode.TCKind.TK_↩ ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_↩ STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), and **com.rti.dds.typecode.TCKind.TK_↩ UNION** (p. 1789).

MT Safety:

UNSAFE.

Parameters

info	*<<**out**>> (p. 156)* The descriptor object whose contents will be overwritten by this operations.
index	*<<**in**>> (p. 156)* The zero-base of the member for which to get the info.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 871)

com.rti.dds.dynamicdata.DynamicData.get_member_count (p. 870)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 951)

8.93.6.20 get_member_type()

```
TypeCode get_member_type (
            String member_name,
            int member_id )
```

Get the type of the given member of this sample.

The member can be looked up either by name or by ID.

This operation is valid for objects of **com.rti.dds.typecode.TCKind** (p. 1786) **com.rti.dds.typecode.TCKind.TK_↩ ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_↩ STRUCT** (p. 1789), and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792). For type kinds **com.rti.dds.typecode.↩ TCKind.TK_ARRAY** (p. 1790) and **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790), the index into the collection is taken to be one less than the ID, if specified. If this index is valid, this operation will return the content type of this collection.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 871)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 951)

Referenced by **DynamicData.get_char()**, **DynamicData.get_char_array()**, **DynamicData.get_char_seq()**, **DynamicData.get_string()**, **DynamicData.set_char()**, **DynamicData.set_char_array()**, **DynamicData.set_char_seq()**, and **DynamicData.set_string()**.

8.93.6.21 is_member_key()

```
boolean is_member_key (
            String member_name,
            int member_id )
```

Indicates whether a given member forms part of the key of this sample's data type.

This operation is only valid for samples of types of kind **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.93.6.22 get_int()

```
int get_int (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.int or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, or **com.rti.dds.util.Enum** (p. 1038)).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int

Referenced by **DynamicData.get_uint()**.

8.93.6.23 get_uint()

```
long get_uint (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.int or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, or **com.rti.dds.util.Enum** (p. 1038)).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int

References **DynamicData.get_int()**.

8.93.6.24 get_int_array()

```
int get_int_array (
             int[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member. The array may contain members of type com.rti.dds.infrastructure.int or **com.rti.dds.util.Enum** (p. 1038).

This method will perform an automatic conversion from **com.rti.dds.infrastructure.IntSeq** (p. 1162).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_seq

Referenced by **DynamicData.get_uint_array()**.

8.93.6.25 get_uint_array()

```
int get_uint_array (
             long[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member. The array may contain members of type com.rti.dds.infrastructure.long or **com.rti.dds.util.Enum** (p. 1038).

This method will perform an automatic conversion from com.rti.dds.infrastructure.UnsignedLongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 156)* An already-allocated array, into which the elements will be copied. The array must be next bigger type to hold the unsigned values.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_ulong_array (p. 940)

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint_seq

References **DynamicData.get_int_array()**.

8.93.6.26 get_int_seq()

```
void get_int_seq (
              IntSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.int.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(*p. *156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_array

References **IntSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.get_uint_seq()**.

8.93.6.27 get_uint_seq()

```
void get_uint_seq (
              LongSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_ulong_seq (p. 942)

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_array

References **DynamicData.get_int_seq()**, **IntSeq.getInt()**, **IntSeq.getMaximum()**, **LongSeq.getMaximum()**, **LongSeq.set()**, **AbstractSequence.setMaximum()**, **AbstractPrimitiveSequence.setSize()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.28 get_short()

```
short get_short (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type short or another type implicitly convertible to it (byte or char).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.set_short (p. 916)

Referenced by **DynamicData.get_ushort()**.

8.93.6.29 get_ushort()

```
int get_ushort (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.short or another type implicitly convertible to it (byte or char).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

A Java int value with the unsigned value of the indicated member.

See also

com.rti.dds.dynamicdata.DynamicData.set_ushort (p. 916)

References **DynamicData.get_short()**.

8.93.6.30 get_short_array()

```
int get_short_array (
             short[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.ShortSeq** (p. 1686).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_short_array (p. 917)

com.rti.dds.dynamicdata.DynamicData.get_short_seq (p. 884)

Referenced by **DynamicData.get_ushort_array()**.

8.93.6.31 get_ushort_array()

```
int get_ushort_array (
             int[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from com.rti.dds.infrastructure.UnsignedShortSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied. The array must be next bigger type to hold the unsigned values.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_ushort_array (p. 918)

com.rti.dds.dynamicdata.DynamicData.get_ushort_seq (p. 885)

References **DynamicData.get_short_array()**.

8.93.6.32 get_short_seq()

```
void get_short_seq (
              ShortSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_short_seq (p. 919)

com.rti.dds.dynamicdata.DynamicData.get_short_array (p. 882)

References **ShortSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.get_ushort_seq()**.

8.93.6.33 get_ushort_seq()

```
void get_ushort_seq (
              IntSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_ushort_seq (p. 920)

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ushort_array

References **DynamicData.get_short_seq()**, **IntSeq.getMaximum()**, **ShortSeq.getMaximum()**, **ShortSeq.get↩ Short()**, **IntSeq.set()**, **AbstractSequence.setMaximum()**, **AbstractPrimitiveSequence.setSize()**, and **Abstract↩ PrimitiveSequence.size()**.

8.93.6.34 get_float()

```
float get_float (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type float.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.set_float (p. 921)

8.93.6.35 get_float_array()

```
int get_float_array (
             float[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.FloatSeq** (p. 1049).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_float_array (p. 921)

com.rti.dds.dynamicdata.DynamicData.get_float_seq (p. 887)

8.93.6.36 get_float_seq()

```
void get_float_seq (
              FloatSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of float.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_float_seq (p. 922)

com.rti.dds.dynamicdata.DynamicData.get_float_array (p. 886)

References **FloatSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.37 get_double()

```
double get_double (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type double or another type implicitly convertible to it (float).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.set_double (p. 923)

8.93.6.38 get_double_array()

```
int get_double_array (
             double[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.DoubleSeq** (p. 824).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_double_array (p. 924)

com.rti.dds.dynamicdata.DynamicData.get_double_seq (p. 890)

8.93.6.39 get_double_seq()

```
void get_double_seq (
              DoubleSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of double.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_double_seq (p. 925)

com.rti.dds.dynamicdata.DynamicData.get_double_array (p. 889)

References **DoubleSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.40 get_boolean()

```
boolean get_boolean (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type boolean.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.set_boolean (p. 925)

8.93.6.41 get_boolean_array()

```
int get_boolean_array (
             boolean[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.BooleanSeq** (p. 359).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 156)* An already-allocated array, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_boolean_array (p. 926)

com.rti.dds.dynamicdata.DynamicData.get_boolean_seq (p. 892)

8.93.6.42 get_boolean_seq()

```
void get_boolean_seq (
              BooleanSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of boolean.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(*p. *156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_boolean_seq (p. 927)

com.rti.dds.dynamicdata.DynamicData.get_boolean_array (p. 891)

References **BooleanSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.43 get_char()

```
char get_char (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type char.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.set_char (p. 928)

References **DynamicData.get_member_type()**, **TypeCode.kind()**, and **TCKind.TK_WCHAR**.

8.93.6.44 get_char_array()

```
int get_char_array (
             char[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.CharSeq** (p. 416).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 156)* An already-allocated array, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_char_array (p. 929)

com.rti.dds.dynamicdata.DynamicData.get_char_seq (p. 895)

References **TypeCode.content_type()**, **DynamicData.get_member_type()**, **TypeCode.kind()**, **TCKind.TK_ARRAY**, and **TCKind.TK_WCHAR**.

8.93.6.45 get_char_seq()

```
void get_char_seq (
             CharSeq seq,
            String member_name,
            int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of char.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_char_seq (p. 930)

com.rti.dds.dynamicdata.DynamicData.get_char_array (p. 894)

References **TypeCode.content_type()**, **DynamicData.get_member_type()**, **CharSeq.getMaximum()**, **TypeCode.kind()**, **AbstractPrimitiveSequence.size()**, **TCKind.TK_SEQUENCE**, and **TCKind.TK_WCHAR**.

8.93.6.46 get_byte()

```
byte get_byte (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type byte.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte

Referenced by **DynamicData.get_int8()**, and **DynamicData.get_uint8()**.

8.93.6.47 get_uint8()

```
short get_uint8 (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.byte.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

A short value with the unsigned value of the indicated member.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_uint8

References **DynamicData.get_byte()**.

8.93.6.48 get_int8()

```
byte get_int8 (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.byte.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

A byte value with the value of the indicated member.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int8

References **DynamicData.get_byte()**.

8.93.6.49 get_byte_array()

```
int get_byte_array (
            byte[] array,
            String member_name,
            int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_byte_seq

Referenced by **DynamicData.get_int8_array()**, and **DynamicData.get_uint8_array()**.

8.93.6.50 get_int8_array()

```
int get_int8_array (
             byte[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 156)* An already-allocated array, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int8_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int8_seq

References **DynamicData.get_byte_array()**.

8.93.6.51 get_uint8_array()

```
int get_uint8_array (
             short[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied. The array must be next bigger type to hold the unsigned values.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_uint8_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint8_seq

References **DynamicData.get_byte_array()**.

8.93.6.52 get_byte_seq()

```
void get_byte_seq (
              ByteSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_byte_array

References **ByteSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.get_int8_seq()**, and **DynamicData.get_uint8_seq()**.

8.93.6.53 get_int8_seq()

```
void get_int8_seq (
               ByteSeq seq,
              String member_name,
              int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	*<<**out**>> (p. 156)* A sequence, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int8_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int8_array

References **DynamicData.get_byte_seq()**.

8.93.6.54 get_uint8_seq()

```
void get_uint8_seq (
               ShortSeq seq,
```

```
            String member_name,
            int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(*p. *156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_uint8_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint8_array

References **DynamicData.get_byte_seq()**, **ByteSeq.getByte()**, **ByteSeq.getMaximum()**, **ShortSeq.getMaximum()**, **ShortSeq.set()**, **AbstractSequence.setMaximum()**, **AbstractPrimitiveSequence.setSize()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.55 get_long()

```
long get_long (
            String member_name,
            int member_id )
```

Get the value of the given field, which is of type long or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, com.rti.dds.infrastructure.int, com.rti.dds.infrastructure.long, or **com.rti.dds.util.Enum** (p. 1038)).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_long

Referenced by **DynamicData.get_ulong()**.

8.93.6.56 get_ulong()

```
BigInteger get_ulong (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.long or another type implicitly convertible to it (byte, char, short, com.rti.dds.infrastructure.short, com.rti.dds.infrastructure.int, com.rti.dds.infrastructure.long, or **com.rti.dds.util.Enum** (p. 1038)).

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

A Java BigInteger value with the unsigned value of the indicated member.

See also

com.rti.dds.dynamicdata.DynamicData.set_ulonglong

References **DynamicData.get_long()**.

8.93.6.57 get_long_array()

```
int get_long_array (
            long[] array,
            String member_name,
            int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **com.rti.dds.infrastructure.LongSeq** (p. 1288).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	*<<**out**>> (p. 156)* An already-allocated array, into which the elements will be copied.
member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_long_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_long_seq

Referenced by **DynamicData.get_ulong_array()**.

8.93.6.58 get_ulong_array()

```
int get_ulong_array (
             BigInteger[] array,
             String member_name,
             int member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from com.rti.dds.infrastructure.BigIntegerSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

array	<<***out***>> *(p. 156)* An already-allocated array, into which the elements will be copied. The array must be next bigger type to hold the unsigned values.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Returns

The size of the output array

See also

com.rti.dds.dynamicdata.DynamicData.set_ulonglong_array

com.rti.dds.dynamicdata.DynamicData.get_ulonglong_seq

References **DynamicData.get_long_array()**.

8.93.6.59 get_long_seq()

```
void get_long_seq (
              LongSeq seq,
             String member_name,
             int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_long_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_long_array

References **LongSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.get_ulong_seq()**.

8.93.6.60 get_ulong_seq()

```
void get_ulong_seq (
            List< BigInteger > seq,
            String member_name,
            int member_id )
```

Get a copy of the given sequence member.

The provided sequence will be automatically resized as necessary.

This method will perform an automatic conversion from an array of com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

seq	<<***out***>> *(p. 156)* A sequence, into which the elements will be copied.
member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_ulonglong_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ulong_array

References **DynamicData.get_long_seq()**, **LongSeq.getLong()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.61 get_string()

```
String get_string (
             String member_name,
             int member_id )
```

Get the value of the given field, which is of type com.rti.dds.infrastructure.String.

The member may be specified by name or by ID.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

This operation is also valid for members of kind **com.rti.dds.typecode.TCKind.TK_WSTRING** (p. 1792). In this case, each wide character is truncated into a String character. This may cause loss of information for wide strings that are encoded using more bytes per character than String does.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_string (p. 943)

References **DynamicData.get_member_type()**, **TypeCode.kind()**, and **TCKind.TK_WSTRING**.

8.93.6.62 get_complex_member()

```
void get_complex_member (
             DynamicData value_out,
```

```
            String member_name,
            int member_id )
```

Get a copy of the value of the given field, which is of some composed type.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

The member may be of type kind **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), **com.rti.dds.typecode.↩ TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.↩ TCKind.TK_VALUE** (p. 1792), or **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789). It may be specified by name or by ID.

This method is logically related to **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) in that both allow you to examine the state of nested objects. They are different in an important way: this method provides a *copy* of the data; changes to it will not be reflected in the source object.

MT Safety:

UNSAFE.

Parameters

value_out	<<***out***>> *(*p. *156)* The **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample whose contents will be overwritten by this operation. This object must *not* be a bound member of another **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.
member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).

Exceptions

One	of the **Standard Return Codes** (p. 261). If the member is optional and not set, this operation will return **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This operation may also return **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.dds.dynamicdata.DynamicData.set_complex_member (p. 944)

com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 867)

8.93.6.63 set_int()

```
void set_int (
            String member_name,
```

```
            int member_id,
            int value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.int.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int

Referenced by **DynamicData.set_uint()**.

8.93.6.64 set_uint()

```
void set_uint (
            String member_name,
            int member_id,
            long value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* The value to which to set the member.

Exceptions

IllegalArgumentException	if the value is bigger than the maximum unsigged value of the member. Or lower than 0.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint

References **DynamicData.set_int()**.

8.93.6.65 set_int_array()

```
void set_int_array (
             String member_name,
             int member_id,
             int[] array )
```

Set the contents of the given array member. The array may contain members of type com.rti.dds.infrastructure.int or **com.rti.dds.util.Enum** (p. 1038).

This method will perform an automatic conversion to **com.rti.dds.infrastructure.IntSeq** (p. 1162).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(*p. *156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int_seq

Referenced by **DynamicData.set_uint_array()**.

8.93.6.66 set_uint_array()

```
void set_uint_array (
            String member_name,
            int member_id,
            long[] array )
```

Set the contents of the given array member. The array may contain members of type com.rti.dds.infrastructure.long or **com.rti.dds.util.Enum** (p. 1038).

This method will perform an automatic conversion to com.rti.dds.infrastructure.UnsignedLongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_array

com.rti.dds.dynamicdata.DynamicData.set_ulong_seq (p. 942)

References **DynamicData.set_int_array()**.

8.93.6.67 set_int_seq()

```
void set_int_seq (
             String member_name,
             int member_id,
              IntSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.int.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int_array

References **IntSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.set_uint_seq()**.

8.93.6.68 set_uint_seq()

```
void set_uint_seq (
             String member_name,
             int member_id,
              LongSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint_seq

com.rti.dds.dynamicdata.DynamicData.set_ulong_array (p. 940)

References **LongSeq.getLong()**, **LongSeq.getMaximum()**, **IntSeq.set()**, **DynamicData.set_int_seq()**, **Abstract↩ PrimitiveSequence.setSize()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.69 set_short()

```
void set_short (
            String member_name,
            int member_id,
            short value )
```

Set the value of the given field, which is of type short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_short (p. 881)

Referenced by **DynamicData.set_ushort()**.

8.93.6.70 set_ushort()

```
void set_ushort (
            String member_name,
```

```
            int member_id,
            int value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* The value to which to set the member.

Exceptions

IllegalArgumentException	if the value is bigger than the maximum unsigged value of the member. Or lower than 0.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ushort

References **DynamicData.set_short()**.

8.93.6.71 set_short_array()

```
void set_short_array (
            String member_name,
            int member_id,
            short[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.ShortSeq** (p. 1686).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_short_array (p. 882)

com.rti.dds.dynamicdata.DynamicData.set_short_seq (p. 919)

Referenced by **DynamicData.set_ushort_array()**.

8.93.6.72 set_ushort_array()

```
void set_ushort_array (
             String member_name,
             int member_id,
             int[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to com.rti.dds.infrastructure.UnsignedShortSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ushort_array

com.rti.dds.dynamicdata.DynamicData.set_ushort_seq (p. 920)

References **DynamicData.set_short_array()**.

8.93.6.73 set_short_seq()

```
void set_short_seq (
             String member_name,
             int member_id,
              ShortSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_short_seq (p. 884)

com.rti.dds.dynamicdata.DynamicData.set_short_array (p. 917)

References **ShortSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.set_ushort_seq()**.

8.93.6.74 set_ushort_seq()

```
void set_ushort_seq (
             String member_name,
             int member_id,
              IntSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.short.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_ushort_seq (p. 885)

com.rti.dds.dynamicdata.DynamicData.set_ushort_array (p. 918)

References **IntSeq.getInt()**, **IntSeq.getMaximum()**, **ShortSeq.set()**, **DynamicData.set_short_seq()**, **AbstractPrimitiveSequence.setSize()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.75 set_float()

```
void set_float (
            String member_name,
            int member_id,
            float value )
```

Set the value of the given field, which is of type float.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_float (p. 886)

8.93.6.76 set_float_array()

```
void set_float_array (
            String member_name,
```

```
int member_id,
float[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.FloatSeq** (p. 1049).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_float_array (p. 886)

com.rti.dds.dynamicdata.DynamicData.set_float_seq (p. 922)

8.93.6.77 set_float_seq()

```
void set_float_seq (
             String member_name,
             int member_id,
              FloatSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of float.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_float_seq (p. 887)

com.rti.dds.dynamicdata.DynamicData.set_float_array (p. 921)

References **FloatSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.78 set_double()

```
void set_double (
             String member_name,
             int member_id,
             double value )
```

Set the value of the given field, which is of type double.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_double (p. 888)

8.93.6.79 set_double_array()

```
void set_double_array (
             String member_name,
             int member_id,
             double[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.DoubleSeq** (p. 824).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	*<<**in**>> (p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_double_array (p. 889)

com.rti.dds.dynamicdata.DynamicData.set_double_seq (p. 925)

8.93.6.80 set_double_seq()

```
void set_double_seq (
             String member_name,
             int member_id,
              DoubleSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of double.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_double_seq (p. 890)

com.rti.dds.dynamicdata.DynamicData.set_double_array (p. 924)

References **DoubleSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.81 set_boolean()

```
void set_boolean (
            String member_name,
            int member_id,
            boolean value )
```

Set the value of the given field, which is of type boolean.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_boolean (p. 891)

8.93.6.82 set_boolean_array()

```
void set_boolean_array (
            String member_name,
            int member_id,
            boolean[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.BooleanSeq** (p. 359).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	*<<**in**>> (p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_boolean_array (p. 891)

com.rti.dds.dynamicdata.DynamicData.set_boolean_seq (p. 927)

8.93.6.83 set_boolean_seq()

```
void set_boolean_seq (
             String member_name,
             int member_id,
              BooleanSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of boolean.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_boolean_seq (p. 892)

com.rti.dds.dynamicdata.DynamicData.set_boolean_array (p. 926)

References **BooleanSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.84 set_char()

```
void set_char (
             String member_name,
             int member_id,
             char value )
```

Set the value of the given field, which is of type char.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_char (p. 893)

References **DynamicData.get_member_type()**, **TypeCode.kind()**, and **TCKind.TK_WCHAR**.

8.93.6.85 set_char_array()

```
void set_char_array (
            String member_name,
            int member_id,
            char[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.CharSeq** (p. 416).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_char_array (p. 894)

com.rti.dds.dynamicdata.DynamicData.set_char_seq (p. 930)

References **TypeCode.content_type()**, **DynamicData.get_member_type()**, **TypeCode.kind()**, **TCKind.TK_ARRAY**, and **TCKind.TK_WCHAR**.

8.93.6.86 set_char_seq()

```
void set_char_seq (
            String member_name,
            int member_id,
             CharSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of char.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_char_seq (p. 895)

com.rti.dds.dynamicdata.DynamicData.set_char_array (p. 929)

References **TypeCode.content_type()**, **DynamicData.get_member_type()**, **CharSeq.getMaximum()**, **TypeCode.kind()**, **AbstractPrimitiveSequence.size()**, **TCKind.TK_SEQUENCE**, and **TCKind.TK_WCHAR**.

8.93.6.87 set_byte()

```
void set_byte (
            String member_name,
            int member_id,
            byte value )
```

Set the value of the given field, which is of type byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_byte

Referenced by **DynamicData.set_int8()**, and **DynamicData.set_uint8()**.

8.93.6.88 set_uint8()

```
void set_uint8 (
            String member_name,
            int member_id,
            short value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* The value to which to set the member.

Exceptions

IllegalArgumentException	if the value is bigger than the maximum unsigged value of the member. Or lower than 0.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint8

References **DynamicData.set_byte()**.

8.93.6.89 set_int8()

```
void set_int8 (
             String member_name,
             int member_id,
             byte value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int8

References **DynamicData.set_byte()**.

8.93.6.90 set_byte_array()

```
void set_byte_array (
            String member_name,
            int member_id,
            byte[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_byte_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte_seq

Referenced by **DynamicData.set_int8_array()**, and **DynamicData.set_uint8_array()**.

8.93.6.91 set_int8_array()

```
void set_int8_array (
            String member_name,
            int member_id,
            byte[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int8_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int8_seq

References **DynamicData.set_byte_array()**.

8.93.6.92 set_uint8_array()

```
void set_uint8_array (
            String member_name,
            int member_id,
            short[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.ByteSeq** (p. 395).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint8_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_uint8_seq

References **DynamicData.set_byte_array()**.

8.93.6.93 set_byte_seq()

```
void set_byte_seq (
            String member_name,
            int member_id,
             ByteSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_byte_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_byte_array

References **ByteSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.set_int8_seq()**, and **DynamicData.set_uint8_seq()**.

8.93.6.94 set_int8_seq()

```
void set_int8_seq (
            String member_name,
```

```
int member_id,
 ByteSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_int8_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int8_array

References **DynamicData.set_byte_seq()**.

8.93.6.95 set_uint8_seq()

```
void set_uint8_seq (
             String member_name,
             int member_id,
              ShortSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.byte.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_uint8_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_uint8_array

References **ShortSeq.getMaximum()**, **ShortSeq.getShort()**, **ByteSeq.set()**, **DynamicData.set_byte_seq()**, **AbstractPrimitiveSequence.setSize()**, and **AbstractPrimitiveSequence.size()**.

8.93.6.96 set_long()

```
void set_long (
            String member_name,
            int member_id,
            long value )
```

Set the value of the given field, which is of type long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_long

Referenced by **DynamicData.set_ulong()**.

8.93.6.97 set_ulong()

```
void set_ulong (
            String member_name,
            int member_id,
            BigInteger value )
```

Set the value of the given field, which is of type com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The value to which to set the member.

Exceptions

IllegalArgumentException	if the value is bigger than the maximum unsigged value of the member. Or lower than 0.

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ulong

References **DynamicData.set_long()**.

8.93.6.98 set_long_array()

```
void set_long_array (
            String member_name,
            int member_id,
            long[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to **com.rti.dds.infrastructure.LongSeq** (p. 1288).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(p. 156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_long_array

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_long_seq

Referenced by **DynamicData.set_ulong_array()**.

8.93.6.99 set_ulong_array()

```
void set_ulong_array (
            String member_name,
            int member_id,
            BigInteger[] array )
```

Set the contents of the given array member.

This method will perform an automatic conversion to com.rti.dds.infrastructure.BigIntegerSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
array	<<***in***>> *(*p. *156)* The elements to copy.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_ulong_array

com.rti.dds.dynamicdata.DynamicData.set_ulonglong_seq

References **DynamicData.set_long_array()**.

8.93.6.100 set_long_seq()

```
void set_long_seq (
            String member_name,
```

```
int member_id,
 LongSeq value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.DynamicData.get_long_seq

com.rti.dds.dynamicdata.DynamicData.DynamicData.set_long_array

References **LongSeq.getMaximum()**, and **AbstractPrimitiveSequence.size()**.

Referenced by **DynamicData.set_ulong_seq()**.

8.93.6.101 set_ulong_seq()

```
void set_ulong_seq (
             String member_name,
             int member_id,
             List< BigInteger > value )
```

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of com.rti.dds.infrastructure.long.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

MT Safety:

UNSAFE.

Parameters

member_name	*<<**in**>> (p. 156)* The name of the member or null to look up the member by its ID.
member_id	*<<**in**>> (p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	*<<**in**>> (p. 156)* A sequence, from which the elements will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_ulonglong_seq

com.rti.dds.dynamicdata.DynamicData.set_ulonglong_array

References **LongSeq.set()**, **DynamicData.set_long_seq()**, **AbstractPrimitiveSequence.setSize()**, and **Abstract↩ PrimitiveSequence.size()**.

8.93.6.102 set_string()

```
void set_string (
             String member_name,
             int member_id,
             String value )
```

Set the value of the given field of type com.rti.dds.infrastructure.String.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

This operation is also valid for members of kind **com.rti.dds.typecode.TCKind.TK_WSTRING** (p. 1792). In this case, each String character is encoded in a wide char.

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(*p. *156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(*p. *156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(*p. *156)* The value to which to set the member.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_string (p. 908)

References **DynamicData.get_member_type()**, **TypeCode.kind()**, and **TCKind.TK_WSTRING**.

8.93.6.103 set_complex_member()

```
void set_complex_member (
             String member_name,
             int member_id,
              DynamicData value )
```

Copy the state of the given **com.rti.dds.dynamicdata.DynamicData** (p. 847) object into a member of this object.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

The member may be of type kind **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), **com.rti.dds.typecode.↩ TCKind.TK_SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.↩ TCKind.TK_VALUE** (p. 1792), or **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789). It may be specified by name or by ID.

Example: Copying Data

This method can be used with **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 867) to copy from one **com.rti.dds.dynamicdata.DynamicData** (p. 847) object to another efficiently. Suppose the following data structure:

```
struct Bar {

    short theShort;

};

struct Foo {
```

```
    Bar theBar;
};
```

Suppose we have two instances of `Foo`, `foo_dst` and `foo_src`, and we want to replace the contents of `foo_dst.theBar` with the contents of `foo_src.theBar`. The following example shows how to do this (error handling has been omitted for the sake of brevity).

```
DynamicData foo_dst = ...;
DynamicData foo_src = ...;
DynamicData bar = new DynamicData(null, myProperties);
// Point to the source of the copy:
foo_src.bind_complex_member(
    bar,
    "theBar",
    DynamicData.MEMBER_ID_UNSPECIFIED);
try {
    // Just one copy:
    foo_dst.set_complex_member(
        "theBar",
        DynamicData.MEMBER_ID_UNSPECIFIED,
        bar);
} finally {
    // Tear down:
    foo_src.unbind_complex_member(bar);
}
bar.delete();
```

MT Safety:

UNSAFE.

Parameters

member_name	<<***in***>> *(p. 156)* The name of the member or null to look up the member by its ID.
member_id	<<***in***>> *(p. 156)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 951) to look up by name. See **Member Names and IDs** (p. 853).
value	<<***in***>> *(p. 156)* The source **com.rti.dds.dynamicdata.DynamicData** (p. 847) object whose contents will be copied.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

See also

com.rti.dds.dynamicdata.DynamicData.get_complex_member (p. 909)

com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 867)

8.93.6.104 to_cdr_buffer() [1/4]

```
int to_cdr_buffer (
            byte[] buffer,
            short representation )
```

Serializes a **DynamicData** (p. 847) object into a buffer of octets.

This method serializes a **DynamicData** (p. 847) object into a buffer of octets using the input data representation. See **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) for details.

Parameters

buffer	<<***out***>> *(p. 156)*. Serialization buffer.
representation	<<***in***>> *(p. 156)*. Representation used to serialize the data

Referenced by **DynamicData.to_cdr_buffer()**, and **DynamicDataTypeSupport.to_cdr_buffer()**.

8.93.6.105 to_cdr_buffer() [2/4]

```
int to_cdr_buffer (
            byte[] buffer )
```

Serializes a **DynamicData** (p. 847) object into a CDR buffer of octets.

This method serializes a **DynamicData** (p. 847) object into a buffer of octets, using CDR as the data representation. Calling this method is equivalent to calling **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) with **com.rti.↩ dds.infrastructure.DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION** (p. 214) as the representation.

The input buffer must be large enough to store the serialized representation of the **DynamicData** (p. 847) object. Otherwise, the method will return an error code.

To determine the minimum size of the input buffer, you must call this method with the buffer set to null.

Parameters

buffer	<<***out***>> *(p. 156)*. Serialization buffer.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **DynamicData.to_cdr_buffer()**.

8.93.6.106 from_cdr_buffer() [1/2]

```
void from_cdr_buffer (
            byte[] buffer )
```

Deserializes a **DynamicData** (p. 847) object from a buffer of octets.

This method deserializes a **DynamicData** (p. 847) object from a CDR buffer of octets.

The content of the buffer generated by the method **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) can be provided to this method to get the **DynamicData** (p. 847) object back.

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

Parameters

buffer	<<***in***>> *(*p. *156)*. Deserialization buffer.

Exceptions

One	of the **Standard Return Codes** (p. 261)

Referenced by **BytesTypeSupport.data_to_string()**, **KeyedBytesTypeSupport.data_to_string()**, **KeyedString↩ TypeSupport.data_to_string()**, **StringTypeSupport.data_to_string()**, **DynamicData.from_cdr_buffer()**, and **DynamicDataTypeSupport.from_cdr_buffer()**.

8.93.6.107 to_cdr_buffer() [3/4]

```
void to_cdr_buffer (
             ByteSeq sequence,
            short representation )
```

Serializes a **DynamicData** (p. 847) object into a sequence of octets.

This method serializes a **DynamicData** (p. 847) object into a sequence of octets using the input data representation.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) method, except that it receives a **com.rti.dds.infrastructure.ByteSeq** (p. 395) as a serialization output.

This method may resize `sequence` as needed to fit the serialized **DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer (p. 946)

Parameters

sequence	<<***out***>> *(p. 156).* Serialization byte sequence.
representation	<<***in***>> *(p. 156).* Representation used to serialize the data.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **ByteSeq.getMaximum()**, **AbstractSequence.setMaximum()**, **AbstractPrimitiveSequence.setSize()**, and **DynamicData.to_cdr_buffer()**.

8.93.6.108 to_cdr_buffer() [4/4]

```
void to_cdr_buffer (
            ByteSeq sequence )
```

Serializes a **DynamicData** (p. 847) object into a CDR sequence of octets.

This method serializes a **DynamicData** (p. 847) object into a sequence of octets, using CDR as the data representation.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) method, except that it receives a **com.rti.dds.infrastructure.ByteSeq** (p. 395) as a serialization output.

This method may resize `sequence` as needed to fit the serialized **DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer (p. 946)

Parameters

sequence	<<***out***>> *(p. 156).* Serialization byte sequence.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **DynamicData.to_cdr_buffer()**.

8.93.6.109 from_cdr_buffer() [2/2]

```
void from_cdr_buffer (
            ByteSeq sequence )
```

Deserializes a **DynamicData** (p. 847) object from a sequence octets.

This method deserializes the a **DynamicData** (p. 847) object from a CDR sequence of octets.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.from_cdr_buffer** (p. 947) method, except that it receives a **com.rti.dds.infrastructure.ByteSeq** (p. 395) as deserialization input.

See also

com.rti.dds.dynamicdata.DynamicData.from_cdr_buffer (p. 947)

Parameters

sequence	<<***in***>> *(*p. *156)*. Deserialization octet sequence. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DynamicData.from_cdr_buffer()**.

8.93.6.110 to_string() [1/2]

```
String to_string (
            PrintFormatProperty property )
```

Get a string representation of a **DynamicData** (p. 847) object.

This method takes a dynamic data sample and creates a string representation of the data.

If the size of the output string is longer than the size of an unsigned 32-bit integer, this operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

This API is not supported when the **DynamicData** (p. 847) object is in CDR format and will fail with **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). See com.rti.dds.dynamicdata.DynamicData.is_cdr for more information.

Parameters

property	*<<**in**>> (p. 156).* Properties describing what the format of the output string should be. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) , java.lang.IllegalStateException

References **PrintFormatProperty.enum_as_int**, **PrintFormatProperty.include_root_elements**, **PrintFormat↩ Property.kind**, **Enum.ordinal()**, and **PrintFormatProperty.pretty_print**.

Referenced by **BytesTypeSupport.data_to_string()**, **KeyedBytesTypeSupport.data_to_string()**, **KeyedString↩ TypeSupport.data_to_string()**, and **StringTypeSupport.data_to_string()**.

8.93.6.111 to_string() [2/2]

```
String to_string ( )
```

Get a string representation of a **DynamicData** (p. 847) object using the default values for **com.rti.dds.topic.Print↩ FormatProperty** (p. 1387).

See also

com.rti.dds.dynamicdata.DynamicData.to_string (p. 949)

References **DynamicData.to_string()**.

Referenced by **DynamicData.to_string()**.

8.93.6.112 from_string() [1/2]

```
void from_string (
            String str,
             PrintFormatKind format_kind )
```

Populates a **DynamicData** (p. 847) object from a JSON string representation.

This method takes a JSON string representing a data sample of this type, and uses it to populate this **DynamicData** (p. 847).

Parameters

str	<<***in***>> *(p. 156).* The JSON string representation
format_kind	<<***in***>> *(p. 156).* Must be set to **com.rti.dds.topic.PrintFormatKind.JSON_PRINT_FORMAT** (p. 1386), the only currently supported format.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

References **Enum.ordinal()**.

Referenced by **DynamicData.from_string()**.

8.93.6.113 from_string() [2/2]

```
void from_string (
             String str )
```

Populates a **DynamicData** (p. 847) object from a JSON string representation.

This method is an overload that does not take a PrintFormatKind.

Parameters

str	<<***in***>> *(p. 156).* The JSON string representation

See also

com.rti.dds.dynamicdata.DynamicData.from_string (p. 950)

References **DynamicData.from_string()**, and **PrintFormatKind.JSON_PRINT_FORMAT**.

8.93.7 Member Data Documentation

8.93.7.1 MEMBER_ID_UNSPECIFIED

```
final int MEMBER_ID_UNSPECIFIED  [static]
```

A sentinel value that indicates that no member ID is needed in order to perform some operation.

Most commonly, this constant will be used in "get" operations to indicate that a lookup should be performed based on a name, not on an ID.

8.94 DynamicDataInfo Class Reference

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

Inherits Struct.

Public Member Functions

- **DynamicDataInfo** ()

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*
- **DynamicDataInfo** (int **member_count**, int **stored_size**, boolean is_optimized_storage)

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.*

Public Attributes

- int **member_count**

  *The number of data members in this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*
- int **stored_size**

  *The number of bytes currently used to store the data of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*

8.94.1 Detailed Description

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.get_info (p. 865)

8.94.2 Member Data Documentation

8.94.2.1 member_count

```
int member_count
```

The number of data members in this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

Referenced by **DynamicDataInfo.DynamicDataInfo()**.

8.94.2.2 stored_size

```
int stored_size
```

The number of bytes currently used to store the data of this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

Referenced by **DynamicDataInfo.DynamicDataInfo()**.

8.95 DynamicDataMemberInfo Class Reference

A descriptor for a single member (i.e. field) of dynamically defined data type.

Inherits Struct.

Public Member Functions

- **DynamicDataMemberInfo** ()

  A descriptor for a single member (i.e. field) of dynamically defined data type.
- **DynamicDataMemberInfo** (int **member_id**, String **member_name**, boolean **member_exists**, **TCKind member_kind**, int representation_count, int **element_count**, **TCKind element_kind**)

  A descriptor for a single member (i.e. field) of dynamically defined data type.

Public Attributes

- int **member_id**

  *An integer that uniquely identifies the data member within this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample's type.*
- String **member_name**

  The string name of the data member.
- boolean **member_exists**

  Indicates whether the member exists in this sample.
- **TCKind member_kind**

  The kind of type of this data member (e.g., integer, structure, etc.).
- int **element_count**

  The number of elements within this data member.
- **TCKind element_kind**

  The kind of type of the elements within this data member.

8.95.1 Detailed Description

A descriptor for a single member (i.e. field) of dynamically defined data type.

See also

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 871)

8.95.2 Member Data Documentation

8.95.2.1 member_id

```
int member_id
```

An integer that uniquely identifies the data member within this **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample's type.

The member ID is assigned automatically by the middleware based on the member's declaration order within the type. It is a 1-based index of the member's position in the type.

See also

com.rti.dds.typecode.TCKind (p. 1786)

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.95.2.2 member_name

```
String member_name
```

The string name of the data member.

This name will be unique among members of the same type. However, a single named member may have multiple type representations.

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.95.2.3 member_exists

```
boolean member_exists
```

Indicates whether the member exists in this sample.

A member that is defined in a type may not exist in a sample of that type in the following situations

- The member is optional and is unset
- Being part of a union, the discriminator doesn't select this member
- When getting the information about a sequence element, with index `i`, and `i` is greater than the current sequence length but smaller than its maximum length.

See also

com.rti.dds.dynamicdata.DynamicData.member_exists_in_type (p. 871)

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.95.2.4 member_kind

```
TCKind member_kind
```

The kind of type of this data member (e.g., integer, structure, etc.).

This is a convenience field; it is equivalent to looking up the member in the **com.rti.dds.typecode.TypeCode** (p. 1873) and getting the **com.rti.dds.typecode.TCKind** (p. 1786) from there.

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.95.2.5 element_count

```
int element_count
```

The number of elements within this data member.

This information is only valid for members of array or sequence types. Members of other types will always report zero (0) here.

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.95.2.6 element_kind

```
TCKind element_kind
```

The kind of type of the elements within this data member.

This information is only valid for members of array or sequence types. Members of other types will always report **com.rti.dds.typecode.TCKind.TK_NULL** (p. 1787) here.

Referenced by **DynamicDataMemberInfo.DynamicDataMemberInfo()**.

8.96 DynamicDataProperty_t Class Reference

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

Public Member Functions

- **DynamicDataProperty_t** ()

  The constructor.
- **DynamicDataProperty_t** (int **buffer_initial_size**, int **buffer_max_size**, int buffer_max_size_increment, Charset **string_character_encoding**)

  Constructor accepting an alternative string encoding.
- **DynamicDataProperty_t** (int **buffer_initial_size**, int **buffer_max_size**, int buffer_max_size_increment)

  The constructor.

Public Attributes

- int **buffer_initial_size** = 0

  *The initial amount of memory used by the underlying **com.rti.dds.dynamicdata.DynamicData*** (p. *847) buffer, in bytes.*
- int **buffer_max_size** = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**

  *The maximum amount of memory that the underlying **com.rti.dds.dynamicdata.DynamicData*** (p. *847) buffer may use, in bytes.*
- Charset **string_character_encoding** = StandardCharsets.UTF_8

  *String encoding that this **com.rti.dds.dynamicdata.DynamicData*** (p. *847) object will use internally.*

8.96.1 Detailed Description

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

8.96.2 Constructor & Destructor Documentation

8.96.2.1 DynamicDataProperty_t() [1/3]

```
DynamicDataProperty_t ( )
```

The constructor.

8.96.2.2 DynamicDataProperty_t() [2/3]

```
DynamicDataProperty_t (
            int buffer_initial_size,
            int buffer_max_size,
            int buffer_max_size_increment,
            Charset string_character_encoding )
```

Constructor accepting an alternative string encoding.

References **DynamicDataProperty_t.buffer_initial_size**, **DynamicDataProperty_t.buffer_max_size**, and **DynamicDataProperty_t.string_character_encoding**.

8.96.2.3 DynamicDataProperty_t() [3/3]

```
DynamicDataProperty_t (
            int buffer_initial_size,
            int buffer_max_size,
            int buffer_max_size_increment )
```

The constructor.

References **DynamicDataProperty_t.buffer_initial_size**, and **DynamicDataProperty_t.buffer_max_size**.

8.96.3 Member Data Documentation

8.96.3.1 buffer_initial_size

```
int buffer_initial_size = 0
```

The initial amount of memory used by the underlying **com.rti.dds.dynamicdata.DynamicData** (p. 847) buffer, in bytes.

This property is used to configure the **DynamicData** (p. 847) objects that are created stand-alone as well as the **DynamicData** (p. 847) samples that are obtained from the sample pool that is created by each **DynamicData** (p. 847) DataReader.

If set to 0 (default): The initial buffer size will be set to the minimum amount of space required to hold the overhead required by the **DynamicData** (p. 847) internal representation (about 100 bytes) in addition to the minimum deserialized size of a sample. The minimum deserialized size of a sample assumes that all strings are allocated to their default values, sequences are left to length 0, and all optional members are unset.

If set to any value other than 0: The underlying buffer will be allocated to the provided size plus the overhead required by the **DynamicData** (p. 847) internal representation (about 100 bytes). If the provided size plus the overhead is less than the size used when buffer_initial_size is left to 0, then the default value is used.

See also

com.rti.dds.dynamicdata.DynamicDataProperty_t.buffer_max_size (p. 957)

Referenced by **DynamicDataProperty_t.DynamicDataProperty_t()**, **DynamicDataTypeProperty_t.DynamicData↩TypeProperty_t()**, and **DynamicDataTypeSupport.DynamicDataTypeSupport()**.

8.96.3.2 buffer_max_size

```
int buffer_max_size =  ResourceLimitsQosPolicy.LENGTH_UNLIMITED
```

The maximum amount of memory that the underlying **com.rti.dds.dynamicdata.DynamicData** (p. 847) buffer may use, in bytes.

This property is used to configure the **DynamicData** (p. 847) objects that are created stand-alone as well as the **DynamicData** (p. 847) samples that are obtained from the sample pool that is created by each **DynamicData** (p. 847) DataReader.

A **DynamicData** (p. 847) object will grow to this size from the initial size as needed. The buffer_max_size includes all overhead that is required for the internal **DynamicData** (p. 847) representation and therefore represents a hard upper limit on the size of the underlying **DynamicData** (p. 847) buffer.

If set to -1 (default): The buffer will grow unbounded to the size required to fit all members.

If set to any value other than -1: The buffer will not grow beyond this size. If setting a member's values requires the buffer to grow beyond this maximum, the member will fail to be set. If the buffer is required to grow beyond this maximum during deserialization, the sample will fail to be deserialized. The buffer_max_size cannot be smaller than the buffer_initial_size.

See also

com.rti.dds.dynamicdata.DynamicDataProperty_t.buffer_initial_size (p. 957)

Referenced by **DynamicDataProperty_t.DynamicDataProperty_t()**, **DynamicDataTypeProperty_t.DynamicData↩ TypeProperty_t()**, and **DynamicDataTypeSupport.DynamicDataTypeSupport()**.

8.96.3.3 string_character_encoding

```
Charset string_character_encoding = StandardCharsets.UTF_8
```

String encoding that this **com.rti.dds.dynamicdata.DynamicData** (p. 847) object will use internally.

This **com.rti.dds.dynamicdata.DynamicData** (p. 847) object will store IDL strings with the encoding configured by this field.

In addition, IDL strings will be serialized with this encoding when sent on the wire.

Currently, only encodings ISO-8859-1 and UTF-8 are supported.

In Java 1.7 and above, the Charset objects corresponding to these encodings can be obtained from java.nio.charset.↩ StandardCharsets.

In previous Java versions, you can invoke Charset.availableCharsets().get(name) where name can be ISO-8859-1 and UTF-8.

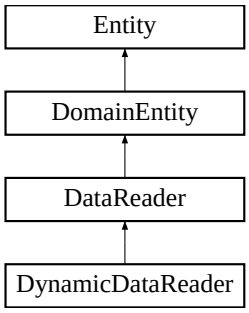
[default] UTF_8

Referenced by **DynamicDataProperty_t.DynamicDataProperty_t()**.

8.97 DynamicDataReader Class Reference

Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).

Inheritance diagram for DynamicDataReader:



Public Member Functions

- void **read** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **take** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data-samples from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **read_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.read*** (p. *1069*) *the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition*** (p. *1514*).
- void **take_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Analogous to **com.rti.ndds.example.FooDataReader.read_w_condition*** (p. *1076*) *except it accesses samples via the **com.rti.ndds.example.FooDataReader.take*** (p. *1071*) *operation.*
- void **read_next_sample** (**DynamicData** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **take_next_sample** (**DynamicData** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **read_instance** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **take_instance** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader*** (p. *450*).
- void **read_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_↩ samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  <<***extension***>> (p. *155*) *Accesses via **com.rti.ndds.example.FooDataReader.read_instance*** (p. *1081*) *the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition*** (p. *1514*).
- void **take_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_↩ samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

*<<**extension**>> (p. 155) Accesses via **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **read_next_instance** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_next_instance** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_next_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_next_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **return_loan** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq)

  *Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.↩ subscription.DataReader** (p. 450).*
- void **get_key_value** (**DynamicData** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- **InstanceHandle_t lookup_instance** (**DynamicData** key_holder)

  Retrieves the instance `handle` that corresponds to an instance `key_holder`.

Additional Inherited Members

8.97.1 Detailed Description

Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).

Instantiates **`com.rti.dds.subscription.DataReader`** (p. 450) < **com.rti.dds.dynamicdata.Dynamic↩ Data** (p. 847) > .

See also

com.rti.dds.subscription.DataReader (p. 450)

com.rti.ndds.example.FooDataReader (p. 1067)

com.rti.dds.dynamicdata.DynamicData (p. 847)

8.97.2 Member Function Documentation

8.97.2.1 read()

```
void read (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation offers the same functionality and API as **com.rti.ndds.example.FooDataReader.take** (p. 1071) except that the samples returned remain in the **com.rti.dds.subscription.DataReader** (p. 450) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **com.rti.ndds.example.FooDataReader.take()** (p. 1071) for details on the number of samples returned within the received_data and info_seq as well as the order in which the samples appear in these sequences.

The act of reading a sample changes its `sample_state` to com.rti.dds.subscription.SampleStateKind.SampleState↩ Kind.READ_SAMPLE_STATE. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to be com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the instance.

Once the application completes its use of the samples, it must 'return the loan' to the **com.rti.dds.subscription.Data↩ Reader** (p. 450) by calling the **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **com.rti.dds.subscription.SampleInfo** (p. 1634) objects after the call to **com.rti.ndds.example.FooDataReader.↩ return_loan** (p. 1094). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While the loan must be returned at some point, you do *not* have to do so before the next **com.rti.ndds.↩ example.FooDataReader.take** (p. 1071) call. However, failure to return the loan will eventually deplete the **com.rti.↩ dds.subscription.DataReader** (p. 450) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **com.rti.dds.subscription.DataReader** (p. 450) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) and the **com.rti.dds.infrastructure.DataReader↩ ResourceLimitsQosPolicy** (p. 529).

Important: If the samples "returned" by this method are loaned from RTI Connext (see **com.rti.ndds.example.Foo↩ DataReader.take** (p. 1071) for more information on memory loaning), it is important that their contents not be changed. Because the memory in which the data is stored belongs to the middleware, any modifications made to the data will be seen the next time the same samples are read or taken; the samples will no longer reflect the state that was received from the network.

Parameters

received_data	<<***inout***>> *(p. 156)* User data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

info_seq	<<***inout***>> *(p. 156)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
sample_states	<<***in***>> *(p. 156)* Data samples matching one of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(p. 156)* Data samples matching one of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* Data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076),

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_untyped()**.

8.97.2.2 take()

```
void take (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

The operation will return the list of samples received by the **com.rti.dds.subscription.DataReader** (p. 450) since the last **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation that matches the specified com.rti.dds.↩ subscription.SampleStateMask, com.rti.dds.subscription.ViewStateMask and com.rti.dds.subscription.InstanceState↩ Mask.

This operation may fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595) if the **com.rti.dds.↩ infrastructure.DataReaderResourceLimitsQosPolicy.max_outstanding_reads** (p. 534) limit has been exceeded.

The actual number of samples returned depends on the information that has been received by the middleware as well as the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590), **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529) and the characteristics of the data-type that is associated with the **com.rti.dds.subscription.DataReader** (p. 450):

- In the case where the **com.rti.dds.infrastructure.HistoryQosPolicy.kind** (p. 1146) is com.rti.dds.↩ infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, the call will return at most **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) samples for each ALIVE instance and (**com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) +1) samples for each NOT_ALIVE instance. The extra sample is an invalid sample (**com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643) is FALSE) that is used to indicate the instance state transition from ALIVE to NOT_ALIVE.

- The maximum number of samples returned is limited by **com.rti.dds.infrastructure.ResourceLimitsQos↩ Policy.max_samples** (p. 1592), and by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.↩ max_samples_per_read** (p. 534).

- For multiple instances, the number of samples returned is additionally limited by the product (**com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)
  - **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592))

- If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_infos** (p. 532) is limited, the number of samples returned may also be limited if insufficient **com.rti.dds.subscription.SampleInfo** (p. 1634) resources are available.

If the read or take succeeds and the number of samples returned has been limited (by means of a maximum limit, as listed above, or insufficient **com.rti.dds.subscription.SampleInfo** (p. 1634) resources), the call will complete successfully and provide those samples the reader is able to return. The user may need to make additional calls, or return outstanding loaned buffers in the case of insufficient resources, in order to access remaining samples.

Note that in the case where the **com.rti.dds.topic.Topic** (p. 1807) associated with the **com.rti.dds.subscription.↩ DataReader** (p. 450) is bound to a data-type that has no key definition, then there will be at most one instance in the **com.rti.dds.subscription.DataReader** (p. 450). So the per-sample limits will apply.

The act of *taking* a sample removes it from RTI Connext so it cannot be read or taken again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the sample's instance to com.rti.dds.↩ subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the sample's instance.

After **com.rti.ndds.example.FooDataReader.take** (p. 1071) completes, `received_data` and `info_seq` will be of the same length and contain the received data.

If the sequences are empty (maximum size equals 0) when the **com.rti.ndds.example.FooDataReader.take** (p. 1071) is called, the samples returned in the `received_data` and the corresponding `info_seq` are 'loaned' to the application from buffers provided by the **com.rti.dds.subscription.DataReader** (p. 450). The application can use them as desired and has guaranteed exclusive access to them.

Once the application completes its use of the samples it must 'return the loan' to the **com.rti.dds.subscription.Data↩ Reader** (p. 450) by calling the **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **com.rti.dds.subscription.SampleInfo** (p. 1634) objects after the call to **com.rti.ndds.example.FooDataReader.↩ return_loan** (p. 1094). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While you must call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) at some point, you do *not* have to do so before the next **com.rti.ndds.example.FooDataReader.take** (p. 1071) call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader** (p. 450) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **com.rti.dds.↩ subscription.DataReader** (p. 450) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when **com.rti.ndds.example.↩ FooDataReader.take** (p. 1071) is called, samples are copied to received_data and info_seq. The application will not need to call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

The order of the samples returned to the caller depends on the **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1379).

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION↩ _QOS, the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.true, then the returned collection is a list where the relative order of samples as they were written by a DataWriter is preserved also across different instances. In other words, changes made by a single DataWriter are made available to subscribers in the same order in which they occur. Samples belonging to the same instance may or may not be consecutive. This is because, to preserve order within a single DataWriter, it may be necessary to mix samples from different instances.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.true, then changes made to instances by a set of DataWriters attached to a common Publisher are made available in the order in which they were written. For this to happen, the application must take the samples using the Subscriber's **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) and **com.↩ rti.dds.subscription.Subscriber.end_access** (p. 1750) operations (see the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the `Core Libraries User's Manual`).

In all of the above cases, the relative order between the samples of one instance is consistent with the **DESTINATION↩ _ORDER** (p. 218) policy:

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is com.rti.dds.infrastructure.↩ DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER↩ _QOS, samples belonging to the same instances will appear in the relative order in which there were received (FIFO, earlier samples ahead of the later samples).

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is com.rti.dds.infrastructure.↩ DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER↩ _QOS, samples belonging to the same instances will appear in the relative order implied by the `source_↩ timestamp` (FIFO, smaller values of `source_timestamp` ahead of the larger values).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

In addition to the collection of samples, the read and take operations also use a collection of **com.rti.dds.↩ subscription.SampleInfo** (p. 1634) structures.

The initial (input) properties of the `received_data` and `info_seq` collections will determine the precise behavior of the read or take operation. For the purposes of this description, the collections are modeled as having these properties:

- the current-length (`len`, see com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size())
- the maximum length (`max_len`, see com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.get↩ Maximum)

The initial values of the `len` and `max_len` properties for the `received_data` and `info_seq` collections govern the behavior of the read and take operations as specified by the following rules:

1. The values of `len` and `max_len` properties for the two collections must be identical. Otherwise read/take will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).
2. On successful output, the values of `len` and `max_len` will be the same for both collections.
3. If the initial `max_len==0`, then the `received_data` and `info_seq` collections will be filled with elements that are loaned by the **com.rti.dds.subscription.DataReader** (p. 450). On output, `len` will be set to the number of values returned, and `max_len` will be set to a value verifying `max_len >= len`. The use of this variant allows for Zero Copy access to the data and the application will need to return the loan to the **com.rti.dds.↩ publication.DataWriter** (p. 553) using **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

4. If initial max_len>0 then the read or take operation will copy the received_data values and **com.rti.dds.subscription.SampleInfo** (p. 1634) values into the elements already inside the collections. On output, len will be set to the number of values copied and max_len will remain unchanged. The use of this variant forces a copy but the application can control where the copy is placed and the application will not need to return the loan. The number of samples copied depends on the relative values of max_len and max_samples:

   - If max_samples == LENGTH_UNLIMITED, then at most max_len values will be copied. The use of this variant lets the application limit the number of samples returned to what the sequence can accommodate.
   - If max_samples <= max_len, then at most max_samples values will be copied. The use of this variant lets the application limit the number of samples returned to fewer that what the sequence can accommodate.
   - If max_samples > max_len, then the read or take operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598). This avoids the potential confusion where the application expects to be able to access up to max_samples, but that number can never be returned, even if they are available in the **com.rti.dds.subscription.DataReader** (p. 450), because the output sequence cannot accommodate them.

As described above, upon completion, the received_data and info_seq collections may contain elements loaned from the **com.rti.dds.subscription.DataReader** (p. 450). If this is the case, the application will need to use **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) to return the loan once it is no longer using the received_data in the collection. When **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) completes, the collection will have max_len=0. The application can determine whether it is necessary to return the loan or not based on how the state of the collections when the read/take operation was called However, in many cases it may be simpler to always call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094), as this operation is harmless (i.e., it leaves all elements unchanged) if the collection does not have a loan.

On output, the collection of **com.rti.ndds.example.Foo** (p. 1066) values and the collection of **com.rti.dds.subscription.SampleInfo** (p. 1634) structures are of the same length and are in a one-to-one correspondence. Each **com.rti.dds.subscription.SampleInfo** (p. 1634) provides information, such as the source_timestamp, the sample_state, view_state, and instance_state, etc., about the corresponding sample. Some elements in the returned collection may not have valid data. If the instance_state in the **com.rti.dds.subscription.SampleInfo** (p. 1634) is **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) or **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161), then the last sample for that instance in the collection (that is, the one whose **com.rti.dds.subscription.SampleInfo** (p. 1634) has sample_rank==0) does not contain valid data.

Samples that contain no data do not count towards the limits imposed by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590). The act of reading/taking a sample sets its sample_state to com.rti.dds.subscription.SampleStateKind.SampleStateKind.READ_SAMPLE_STATE.

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the instance_state of the instance.

This operation must be provided on the specialized class that is generated for the particular application data-type that is being read (**com.rti.ndds.example.Foo** (p. 1066)).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the operations fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). For an example on how take can be used, please refer to the **Access received data via a reader** (p. 134) "receive example".

Parameters

received_data	<<***inout***>> *(*p. *156)* User data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(*p. *156)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described above.
sample_states	<<***in***>> *(*p. *156)* Data samples matching one of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(*p. *156)* Data samples matching one of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(*p. *156)* Data samples matching one of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076), **com.rti.ndds.example.FooData↩ Reader.take_w_condition** (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_untyped()**.

8.97.2.3 read_w_condition()

```
void read_w_condition (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.read** (p. 1069) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1510) to filter data samples based on the content.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1514) must be attached to the **com.rti.dds.↩ subscription.DataReader** (p. 450); otherwise the operation will fail with **com.rti.dds.infrastructure.RETCODE_↩ PRECONDITION_NOT_MET** (p. 1598).

In case the **com.rti.dds.subscription.ReadCondition** (p. 1514) is a plain **com.rti.dds.subscription.ReadCondition** (p. 1514) and not the specialized **com.rti.dds.subscription.QueryCondition** (p. 1510), the operation is equivalent to calling **com.rti.ndds.example.FooDataReader.read** (p. 1069) and passing as `sample_states`, `view_states` and `instance_states` the value of the corresponding attributes in the `read_condition`. Using this operation, the application can avoid repeating the same parameters specified when creating the **com.rti.dds.subscription.↩ ReadCondition** (p. 1514).

The samples are accessed with the same semantics as **com.rti.ndds.example.FooDataReader.read** (p. 1069).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the operation will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.ndds.example.FooDataReader.take (p. 1071), **com.rti.ndds.example.FooDataReader.take_w_↩ condition** (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_w_condition_untyped()**.

8.97.2.4 take_w_condition()

```
void take_w_condition (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) except it accesses samples via the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation.

This operation is analogous to **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) except that it accesses samples via the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1514) must be attached to the **com.rti.dds.↩ subscription.DataReader** (p. 450); otherwise the operation will fail with **com.rti.dds.infrastructure.RETCODE_↩ PRECONDITION_NOT_MET** (p. 1598).

The samples are accessed with the same semantics as **com.rti.ndds.example.FooDataReader.take** (p. 1071).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1510) to filter data samples based on the content.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076), **com.rti.ndds.example.FooDataReader.read** (p. 1069)

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_w_condition_untyped()**.

8.97.2.5 read_next_sample()

```
void read_next_sample (
            DynamicData received_data,
            SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450). This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634). The implied order among the samples stored in the **com.rti.dds.subscription.DataReader** (p. 450) is the same as for the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation.

The **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) operation provides a simplified API to 'read' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **com.rti.dds.subscription.DataReader** (p. 450), the operation will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) and nothing is copied.

Note

Calling **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) from the **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) callback reads only one sample of potentially many samples in the reader queue, because **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) in a loop within the on_data_available callback until **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) returns **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use **com.rti.ndds.example.FooDataReader.read** (p. 1069) rather than **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) in order to read more than one sample at a time.)

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific **com.rti.ndds.example.Foo** (p. 1066) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL Foo. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_info	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfo** (p. 1634) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1634). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_next_sample_untyped()**.

8.97.2.6 take_next_sample()

```
void take_next_sample (
              DynamicData received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' it from the **com.rti.dds.subscription.DataReader** (p. 450) so that it is no longer accessible. This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634). This operation is analogous to the **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) except for the fact that the sample is removed from the **com.rti.dds.subscription.DataReader** (p. 450).

The **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) operation provides a simplified API to 'take' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **com.rti.dds.subscription.DataReader** (p. 450), the operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_NO_DATA** (p. 1597) and nothing is copied.

Note

Calling **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) from the **com.rti.dds.↩ subscription.DataReaderListener.on_data_available()** (p. 500) callback retrieves only one sample of potentially many samples in the reader queue, because **com.rti.dds.subscription.DataReaderListener.on_↩ data_available()** (p. 500) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) in a loop within the on_data_available callback until **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) returns **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use the **com.rti.ndds.example.FooDataReader.take** (p. 1071) rather than **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) in order to take more than one sample at a time.)

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific **com.rti.ndds.example.Foo** (p. 1066) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL Foo. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_info	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfo** (p. 1634) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1634). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261),

com.rti.dds.infrastructure.RETCODE_NO_DATA (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_↩ ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_next_sample_untyped()**.

8.97.2.7 read_instance()

```
void read_instance (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read** (p. 1069), except that all samples returned belong to the single specified instance whose handle is `a_handle`.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634) verifies **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1640) == `a_handle`.

The **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 450) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The behavior of the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	<<***in***>> *(p. 156)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).
sample_states	<<***in***>> *(p. 156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).

Parameters

view_states	<<***in***>> *(p. 156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_instance_untyped()**.

8.97.2.8 take_instance()

```
void take_instance (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.take** (p. 1071), except for that all samples returned belong to the single specified instance whose handle is `a_handle`.

The semantics are the same for the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 450) will check that the sample belongs to the specified instance, and otherwise it will not place the sample in the returned collection.

The behavior of the **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.↩ read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) operation may 'loan' elements to

the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	<<***in***>> *(p. 156)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).
sample_states	<<***in***>> *(p. 156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(p. 156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_instance_untyped()**.

8.97.2.9 read_instance_w_condition()

```
void read_instance_w_condition (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

<<***extension***>> *(p. 155)* Accesses via **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

The behavior of the **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooData↩ Reader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.Foo↩ DataReader.return_loan** (p. 1094).

Similar to **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_instance_w_condition_untyped()**.

8.97.2.10 take_instance_w_condition()

```
void take_instance_w_condition (
               DynamicDataSeq received_data,
               SampleInfoSeq info_seq,
              int max_samples,
               InstanceHandle_t a_handle,
               ReadCondition condition )
```

*<<**extension**>> (p. 155)* Accesses via **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

The operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

The behavior of the **com.rti.ndds.example.FooDataReader.take_instance_w_condition** (p. 1089) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooData↩ Reader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_instance_w_condition** (p. 1089) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.Foo↩ DataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_instance_w_condition_untyped()**.

8.97.2.11 read_next_instance()

```
void read_next_instance (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) where all the samples belong to a single instance. The behavior is similar to **com.rti.ndds.example.FooDataReader.read_↩ instance** (p. 1081), except that the actual instance is not directly specified. Rather, the samples will all belong to the 'next' instance with `instance_handle` 'greater' than the specified 'previous_handle' that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance handles that do not correspond to instances currently managed by the **com.rti.dds.subscription.DataReader** (p. 450). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) is 'as if' the **com.rti.dds.↩ subscription.DataReader** (p. 450) invoked **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **com.rti.dds.↩ infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

> The operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **com.rti.↩ dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), examines the samples returned, and then uses the `instance_handle` returned in the **com.rti.dds.subscription.SampleInfo** (p. 1634) as the value of the `previous_handle` argument to the next call to **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084). The iteration continues until **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) operation with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.↩ subscription.DataReader** (p. 450). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **com.rti.dds.subscription.DataReader** (p. 450). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **com.rti.dds.subscription.↩ InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance.

The behavior of the **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	<<***in***>> *(p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_states	<<***in***>> *(p. 156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(p. 156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_next_instance_untyped()**.

8.97.2.12 take_next_instance()

```
void take_next_instance (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Note

Like **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), this operation is intended to be used in an application-driven iteration until all samples on the reader queue are taken. The iteration continues until **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) fails with the value **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) .

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> (p. *156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> (p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	<<***in***>> (p. *156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_states	<<***in***>> (p. *156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> (p. *156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> (p. *156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_next_instance_untyped()**.

8.97.2.13 read_next_instance_w_condition()

```
void read_next_instance_w_condition (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
```

```
InstanceHandle_t previous_handle,
ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to the same instance, and the instance is the instance with 'smallest' `instance_handle` among the ones that verify: (a) `instance_handle` >= `previous_handle`, and (b) have samples for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	<<***in***>> *(p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	<<***in***>> *(p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_next_instance_w_condition_untyped()**.

8.97.2.14 take_next_instance_w_condition()

```
void take_next_instance_w_condition (
              DynamicDataSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
              ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450).

The operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_next_instance_w_condition_untyped()**.

8.97.2.15 return_loan()

```
void return_loan (
            DynamicDataSeq received_data,
            SampleInfoSeq info_seq )
```

Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.↩ subscription.DataReader** (p. 450).

This operation indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450).

The `received_data` and `info_seq` must belong to a single related "pair"; that is, they should correspond to a pair returned from a single call to read or take. The `received_data` and `info_seq` must also have been obtained from the same **com.rti.dds.subscription.DataReader** (p. 450) to which they are returned. If either of these conditions is not met, the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

The operation **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) allows implementations of the read and take operations to "loan" buffers from the **com.rti.dds.subscription.DataReader** (p. 450) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **com.rti.dds.subscription.DataReader** (p. 450) will guarantee that the data and sample-information are not modified.

It is not necessary for an application to return the loans immediately after the read or take calls. However, as these buffers correspond to internal resources inside the **com.rti.dds.subscription.DataReader** (p. 450), the application should not retain them indefinitely.

The use of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) is only necessary if the read or take calls "loaned" buffers to the application. This only occurs if the `received_data` and `info_Seq` collections had `max_len=0` at the time read or take was called.

If the collections had a loan, upon completion of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094), the collections will have `max_len=0`.

Similar to read, this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

Parameters

received_data	*<<**in**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450). Must be a valid non-NULL FooSeq. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

info_seq	*<<**in**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450). Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

References **DataReader.return_loan_untyped()**.

8.97.2.16 get_key_value()

```
void get_key_value (
                DynamicData key_holder,
                InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

key_holder	<<***inout***>> *(p. 156)* a user data type specific key holder, whose `key` fields are filled by this operation. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `key_holder` is NULL.
handle	<<***in***>> *(p. 156)* the `instance` whose key is to be retrieved. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` must represent an existing instance of type **com.rti.ndds.example.Foo** (p. 1066) known to the **com.rti.dds.subscription.DataReader** (p. 450). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

If **com.rti.ndds.example.Foo** (p. 1066) has a key and `handle` represents an instance of another type or an instance of type **com.rti.ndds.example.Foo** (p. 1066) that has been unregistered, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataReader.get_key_value_untyped()**.

8.97.2.17 lookup_instance()

```
InstanceHandle_t lookup_instance (
            DynamicData key_holder )
```

Retrieves the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. If the instance is unknown to the DataReader, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

Parameters

key_holder	*<<**in**>> (p. 156)* a user data type specific key holder.

Returns

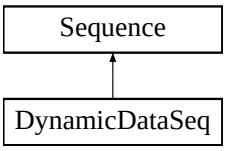
the instance handle associated with this instance. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect and returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156)

References **DataReader.lookup_instance_untyped()**.

8.98 DynamicDataSeq Class Reference

An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 847) elements.

Inheritance diagram for DynamicDataSeq:



Public Member Functions

- **DynamicDataSeq** ()

  *Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988).*
- **DynamicDataSeq** (int initialMaximum)

  *Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988).*
- **DynamicDataSeq** (Collection elements)

  *Construct a new **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988) containing the same elements as the given collection.*

8.98.1 Detailed Description

An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 847) elements.

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.dynamicdata.↩ DynamicData** (p. 847) > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

com.rti.dds.dynamicdata.DynamicData (p. 847)

`http://java.sun.com/javase/6/docs/api/java/util/List.html`

8.98.2 Constructor & Destructor Documentation

8.98.2.1 DynamicDataSeq() [1/3]

```
DynamicDataSeq ( )
```

Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988).

8.98.2.2 DynamicDataSeq() [2/3]

```
DynamicDataSeq (
            int initialMaximum )
```

Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988).

The new sequence will have the given maximum. (The *maximum* of the sequence is equivalent to what, in an `Array↩ List`, is called the *capacity*.)

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.getMaximum

`http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html`

8.98.2.3 DynamicDataSeq() [3/3]

```
DynamicDataSeq (
            Collection elements )
```

Construct a new **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 988) containing the same elements as the given collection.

See also

http://java.sun.com/javase/6/docs/api/java/util/Collection.html

8.99 DynamicDataTypeProperty_t Class Reference

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

Public Member Functions

- **DynamicDataTypeProperty_t** ()

  The constructor.
- **DynamicDataTypeProperty_t** (**DynamicDataProperty_t data**, **DynamicDataTypeSerializationProperty_t serialization**)

  The constructor.

Public Attributes

- final **DynamicDataProperty_t data**

  *These properties will be provided to every new **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample created from the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995).*
- final **DynamicDataTypeSerializationProperty_t serialization**

  Properties that govern how the data of this type will be serialized on the network.

8.99.1 Detailed Description

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

8.99.2 Constructor & Destructor Documentation

8.99.2.1 DynamicDataTypeProperty_t() [1/2]

```
DynamicDataTypeProperty_t ( )
```

The constructor.

8.99.2.2 DynamicDataTypeProperty_t() [2/2]

```
DynamicDataTypeProperty_t (
            DynamicDataProperty_t data,
            DynamicDataTypeSerializationProperty_t serialization )
```

The constructor.

References **DynamicDataProperty_t.buffer_initial_size**, **DynamicDataProperty_t.buffer_max_size**, **DynamicDataProperty_t.data**, **DynamicDataTypeSerializationProperty_t.max_size_serialized**, **DynamicDataTypeSerializationProperty_t.min_size_serialized**, **DynamicDataTypeProperty_t.serialization**, **DynamicDataTypeSerializationProperty_t.trim_to_size**, and **DynamicDataTypeSerializationProperty_t.use_42e_compatible_alignment**.

8.99.3 Member Data Documentation

8.99.3.1 data

```
final DynamicDataProperty_t data
```

Initial value:

```
=
        new DynamicDataProperty_t()
```

These properties will be provided to every new **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample created from the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995).

These properties are also used to create the samples that are kept in the DataReader's queue.

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, and **DynamicDataTypeSupport.DynamicDataTypeSupport()**.

8.99.3.2 serialization

```
final DynamicDataTypeSerializationProperty_t serialization
```

Initial value:

```
=
        new DynamicDataTypeSerializationProperty_t()
```

Properties that govern how the data of this type will be serialized on the network.

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, and **DynamicDataTypeSupport.↩ DynamicDataTypeSupport()**.

8.100 DynamicDataTypeSerializationProperty_t Class Reference

Properties that govern how data of a certain type will be serialized on the network.

Public Member Functions

- **DynamicDataTypeSerializationProperty_t** ()

  The constructor.
- **DynamicDataTypeSerializationProperty_t** (boolean **use_42e_compatible_alignment**, int **max_size_↩ serialized**, int **min_size_serialized**, boolean **trim_to_size**)

  The constructor.

Public Attributes

- boolean **use_42e_compatible_alignment** = false

  *[**No longer supported**] Use RTI Connext 4.2e-compatible alignment for large primitive types.*
- int **max_size_serialized** = 0xffffffff

  *[**No longer supported**] If you were previously using this property, use **com.rti.dds.dynamicdata.DynamicData↩ Property_t.buffer_max_size** (p. 957) instead. The maximum number of bytes that objects of a given type could consume when serialized on the network.*
- int **min_size_serialized** = 0xffffffff

  *[**No longer supported**] If you were previously using this property, use **com.rti.dds.dynamicdata.DynamicData↩ Property_t.buffer_initial_size** (p. 957) instead. The minimum number of bytes that objects of a given type could consume when serialized on the network.*
- boolean **trim_to_size** = false

  *Controls the growth of the buffer in a **DynamicData** (p. 847) object.*

8.100.1 Detailed Description

Properties that govern how data of a certain type will be serialized on the network.

8.100.2 Constructor & Destructor Documentation

8.100.2.1 DynamicDataTypeSerializationProperty_t() [1/2]

```
DynamicDataTypeSerializationProperty_t ( )
```

The constructor.

8.100.2.2 DynamicDataTypeSerializationProperty_t() [2/2]

```
DynamicDataTypeSerializationProperty_t (
            boolean use_42e_compatible_alignment,
            int max_size_serialized,
            int min_size_serialized,
            boolean trim_to_size )
```

The constructor.

References **DynamicDataTypeSerializationProperty_t.max_size_serialized**, **DynamicDataTypeSerialization↩ Property_t.min_size_serialized**, **DynamicDataTypeSerializationProperty_t.trim_to_size**, and **DynamicData↩ TypeSerializationProperty_t.use_42e_compatible_alignment**.

8.100.3 Member Data Documentation

8.100.3.1 use_42e_compatible_alignment

```
boolean use_42e_compatible_alignment = false
```

[**No longer supported**] Use RTI Connext 4.2e-compatible alignment for large primitive types.

In RTI Connext 4.2e, the default alignment for large primitive types – long, com.rti.dds.infrastructure.long, double, and com.rti.dds.infrastructure.LongDouble – was not RTPS-compliant. This compatibility mode allows applications targeting post-4.2e versions of RTI Connext to interoperate with 4.2e-based applications, regardless of the data types they use.

If this flag is not set, all data will be serialized in an RTPS-compliant manner, which for the types listed above, will not be interoperable with RTI Connext 4.2e.

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, **DynamicDataTypeSerialization↩ Property_t.DynamicDataTypeSerializationProperty_t()**, and **DynamicDataTypeSupport.DynamicDataType↩ Support()**.

8.100.3.2 max_size_serialized

```
int max_size_serialized = 0xffffffff
```

[**No longer supported**] If you were previously using this property, use **com.rti.dds.dynamicdata.DynamicData↩ Property_t.buffer_max_size** (p. 957) instead. The maximum number of bytes that objects of a given type could consume when serialized on the network.

This value is used to set the sizes of certain internal middleware buffers.

The effective value of the maximum serialized size will be the value of this field or the size automatically inferred from the type's **com.rti.dds.typecode.TypeCode** (p. 1873), whichever is smaller.

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, **DynamicDataTypeSerialization↩ Property_t.DynamicDataTypeSerializationProperty_t()**, and **DynamicDataTypeSupport.DynamicDataType↩ Support()**.

8.100.3.3 min_size_serialized

```
int min_size_serialized = 0xffffffff
```

[**No longer supported**] If you were previously using this property, use **com.rti.dds.dynamicdata.DynamicData↩ Property_t.buffer_initial_size** (p. 957) instead. The minimum number of bytes that objects of a given type could consume when serialized on the network.

This value is used to set the sizes of certain internal middleware buffers.

Default: 0xFFFFFFFF, a sentinel that indicates that this value must be equal to the value specified in max_size_↩ serialized.

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, **DynamicDataTypeSerialization↩ Property_t.DynamicDataTypeSerializationProperty_t()**, and **DynamicDataTypeSupport.DynamicDataType↩ Support()**.

8.100.3.4 trim_to_size

```
boolean trim_to_size = false
```

Controls the growth of the buffer in a **DynamicData** (p. 847) object.

This property only applies to **DynamicData** (p. 847) samples that are obtained from the sample pool that is created by each **DynamicData** (p. 847) DataReader.

If set to 0 (default): The buffer will not be reallocated unless the deserialized size of the incoming sample is greater than the current buffer size.

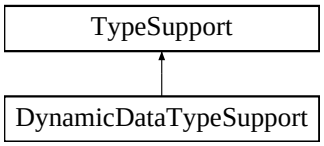
If set to 1: The buffer of a **DynamicData** (p. 847) object obtained from the DDS sample pool will be freed and re-allocated for each sample to just fit the size of the deserialized data of the incoming sample. The newly allocated size will not be smaller than **com.rti.dds.dynamicdata.DynamicDataProperty_t.buffer_initial_size** (p. 957).

Referenced by **DynamicDataTypeProperty_t.DynamicDataTypeProperty_t()**, **DynamicDataTypeSerialization↩ Property_t.DynamicDataTypeSerializationProperty_t()**, and **DynamicDataTypeSupport.DynamicDataType↩ Support()**.

8.101 DynamicDataTypeSupport Class Reference

A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

Inheritance diagram for DynamicDataTypeSupport:



Public Member Functions

- final void **register_type** (**DomainParticipant** participant, String type_name)

  *Associate the **com.rti.dds.typecode.TypeCode** (p. 1873) with the given **com.rti.dds.domain.DomainParticipant** (p. 670) under the given logical name.*
- final void **unregister_type** (**DomainParticipant** participant, String type_name)

  *Remove the definition of this type from the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- final String **get_type_name** ()

  Get the default name of this type.
- final **TypeCode** **get_data_type** ()

  *Get the **com.rti.dds.typecode.TypeCode** (p. 1873) wrapped by this **com.rti.dds.dynamicdata.DynamicDataType↩ Support** (p. 995).*
- Object **create_data** ()

  *Create a new **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample initialized with the **com.rti.dds.typecode.Type↩ Code** (p. 1873) and properties of this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995).*
- void **destroy_data** (Object data)

  *Finalize and deallocate the **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.*
- void **print_data** (**DynamicData** data)

  Print a string representation of the given sample to the given file.
- void **copy_data** (**DynamicData** dst, **DynamicData** src)

  Deeply copy the given data samples.
- void **to_cdr_buffer** (**ByteSeq** sequence, **DynamicData** data, short representation)

  *Serializes the specified **DynamicData** (p. 847) object into a sequence of octets.*
- void **to_cdr_buffer** (**ByteSeq** sequence, **DynamicData** data)

  *Serializes the specified **DynamicData** (p. 847) object into a CDR sequence of octets.*
- void **from_cdr_buffer** (**DynamicData** data, **ByteSeq** sequence)

  *Deserializes the specified **DynamicData** (p. 847) object from a sequence octets.*
- void **delete** ()

  *Delete a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) object.*
- **DynamicDataTypeSupport** (**TypeCode** type, **DynamicDataTypeProperty_t** props)

  *Construct a new **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) object.*

Static Public Attributes

- static final **DynamicDataTypeProperty_t TYPE_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 990).*

Static Protected Attributes

- static final RuntimeException **DYNAMICDATA_TYPE_NOT_SUPPORTED** = new **RETCODE_ERROR**("not a top-level type")

8.101.1 Detailed Description

A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 847) objects.

A **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) has three roles:

1. It associates a **com.rti.dds.typecode.TypeCode** (p. 1873) with policies for managing objects of that type. See the constructor, com.rti.dds.dynamicdata.DynamicDataTypeSupport.DynamicDataTypeSupport.DynamicDataType↩ Support.
2. It registers its type under logical names with a **com.rti.dds.domain.DomainParticipant** (p. 670). See **com.rti.↩ dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 997).
3. It creates **com.rti.dds.dynamicdata.DynamicData** (p. 847) samples pre-initialized with the type and properties of the type support itself. See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data** (p. 998).

8.101.2 Constructor & Destructor Documentation

8.101.2.1 DynamicDataTypeSupport()

```
DynamicDataTypeSupport (
            TypeCode type,
            DynamicDataTypeProperty_t props )
```

Construct a new **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) object.

This step is usually followed by type registration.

The new object created by this constructor retains a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873) that is passed in. It is *not* safe to delete the **com.rti.dds.typecode.TypeCode** (p. 1873) until the **com.rti.dds.dynamicdata.↩ DynamicDataTypeSupport** (p. 995) itself is deleted. You have two options:

- Keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873) object yourself, and delete it with **com.↩ rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934) after you've deleted the **com.rti.dds.dynamicdata.↩ DynamicDataTypeSupport** (p. 995).
- Do not keep a reference to the **com.rti.dds.typecode.TypeCode** (p. 1873). The garbage collector will delete it when it's eligible for collection.

If the object cannot be created, this method throws **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Parameters

type	The **com.rti.dds.typecode.TypeCode** (p. 1873) that describes the members of this type.
props	Policies that describe how to manage the memory and other properties of the data samples created by this factory. In most cases, the default values will be appropriate; see **com.rti.dds.dynamicdata.DynamicDataTypeSupport.TYPE_PROPERTY_DEFAULT** (p. 69).

See also

com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type (p. 997)

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DynamicDataProperty_t.buffer_initial_size**, **DynamicDataProperty_t.buffer_max_size**, **DynamicDataTypeProperty_t.data**, **DynamicDataTypeSerializationProperty_t.max_size_serialized**, **DynamicDataTypeSerializationProperty_t.min_size_serialized**, **DynamicDataTypeProperty_t.serialization**, **DynamicDataTypeSerializationProperty_t.trim_to_size**, and **DynamicDataTypeSerializationProperty_t.use_42e_compatible_alignment**.

8.101.3 Member Function Documentation

8.101.3.1 register_type()

```
final void register_type (
             DomainParticipant participant,
            String type_name )
```

Associate the **com.rti.dds.typecode.TypeCode** (p. 1873) with the given **com.rti.dds.domain.DomainParticipant** (p. 670) under the given logical name.

Once a type has been registered, it can be referenced by name when creating a topic. Statically and dynamically defined types behave the same way in this respect.

If the type cannot be registered, this function will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

See also

com.rti.ndds.example.FooTypeSupport.register_type (p. 1119)

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

com.rti.dds.dynamicdata.DynamicDataTypeSupport.unregister_type (p. 997)

8.101.3.2 unregister_type()

```
final void unregister_type (
              DomainParticipant participant,
             String type_name )
```

Remove the definition of this type from the **com.rti.dds.domain.DomainParticipant** (p. 670).

This operation is optional; all types are automatically unregistered when a **com.rti.dds.domain.DomainParticipant** (p. 670) is deleted. Most application will not need to manually unregister types.

A type cannot be unregistered while it is still in use; that is, while any **com.rti.dds.topic.Topic** (p. 1807) is still referring to it.

If the type cannot be unregistered, this function will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595) or **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

See also

com.rti.ndds.example.FooTypeSupport.unregister_type

com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type (p. 997)

8.101.3.3 get_type_name()

```
final String get_type_name ( )
```

Get the default name of this type.

The **com.rti.dds.typecode.TypeCode** (p. 1873) that is wrapped by this **com.rti.dds.dynamicdata.DynamicData↩ TypeSupport** (p. 995) includes a name; this operation returns that name.

This operation is useful when registering a type, because in most cases it is not necessary for the physical and logical names of the type to be different.

```
myTypeSupport.register_type(myParticipant, myTypeSupport.get_type_name());
```

See also

com.rti.ndds.example.FooTypeSupport.get_type_name (p. 1119)

8.101.3.4 get_data_type()

```
final  TypeCode get_data_type ( )
```

Get the **com.rti.dds.typecode.TypeCode** (p. 1873) wrapped by this **com.rti.dds.dynamicdata.DynamicDataType↩ Support** (p. 995).

8.101.3.5 create_data()

```
Object create_data ( )
```

Create a new **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample initialized with the **com.rti.dds.typecode.↩ TypeCode** (p. 1873) and properties of this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995).

If this method fails, it will throw a Runtime Exception.

See also

com.rti.ndds.example.FooTypeSupport.create_data

com.rti.dds.dynamicdata.DynamicData.DynamicData.DynamicData

com.rti.dds.dynamicdata.DynamicDataTypeProperty_t.data (p. 991)

8.101.3.6 destroy_data()

```
void destroy_data (
             Object data )
```

Finalize and deallocate the **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample.

See also

com.rti.ndds.example.FooTypeSupport.delete_data

com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data (p. 998)

References **DynamicData.delete()**.

8.101.3.7 print_data()

```
void print_data (
              DynamicData data )
```

Print a string representation of the given sample to the given file.

This method is equivalent to **com.rti.dds.dynamicdata.DynamicData.print** (p. 864).

See also

com.rti.dds.dynamicdata.DynamicData.print (p. 864)

8.101.3.8 copy_data()

```
void copy_data (
              DynamicData dst,
              DynamicData src )
```

Deeply copy the given data samples.

8.101.3.9 to_cdr_buffer() [1/2]

```
void to_cdr_buffer (
              ByteSeq sequence,
              DynamicData data,
             short representation )
```

Serializes the specified **DynamicData** (p. 847) object into a sequence of octets.

This method serializes the specified **DynamicData** (p. 847) object into a sequence of octets using the input data representation.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) method, except that it receives the **DynamicData** (p. 847) object to be serialized and a **com.rti.dds.infrastructure.↩ ByteSeq** (p. 395) as a serialization output.

This method may resize `sequence` as needed to fit the serialized **DynamicData** (p. 847) object.

See also

com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer (p. 946)

Parameters

sequence	*<<**out**>> (p. 156).* Serialization byte sequence.
data	*<<**in**>> (p. 156).* The **DynamicData** (p. 847) object to be serialized. Cannot be null.
representation	*<<**in**>> (p. 156).* Representation used to serialize the data.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DynamicData.to_cdr_buffer()**.

Referenced by **DynamicDataTypeSupport.to_cdr_buffer()**.

8.101.3.10 to_cdr_buffer() [2/2]

```
void to_cdr_buffer (
                ByteSeq sequence,
                DynamicData data )
```

Serializes the specified **DynamicData** (p. 847) object into a CDR sequence of octets.

This method serializes the specified **DynamicData** (p. 847) object into a sequence of octets, using CDR as the data representation.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer** (p. 946) method, except that it receives the **DynamicData** (p. 847) object to be serialized and a **com.rti.dds.infrastructure.ByteSeq** (p. 395) as output.

This method may resize `sequence` as needed to fit the serialized dynamicData object.

See also

com.rti.dds.dynamicdata.DynamicData.to_cdr_buffer (p. 946)

Parameters

sequence	<<***out***>> *(p. 156)*. Serialization byte sequence.
data	<<***in***>> *(p. 156)*. The **DynamicData** (p. 847) object to be serialized. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **DynamicDataTypeSupport.to_cdr_buffer()**.

8.101.3.11 from_cdr_buffer()

```
void from_cdr_buffer (
                DynamicData data,
                ByteSeq sequence )
```

Deserializes the specified **DynamicData** (p. 847) object from a sequence octets.

This method deserializes the specified **DynamicData** (p. 847) object from a CDR sequence of octets.

The behavior of this method is the same as that of the **com.rti.dds.dynamicdata.DynamicData.from_cdr_buffer** (p. 947) method, except that it receives the **DynamicData** (p. 847) object that contains the result of the deserialization, and a **com.rti.dds.infrastructure.ByteSeq** (p. 395) as deserialization input.

See also

com.rti.dds.dynamicdata.DynamicData.from_cdr_buffer (p. 947)

Parameters

data	<<***in***>> *(*p. *156)*. The **DynamicData** (p. 847) object to hold the result of the deserialization.
sequence	<<***in***>> *(*p. *156)*. Deserialization octet sequence. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **DynamicData.from_cdr_buffer()**.

8.101.3.12 delete()

```
void delete ( )
```

Delete a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) object.

A **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 995) cannot be deleted while it is still in use. For each **com.rti.dds.domain.DomainParticipant** (p. 670) with which the **com.rti.dds.dynamicdata.DynamicData↩ TypeSupport** (p. 995) is registered, either the type must be unregistered or the participant must be deleted.

Calling this method is optional. If you do not call it, the garbage collector will perform the deletion when it is able.

See also

com.rti.dds.dynamicdata.DynamicDataTypeSupport.unregister_type (p. 997)

com.rti.dds.dynamicdata.DynamicDataTypeSupport.DynamicDataTypeSupport.DynamicDataTypeSupport

8.101.4 Member Data Documentation

8.101.4.1 DYNAMICDATA_TYPE_NOT_SUPPORTED

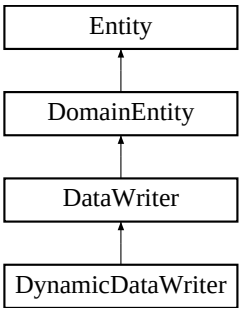
```
final RuntimeException DYNAMICDATA_TYPE_NOT_SUPPORTED = new  RETCODE_ERROR("not a top-level type")
[static], [protected]
```

Cached exception to be thrown in the event that we can't create native type support.

8.102 DynamicDataWriter Class Reference

Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).

Inheritance diagram for DynamicDataWriter:



Public Member Functions

- **InstanceHandle_t register_instance** (**DynamicData** instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- **InstanceHandle_t register_instance_untyped** (Object instance_data)

  Register a new instance with this writer.
- **InstanceHandle_t register_instance_w_timestamp** (**DynamicData** instance_data, **Time_t** source_↩ timestamp)

  Performs the same functions as register_instance except that the application provides the value for the `source_↩ timestamp`*.*
- void **unregister_instance** (**DynamicData** instance_data, **InstanceHandle_t** handle)

  Reverses the action of ***com.rti.ndds.example.FooDataWriter.register_instance*** *(*p. *1098).*
- void **unregister_instance_untyped** (Object instance_data, **InstanceHandle_t** handle)

  Unregister a new instance from this writer.
- void **unregister_instance_w_timestamp** (**DynamicData** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as ***com.rti.ndds.example.FooDataWriter.unregister_instance*** *(*p. *1100) except that it also provides the value for the* `source_timestamp`*.*
- void **write** (**DynamicData** instance_data, **InstanceHandle_t** handle)

  Modifies the value of a data instance.
- void **write_untyped** (Object instance_data, **InstanceHandle_t** handle)

  Publish a data sample.
- void **write_w_timestamp** (**DynamicData** instance_data, **InstanceHandle_t** handle, **Time_t** source_↩ timestamp)

  Performs the same function as ***com.rti.ndds.example.FooDataWriter.write*** *(*p. *1105) except that it also provides the value for the* `source_timestamp`*.*
- void **dispose** (**DynamicData** instance_data, **InstanceHandle_t** instance_handle)

  Requests the middleware to delete the instance.
- void **dispose_w_timestamp** (**DynamicData** instance_data, **InstanceHandle_t** instance_handle, **Time_↩ t** source_timestamp)

Performs the same functions as dispose except that the application provides the value for the `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(*p. *450) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(*p. *1634).*

- void **dispose_w_timestamp_untyped** (Object instance_data, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

  Dispose a data sample using the given time instead of the current time.
- void **get_key_value** (**DynamicData** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` *that corresponds to an instance* `handle`*.*
- **InstanceHandle_t lookup_instance** (**DynamicData** key_holder)

  Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*

8.102.1 Detailed Description

Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 847).

Instantiates **`com.rti.dds.publication.DataWriter`** (p. 553) < **com.rti.dds.dynamicdata.Dynamic↩ Data** (p. 847) > .

See also

com.rti.dds.publication.DataWriter (p. 553)

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.dynamicdata.DynamicData (p. 847)

8.102.2 Member Function Documentation

8.102.2.1 register_instance()

```
InstanceHandle_t register_instance (
            DynamicData instance_data )
```

Informs RTI Connext that the application will be modifying a particular instance.

This operation is only useful for keyed data types. Using it for non-keyed types causes no effect and returns **com.↩ rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). The operation takes as a parameter an instance (of which only the key value is examined) and returns a `handle` that can be used in successive **write()** (p. 1010) or **dispose()** (p. 1016) operations.

The operation gives RTI Connext an opportunity to pre-configure itself to improve performance.

The use of this operation by an application is optional even for keyed types. If an instance has not been pre-registered, the application can use the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) as the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) parameter to the write or dispose operation and RTI Connext will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **com.rti.ndds.example.FooDataWriter.write** (p. 1105), **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109), **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) and **com.rti.ndds.example.FooDataWriter.dispose_w_timestamp** (p. 1113) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) limit has been exceeded.

The operation is **idempotent**. If it is called for an already registered instance, it just returns the already allocated handle. This may be used to lookup and retrieve the handle allocated to a given instance.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 553) has been enabled. Otherwise, **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will be returned.

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.

Returns

For keyed data type, a handle that can be used in the calls that take a **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152), such as write, dispose, unregister_instance, or return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) on failure. If the instance_data is of a data type that has no keys, this function always returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100), **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114), **Relationship between registration, liveliness and ownership** (p. 1344)

References **InstanceHandle_t.HANDLE_NIL**.

Referenced by **DynamicDataWriter.register_instance_untyped()**.

8.102.2.2 register_instance_untyped()

```
InstanceHandle_t register_instance_untyped (
            Object instance_data )
```

Register a new instance with this writer.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.unregister_instance_untyped (p. 574)

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

Implements **DataWriter** (p. 573).

References **DynamicDataWriter.register_instance()**.

8.102.2.3 register_instance_w_timestamp()

```
InstanceHandle_t register_instance_w_timestamp (
            DynamicData instance_data,
            Time_t source_timestamp )
```

Performs the same functions as register_instance except that the application provides the value for the source_↩ timestamp.

The provided source_timestamp potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 218) QoS policy for details.

This operation may fail and return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if **com.rti.↩ dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) limit has been exceeded.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 553) has been enabled. Otherwise, **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will be returned.

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Returns

For keyed data type, return a handle that can be used in the calls that take a **com.rti.dds.infrastructure.↩ InstanceHandle_t** (p. 1152), such as write, dispose, unregister_instance, or return **com.rti.dds.infrastructure.↩ InstanceHandle_t.HANDLE_NIL** (p. 1156) on failure. If the instance_data is of a data type that has no keys, this function always return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100), **com.rti.ndds.example.FooData↩ Writer.get_key_value** (p. 1114)

References **InstanceHandle_t.HANDLE_NIL**, **Time_t.is_invalid()**, **Time_t.nanosec**, and **Time_t.sec**.

8.102.2.4 unregister_instance()

```
void unregister_instance (
              DynamicData instance_data,
              InstanceHandle_t handle )
```

Reverses the action of **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098).

This operation is useful only for keyed data types. Using it for non-keyed types causes no effect and reports no error. The operation takes as a parameter an instance (of which only the key value is examined) and a handle.

This operation should only be called on an instance that is currently registered. This includes instances that have been auto-registered by calling operations such as write or dispose as described in **com.rti.ndds.example.FooDataWriter.↩ register_instance** (p. 1098). Otherwise, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_↩ PARAMETER** (p. 1594).

This only need be called just once per instance, regardless of how many times register_instance was called for that instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Connext that the **com.rti.dds.publication.DataWriter** (p. 553) is no longer going to provide any information about the instance. This operation also indicates that RTI Connext can locally remove all information regarding that instance. The application should not attempt to use the `handle` previously allocated to that instance after calling this function.

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `handle`. This indicates that the identity of the instance should be automatically deduced from the `instance_↩ data` (by means of the `key`).

If `handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the handle is any value other than **com.rti.dds.infrastructure.Instance↩ Handle_t.HANDLE_NIL** (p. 1156), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **unregister_↩ instance()** (p. 1007) operation is for the instance as indicated by the `handle`.

If, after a **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100), the application wants to modify (**com.rti.ndds.example.FooDataWriter.write** (p. 1105) or **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111)) an instance, it has to register it again, or else use the special `handle` value **com.rti.dds.infrastructure.Instance↩ Handle_t.HANDLE_NIL** (p. 1156).

This operation does not indicate that the instance is deleted (that is the purpose of **com.rti.ndds.example.Foo↩ DataWriter.dispose** (p. 1111)). The operation **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) just indicates that the **com.rti.dds.publication.DataWriter** (p. 553) no longer has anything to say about the instance. **com.rti.dds.subscription.DataReader** (p. 450) entities that are reading the instance may receive a sample with **com.↩ rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) for the instance, unless there are other **com.rti.dds.publication.DataWriter** (p. 553) objects writing that same instance.

com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances (p. 2007) controls whether instances are automatically disposed when they are unregistered.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 244)). If the **com.rti.dds.↩ publication.DataWriter** (p. 553) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1007) will relinquish that ownership.

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) is set to **com.rti.dds.infrastructure.Reliability↩ QosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) and the unregistration would overflow the resource limits of this writer or of a reader, this operation may block for up to **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_↩ blocking_time** (p. 1528); if this writer is still unable to unregister after that period, this method will fail with **com.rti.↩ dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) . If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
handle	<<***in***>> *(p. 156)* represents the `instance` to be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` is not used and `instance` is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will report the error **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

Relationship between registration, liveliness and ownership (p. 1344)

Referenced by **DynamicDataWriter.unregister_instance_untyped()**.

8.102.2.5 unregister_instance_untyped()

```
void unregister_instance_untyped (
             Object instance_data,
              InstanceHandle_t handle )
```

Unregister a new instance from this writer.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.register_instance_untyped (p. 573)

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

Implements **DataWriter** (p. 574).

References **DynamicDataWriter.unregister_instance()**.

8.102.2.6 unregister_instance_w_timestamp()

```
void unregister_instance_w_timestamp (
              DynamicData instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) except that it also provides the value for the `source_timestamp`.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 218) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.

This operation may block and may time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for the unregister_instance operation.

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
handle	<<***in***>> *(p. 156)* represents the `instance` to be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` is not used and `instance` is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will report the error **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **Time_t.nanosec**, and **Time_t.sec**.

8.102.2.7 write()

```
void write (
                DynamicData instance_data,
                InstanceHandle_t handle )
```

Modifies the value of a data instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1634) and **DESTINATION_ORDER** (p. 218) QoS policy for details).

As a side effect, this operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 553) itself, the **com.rti.dds.publication.Publisher** (p. 1466) and the **com.rti.dds.domain.DomainParticipant** (p. 670).

Note that the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the handle is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **write()** (p. 1010) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 258) `kind` is set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 259) to be exceeded.

This operation will not block when using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532). If you are using BEST_EFFORT Reliability in combination with com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS, then instead of being blocked, samples that are queued to be sent by the asynchronous publishing thread will be overwritten when the number of DDS samples that are currently queued has reached the `depth` QoS value in the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144).

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) elapses before the **com.rti.dds.publication.DataWriter** (p. 553) can store the modification without exceeding the limits, the operation will fail and return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) for KEEP_ALL configurations.

Here is how the write operation behaves when com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) are used:

- The send window size is determined by the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1616) fields in the **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires.
- Then, the **com.rti.dds.publication.DataWriter** (p. 553) will try to add the new sample to the writer history.
- If the instance associated with the sample is present in the writer history and there are `depth` (in the **HISTORY** (p. 237)) samples in the instance, the DataWriter will replace the oldest sample of that instance independently of that sample's acknowledged status, and the write operation will return com.rti.dds.infrastructure.RETCODE_OK. Otherwise, no sample will be replaced and the write operation will continue.

- If the instance associated with the sample is not present in the writer history and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629) (see **com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind** (p. 623)).
  - If no instance can be replaced, the write operation returns **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).
- If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) is exceeded, the DataWriter will try to drop a sample from a different instance as follows:
  - The DataWriter will try first to remove a fully ACKed (2) sample from a different instance 'I' as long as that sample is not the last remaining sample for the instance 'I'. To find this sample, the DataWriter starts iterating from the oldest sample in the writer history to the newest sample.
  - If no such sample is found, the DataWriter will replace the oldest sample in the writer history.
- The sample is added to the writer history, and the write operation returns com.rti.dds.infrastructure.RETCODE_OK.

Here is how the write operation behaves when com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) are used:

- The send window size is determined by the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1616) fields in the **DATA_WRITER_PROTOCOL** (p. 215). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires.
  - If the max_blocking_time expires, the write operation returns **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
- When a sample is protocol-ACKed (1) before max_blocking_time expires, the DataWriter will try to add the sample to the writer history as follows:
  - If the instance associated with the sample is not present in the writer history and max_instances is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629) (see **com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind** (p. 623)).
    - If no instance can be replaced, the write operation returns **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).
  - If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) is exceeded, the DataWriter will go through the samples in the order in which they were added, and it will replace the first sample that is fully ACKed (2).
    - If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires. If the max_blocking_time expires, the write operation will return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
  - If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) is exceeded, the DataWriter will go through the samples of the instance in the order in which they were added, and it will replace the first sample that is fully ACKed.

    * If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or the max_blocking_time expires. If the max_blocking_time expires, the write operation will return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
  - The sample is added to the writer history, and the write operation returns com.rti.dds.infrastructure.↩ RETCODE_OK.

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_↩ OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help freeing up some resources.

This operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if the timestamp is less than the timestamp used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp).

See **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind** (p. 1530) for more information on the following notes:

(1) A sample in the writer history is considered "protocol ACKed" when the sample has been individually ACKed at the RTPS protocol level by each one of the DataReaders that matched the DataWriter at the moment the sample was added to the writer queue.

- Late joiners do not change the protocol ACK state of a sample. If a sample is marked as protocol ACKed because it has been acknowledged by all the matching DataReaders and a DataReader joins later on, the historical sample is still considered protocol ACKed even if it has not been received by the late joiner.
- If a sample 'S1' is protocol ACKed and a TopicQuery is received, triggering the publication of 'S1', the sample is still considered protocol ACKed. If a sample 'S1' is not ACKed and a TopicQuery is received triggering the publication of 'S1', the DataWriter will require that both the matching DataReaders on the live RTPS channel and the DataReader on the TopicQuery channel individually protocol ACK the sample in order to consider the sample protocol ACKed.

(2) A sample in the writer history is considered "fully ACKed" when all of the following conditions are met:

- The sample is protocol-ACKed.
- The sample has been "application-level ACKed" by all the DataReaders matching the DataWriter that have their **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) set to **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 1531) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgment↩ ModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530). Once the sample is application-level ACKed, it cannot change its status to not ACked after new DataReaders are matched. (Application-level ACK occurs when the application acknowledges receipt of a sample.)
- If required subscriptions are enabled (see **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343)), the sample must also be ACKed by all the required subscriptions configured on the DataWriter.

(3) It is possible within a single call to the write operation for a DataWriter to block both when the send window is full and then again when **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) or **com.rti.↩ dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) is exceeded. This can happen because blocking on the send window only considers protocol-ACKed samples, while blocking based on resource limits considers fully-ACKed samples. In any case, the total max blocking time of a single call to the write operation will not exceed **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528).

Parameters

instance_data	<<***in***>> *(*p. *156)* The data to write.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_data` is NULL.

Parameters

handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

MT Safety:

It is UNSAFE to modify `instance_data` before the operation has finished. The operation is otherwise SAFE.

See also

com.rti.dds.subscription.DataReader (p. 450)

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

DESTINATION_ORDER (p. 218)

Referenced by **DynamicDataWriter.write_untyped()**.

8.102.2.8 write_untyped()

```
void write_untyped (
             Object instance_data,
              InstanceHandle_t handle )
```

Publish a data sample.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.write** (p. 1105) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.write_w_timestamp_untyped (p. 576)

com.rti.ndds.example.FooDataWriter.write (p. 1105)

Implements **DataWriter** (p. 575).

References **DynamicDataWriter.write()**.

8.102.2.9 write_w_timestamp()

```
void write_w_timestamp (
              DynamicData instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.write** (p. 1105) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1634) and **DESTINATION_ORDER** (p. 218) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) operation.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_↩ RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help free up some resources.

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) under the same circumstances described for the write operation.

Parameters

instance_data	<<***in***>> *(p. 156)* The data to write. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_data` is NULL.
handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.

Parameters

source_timestamp	<<***in***>> *(*p. *156)* When using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.↩ DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the timestamp of the previous operation but the difference is less than the **com.rti.dds.infrastructure.DestinationOrderQosPolicy.source_timestamp_tolerance** (p. 637), the timestamp of the previous operation will be used as the source timestamp of this sample. Otherwise, if the difference is greater than **com.rti.dds.infrastructure.DestinationOrderQosPolicy.source_timestamp_tolerance** (p. 637), the function will return **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

com.rti.dds.subscription.DataReader (p. 450)

DESTINATION_ORDER (p. 218)

References **Time_t.nanosec**, and **Time_t.sec**.

8.102.2.10 dispose()

```
void dispose (
                DynamicData instance_data,
                InstanceHandle_t instance_handle )
```

Requests the middleware to delete the instance.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

When an instance is disposed, the **com.rti.dds.publication.DataWriter** (p. 553) communicates this state change to **com.rti.dds.subscription.DataReader** (p. 450) objects by propagating a dispose sample. When the instance changes to a disposed state, you can see the state change on the DataReader by looking at **com.rti.dds.subscription.↩ SampleInfo.instance_state** (p. 1640). Disposed instances have the value **com.rti.dds.subscription.InstanceState↩ Kind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).

The resources allocated to dispose instances on the DataWriter are not removed by default. The removal of the resources allocated to a dispose instance on the DataWriter queue can be controlled by using the QoS **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autopurge_disposed_instances_delay** (p. 2008).

Likewise, on the DataReader, the removal of the resources associated with an instance in the dispose state can be controlled by using the QoS **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_disposed_instances_delay** (p. 1522).

This operation does not modify the value of the instance. The `instance_data` parameter is passed just for the purposes of identifying the instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the source_timestamp attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `instance_handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `instance_handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the `instance_handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), and the `instance_handle` corresponds to an instance that has been registered but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). In this case, the instance that will be disposed is the instance corresponding to the `instance_handle`, not to the `instance_data`.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumstances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help free up some resources.

Parameters

instance_data	<<***in***>> *(*p. *156)* The data to dispose. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Parameters

instance_handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `instance_handle` is not used and it is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `instance_handle` is not used. If `instance_handle` is used, it must represent an instance of type **com.rti.ndds.example.Foo** (p. 1066) that has been written or registered with this writer. Otherwise, this method fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

Relationship between registration, liveliness and ownership (p. 1344)

8.102.2.11 dispose_w_timestamp()

```
void dispose_w_timestamp (
                DynamicData instance_data,
                InstanceHandle_t instance_handle,
                Time_t source_timestamp )
```

Performs the same functions as dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) operation.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help freeing up some resources.

Parameters

instance_data	<<***in***>> *(*p. *156)* The data to dispose. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
instance_handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.
source_timestamp	<<***in***>> *(*p. *156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **Time_t.nanosec**, and **Time_t.sec**.

Referenced by **DynamicDataWriter.dispose_w_timestamp_untyped()**.

8.102.2.12 dispose_w_timestamp_untyped()

```
void dispose_w_timestamp_untyped (
             Object instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Dispose a data sample using the given time instead of the current time.

This method allows type-independent code to work with a variety of concrete **com.rti.ndds.example.FooDataWriter** (p. 1097) classes in a consistent way.

Statically type-safe code should use the appropriate **com.rti.ndds.example.FooDataWriter.dispose_w_timestamp** (p. 1113) method instead of this one. See that method for detailed documentation.

See also

com.rti.dds.publication.DataWriter.dispose_untyped (p. 576)

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

Implements **DataWriter** (p. 577).

References **DynamicDataWriter.dispose_w_timestamp()**.

8.102.2.13 get_key_value()

```
void get_key_value (
               DynamicData key_holder,
               InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

key_holder	<<***inout***>> *(p. 156)* a user data type specific key holder, whose `key` fields are filled by this operation. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `key_holder` is NULL.

Parameters

handle	<<***in***>> *(*p. *156)* the `instance` whose key is to be retrieved. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

8.102.2.14 lookup_instance()

```
InstanceHandle_t lookup_instance (
            DynamicData key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason RTI Connext is unable to provide an instance handle, RTI Connext will return the special value HANDLE_NIL.

Parameters

key_holder	<<***in***>> *(*p. *156)* a user data type specific key holder.

Returns

the instance handle associated with this instance. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect and returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156)

References **InstanceHandle_t.HANDLE_NIL**.

8.103 EndpointGroup_t Class Reference

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

Inherits Struct.

Public Member Functions

- **EndpointGroup_t** ()
  Constructor.
- **EndpointGroup_t** (**EndpointGroup_t** src)
  Copy constructor.
- **EndpointGroup_t** (String **role_name**, int **quorum_count**)
  *Construct an **EndpointGroup_t** (p. 1022) with the given parameters.*

Public Attributes

- String **role_name** = null
  Defines the role name of the endpoint group.
- int **quorum_count** = 0
  Defines the minimum number of members that satisfies the endpoint group.

8.103.1 Detailed Description

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

8.103.2 Constructor & Destructor Documentation

8.103.2.1 EndpointGroup_t() [1/3]

```
EndpointGroup_t ( )
```

Constructor.

8.103.2.2 EndpointGroup_t() [2/3]

```
EndpointGroup_t (
            EndpointGroup_t src )
```

Copy constructor.

References **EndpointGroup_t.quorum_count**, and **EndpointGroup_t.role_name**.

8.103.2.3 EndpointGroup_t() [3/3]

```
EndpointGroup_t (
            String role_name,
            int quorum_count )
```

Construct an **EndpointGroup_t** (p. 1022) with the given parameters.

References **EndpointGroup_t.quorum_count**, and **EndpointGroup_t.role_name**.

8.103.3 Member Data Documentation

8.103.3.1 role_name

```
String role_name = null
```

Defines the role name of the endpoint group.

If used in the **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343) on a **com.rti.dds.publication.DataWriter** (p. 553), it specifies the name that identifies a Durable Subcription.

The role name can be at most 255 characters in length.

Referenced by **EndpointGroup_t.EndpointGroup_t()**.

8.103.3.2 quorum_count

```
int quorum_count = 0
```

Defines the minimum number of members that satisfies the endpoint group.

If used in the **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343) on a **com.rti.dds.publication.DataWriter** (p. 553), it specifies the number of DataReaders that must acknowledge a sample before the sample is considered to be acknowledged by the Durable Subscription.

Referenced by **EndpointGroup_t.EndpointGroup_t()**.

8.104 EndpointGroupSeq Class Reference

A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022).

Inherits ArraySequence.

8.104.1 Detailed Description

A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 1022).

In the context of Collaborative DataWriters, it can be used by a **com.rti.dds.subscription.DataReader** (p. 450) to define a group of remote DataWriters that the **com.rti.dds.subscription.DataReader** (p. 450) will wait to discover before skipping missing samples.

In the context of Durable Subscriptions, it can be used to create a set of Durable Subscriptions identified by a name and a quorum count.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.EndpointGroup_t (p. 1022)

8.105 EndpointTrustAlgorithmInfo Class Reference

Trust Plugins algorithm information associated with the discovered endpoint.

Inherits Struct.

Public Member Functions

- **EndpointTrustAlgorithmInfo** ()

  Create an instance with the default Trust Algorithm Info associated with the discovered endpoint.
- **EndpointTrustAlgorithmInfo** (**EndpointTrustInterceptorAlgorithmInfo interceptor**)

  Create an instance with the Trust Algorithm Info passed as input.

Public Attributes

- **EndpointTrustInterceptorAlgorithmInfo interceptor**

  Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

8.105.1 Detailed Description

Trust Plugins algorithm information associated with the discovered endpoint.

8.105.2 Constructor & Destructor Documentation

8.105.2.1 EndpointTrustAlgorithmInfo() [1/2]

```
EndpointTrustAlgorithmInfo ( )
```

Create an instance with the default Trust Algorithm Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.105.2.2 EndpointTrustAlgorithmInfo() [2/2]

```
EndpointTrustAlgorithmInfo (
            EndpointTrustInterceptorAlgorithmInfo interceptor )
```

Create an instance with the Trust Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

References **EndpointTrustAlgorithmInfo.interceptor**.

8.105.3 Member Data Documentation

8.105.3.1 interceptor

```
EndpointTrustInterceptorAlgorithmInfo interceptor
```

Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

Referenced by **EndpointTrustAlgorithmInfo.EndpointTrustAlgorithmInfo()**.

8.106 EndpointTrustInterceptorAlgorithmInfo Class Reference

Trust Plugins interception algorithm information associated with the discovered endpoint.

Inherits Struct.

Public Member Functions

- **EndpointTrustInterceptorAlgorithmInfo** ()

  Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered endpoint.
- **EndpointTrustInterceptorAlgorithmInfo** (int **required_mask**, int **supported_mask**)

  Create an instance with the Trust Interceptor Algorithm Info passed as input.

Public Attributes

- int **required_mask**

  Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.
- int **supported_mask**

  Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

8.106.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered endpoint.

8.106.2 Constructor & Destructor Documentation

8.106.2.1 EndpointTrustInterceptorAlgorithmInfo() [1/2]

```
EndpointTrustInterceptorAlgorithmInfo ( )
```

Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.106.2.2 EndpointTrustInterceptorAlgorithmInfo() [2/2]

```
EndpointTrustInterceptorAlgorithmInfo (
            int required_mask,
            int supported_mask )
```

Create an instance with the Trust Interceptor Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

References **EndpointTrustInterceptorAlgorithmInfo.required_mask**, and **EndpointTrustInterceptorAlgorithm↩ Info.supported_mask**.

8.106.3 Member Data Documentation

8.106.3.1 required_mask

```
int required_mask
```

Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.

Referenced by **EndpointTrustInterceptorAlgorithmInfo.EndpointTrustInterceptorAlgorithmInfo()**.

8.106.3.2 supported_mask

```
int supported_mask
```

Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

Referenced by **EndpointTrustInterceptorAlgorithmInfo.EndpointTrustInterceptorAlgorithmInfo()**.

8.107 EndpointTrustProtectionInfo Class Reference

Trust Plugins Protection information associated with the discovered endpoint.

Inherits Struct.

Public Member Functions

- **EndpointTrustProtectionInfo** ()

  Create an instance with the default Trust Plugins Protection Info associated with the discovered endpoint.
- **EndpointTrustProtectionInfo** (int **bitmask**, int **plugin_bitmask**)

  Create an instance with the Trust Plugins Protection Info passed as input.

Public Attributes

- int **bitmask**

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- int **plugin_bitmask**

  Internal plugin information that is opaque to DDS.

8.107.1 Detailed Description

Trust Plugins Protection information associated with the discovered endpoint.

8.107.2 Constructor & Destructor Documentation

8.107.2.1 EndpointTrustProtectionInfo() [1/2]

```
EndpointTrustProtectionInfo ( )
```

Create an instance with the default Trust Plugins Protection Info associated with the discovered endpoint.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

8.107.2.2 EndpointTrustProtectionInfo() [2/2]

```
EndpointTrustProtectionInfo (
            int bitmask,
            int plugin_bitmask )
```

Create an instance with the Trust Plugins Protection Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the endpoint is using.

References **EndpointTrustProtectionInfo.bitmask**, and **EndpointTrustProtectionInfo.plugin_bitmask**.

8.107.3 Member Data Documentation

8.107.3.1 bitmask

```
int bitmask
```

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

Referenced by **EndpointTrustProtectionInfo.EndpointTrustProtectionInfo()**.

8.107.3.2 plugin_bitmask

```
int plugin_bitmask
```

Internal plugin information that is opaque to DDS.

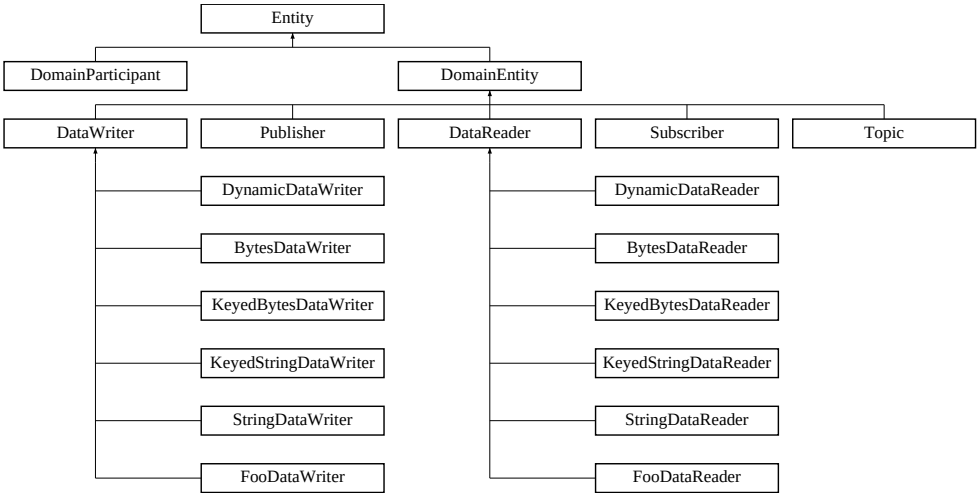
The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

Referenced by **EndpointTrustProtectionInfo.EndpointTrustProtectionInfo()**.

8.108 Entity Interface Reference

<<***interface***>> *(*p. *156)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

Inheritance diagram for Entity:



Public Member Functions

- void **enable** ()

  Enables the ***com.rti.dds.infrastructure.Entity*** *(*p. *1029).*
- **StatusCondition get_statuscondition** ()

  Allows access to the ***com.rti.dds.infrastructure.StatusCondition*** *(*p. *1699) associated with the* ***com.rti.dds.infrastructure.Entity*** *(*p. *1029).*
- int **get_status_changes** ()

  Retrieves the list of communication statuses in the ***com.rti.dds.infrastructure.Entity*** *(*p. *1029) that are triggered.*
- **InstanceHandle_t get_instance_handle** ()

  Allows access to the ***com.rti.dds.infrastructure.InstanceHandle_t*** *(*p. *1152) associated with the* ***com.rti.dds.infrastructure.Entity*** *(*p. *1029).*

8.108.1 Detailed Description

<<***interface***>> *(p. 156)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

All operations except for set_qos(), get_qos(), set_listener(), get_listener() and **enable()** (p. 1032), may return the value **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

QoS:

QoS Policies (p. 250)

Status:

Status Kinds (p. 262)

Listener:

com.rti.dds.infrastructure.Listener (p. 1236)

8.108.2 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Connext.

8.108.2.1 set_qos (abstract)

This operation sets the QoS policies of the **com.rti.dds.infrastructure.Entity** (p. 1029). Each of the derived entity classes provides this operation: **com.rti.dds.infrastructure.Entity** (p. 1029) classes (**com.rti.dds.domain.Domain↩ Participant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.↩ publication.DataWriter** (p. 553), **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.dds.subscription.↩ DataReader** (p. 450)) so that the policies that are meaningful to each **com.rti.dds.infrastructure.Entity** (p. 1029) can be set. For example, see **com.rti.dds.domain.DomainParticipant.set_qos** (p. 714).

Precondition

Certain policies are immutable (see **QoS Policies** (p. 250)): they can only be set at **com.rti.dds.infrastructure.↩ Entity** (p. 1029) creation time or before the entity is enabled. If `set_qos()` is invoked after the **com.rti.dds.↩ infrastructure.Entity** (p. 1029) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596).

Certain values of QoS policies can be incompatible with the settings of the other policies. The `set_qos()` operation will also fail if it specifies a set of values that, once combined with the existing values, would result in an inconsistent set of policies. In this case, the operation will fail and return **com.rti.dds.infrastructure.RETCODE↩ _INCONSISTENT_POLICY** (p. 1596).

If the application supplies a non-default value for a QoS policy that is not supported by the implementation of the service, the set_qos operation will fail and return **com.rti.dds.infrastructure.RETCODE_UNSUPPORTED** (p. 1599).

Postcondition

The existing set of policies is only changed if the `set_qos()` operation succeeds. This is indicated by a return code of com.rti.dds.infrastructure.RETCODE_OK. In all other cases, none of the policies are modified.

Each derived **com.rti.dds.infrastructure.Entity** (p. 1029) class (**com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.subscription.DataReader** (p. 450)) has a corresponding special value of the QoS (**com.rti.dds.domain.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 42), **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46), **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47), **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45), **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71), **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80)). This special value may be used as a parameter to the `set_qos` operation to indicate that the QoS of the **com.rti.dds.infrastructure.Entity** (p. 1029) should be changed to match the current default QoS set in the **com.rti.dds.infrastructure.Entity** (p. 1029)'s factory. The operation set_qos cannot modify the immutable QoS, so a successful return of the operation indicates that the mutable QoS for the **Entity** (p. 1029) has been modified to match the current default for the **com.rti.dds.infrastructure.Entity** (p. 1029)'s factory.

The set of policies specified in the `qos` parameter are applied on top of the existing QoS, replacing the values of any policies previously set.

Possible exceptions thrown in addition to **Standard Return Codes** (p. 261) : **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

8.108.2.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **com.rti.dds.infrastructure.Entity** (p. 1029). This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 1029) classes (**com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.dds.subscription.DataReader** (p. 450)), so that the policies that are meaningful to each **com.rti.dds.infrastructure.Entity** (p. 1029) can be retrieved. For example, see **com.rti.dds.domain.DomainParticipant.get_qos** (p. 715)

Possible error codes are **Standard Return Codes** (p. 261).

8.108.2.3 set_listener (abstract)

This operation installs a **com.rti.dds.infrastructure.Listener** (p. 1236) on the **com.rti.dds.infrastructure.Entity** (p. 1029). The listener will only be invoked on the changes of communication status indicated by the specified `mask`.

This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 1029) classes (**com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.dds.subscription.DataReader** (p. 450)), so that the listener is of the concrete type suitable to the particular **com.rti.dds.infrastructure.Entity** (p. 1029).

There are two components involved when setting up listeners: the listener itself and the mask. Both of these can be NULL.

Listeners for some Entities derive from the Connext DDS Listeners for related Entities. This means that the derived **Listener** (p. 1236) has all of the methods of its parent class. You can install Listeners at all levels of the object hierarchy. At the top is the DomainParticipantListener; only one can be installed in a DomainParticipant. Then every Subscriber and Publisher can have their own **Listener** (p. 1236). Finally, each Topic, DataReader and DataWriter can have their own listeners. All are optional.

Suppose, however, that an **Entity** (p. 1029) does not install a **Listener** (p. 1236), or installs a **Listener** (p. 1236) that does not have particular communication status selected in the bitmask. In this case, if/when that particular status changes for that **Entity** (p. 1029), the corresponding **Listener** (p. 1236) for that **Entity** (p. 1029)'s parent is called. Status changes are "propagated" from child **Entity** (p. 1029) to parent **Entity** (p. 1029) until a **Listener** (p. 1236) is found that is registered for that status. Connext DDS will give up and drop the status-change event only if no Listeners have been installed in the object hierarchy to be called back for the specific status.

The following table describes the effect of different combinations of Listeners and Status Bit Masks considering the hierarchical processing.

Table 8.708 Effect of Different Combinations of Listeners and Status Bit Masks

	No Bits Set in Mask	Some/All Bits Set in Mask
Listener (p. 1236) is Specified	Connext DDS finds the next most relevant listener for the changed status	For the statuses that are enabled in the mask, the most relevant listener will be called. The 'statusChangedFlag' for the relevant status is reset
Listener (p. 1236) is NULL	Connext DDS behaves as if the listener is not installed and finds the next most relevant listener for that status	Connext DDS behaves as if the listener is installed, but the callback is doing nothing. This is called a "nil" listener

Postcondition

Only one listener can be attached to each **com.rti.dds.infrastructure.Entity** (p. 1029). If a listener was already set, the operation `set_listener()` will replace it with the new one. Consequently, if the value null is passed for the listener parameter to the set_listener operation, any existing listener will be removed.

8.108.2.4 get_listener (abstract)

This operation allows access to the existing **com.rti.dds.infrastructure.Listener** (p. 1236) attached to the **com.rti.dds.infrastructure.Entity** (p. 1029).

This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 1029) classes (**com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.publication.DataWriter** (p. 553), **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.dds.subscription.DataReader** (p. 450)) so that the listener is of the concrete type suitable to the particular **com.rti.dds.infrastructure.Entity** (p. 1029).

If no listener is installed on the **com.rti.dds.infrastructure.Entity** (p. 1029), this operation will return null.

8.108.3 Member Function Documentation

8.108.3.1 enable()

```
void enable ( )
```

Enables the **com.rti.dds.infrastructure.Entity** (p. 1029).

This operation enables the **Entity** (p. 1029). **Entity** (p. 1029) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 234) QoS policy on the corresponding factory for the **com.rti.dds.infrastructure.Entity** (p. 1029).

By default, **ENTITY_FACTORY** (p. 234) is set so that it is not necessary to explicitly call **com.rti.dds.infrastructure.Entity.enable** (p. 1032) on newly created entities.

The **com.rti.dds.infrastructure.Entity.enable** (p. 1032) operation is idempotent. Calling enable on an already enabled **Entity** (p. 1029) returns OK and has no effect.

If a **com.rti.dds.infrastructure.Entity** (p. 1029) has not yet been enabled, the following kinds of operations may be invoked on it:

- set or get the QoS policies (including default QoS policies) and listener
- **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034)
- 'factory' operations
- **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034) and other get status operations (although the status of a disabled entity never changes)
- 'lookup' operations

Other operations may explicitly state that they may be called on disabled entities; those that do not will return the error **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

It is legal to delete an **com.rti.dds.infrastructure.Entity** (p. 1029) that has not been enabled by calling the proper operation on its factory .

Entities created from a factory **Entity** (p. 1029) that is disabled are created disabled, regardless of the setting of the **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 1035).

Calling enable on an **Entity** (p. 1029) whose factory **Entity** (p. 1029) is not enabled will fail and return **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

If **com.rti.dds.infrastructure.EntityFactoryQosPolicy.autoenable_created_entities** (p. 1036) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **com.rti.dds.publication.Publisher** (p. 1466) will enable all its contained **com.rti.dds.publication.DataWriter** (p. 553) objects)

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions

One	of the **Standard Return Codes** (p. 261), **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.108.3.2 get_statuscondition()

```
StatusCondition get_statuscondition ( )
```

Allows access to the **com.rti.dds.infrastructure.StatusCondition** (p. 1699) associated with the **com.rti.dds.infrastructure.Entity** (p. 1029).

The returned condition can then be added to a **com.rti.dds.infrastructure.WaitSet** (p. 1973) so that the application can wait for specific status changes that affect the **com.rti.dds.infrastructure.Entity** (p. 1029).

Returns

the status condition associated with this entity.

8.108.3.3 get_status_changes()

```
int get_status_changes ( )
```

Retrieves the list of communication statuses in the **com.rti.dds.infrastructure.Entity** (p. 1029) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_∗_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 1029) itself and does not include statuses that apply to contained entities.

Returns

list of communication statuses in the **com.rti.dds.infrastructure.Entity** (p. 1029) that are triggered.

See also

Status Kinds (p. 262)

8.108.3.4 get_instance_handle()

```
InstanceHandle_t get_instance_handle ( )
```

Allows access to the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) associated with the **com.rti.dds.infrastructure.Entity** (p. 1029).

This operation returns the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) that represents the **com.rti.dds.infrastructure.Entity** (p. 1029).

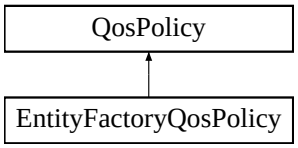
Returns

the instance handle associated with this entity.

8.109 EntityFactoryQosPolicy Class Reference

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.

Inheritance diagram for EntityFactoryQosPolicy:



Public Attributes

- boolean **autoenable_created_entities**

  Specifies whether the entity acting as a factory automatically enables the instances it creates.

8.109.1 Detailed Description

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 1029) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 1029) types.

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 761), **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **YES** (p. 256)

8.109.2 Usage

This policy controls the behavior of the **com.rti.dds.infrastructure.Entity** (p. 1029) as a factory for other entities. It controls whether or not child entities are created in the enabled state.

RTI Connext uses a factory design pattern for creating DDS Entities. That is, a parent entity must be used to create child entities. DomainParticipants create Topics, Publishers and Subscribers. Publishers create DataWriters. Subscribers create DataReaders.

By default, a child object is enabled upon creation (initialized and may be actively used). With this QoS policy, a child object can be created in a disabled state. A disabled entity is only partially initialized and cannot be used until the entity is enabled. Note: an entity can only be *enabled*; it cannot be *disabled* after it has been enabled.

This QoS policy is useful to synchronize the initialization of DDS Entities. For example, when a **com.rti.dds.↩ subscription.DataReader** (p. 450) is created in an enabled state, its existence is immediately propagated for discovery and the **com.rti.dds.subscription.DataReader** (p. 450) object's listener called as soon as data is received. The initialization process for an application may extend beyond the creation of the **com.rti.dds.subscription.DataReader** (p. 450), and thus, it may not be desireable for the **com.rti.dds.subscription.DataReader** (p. 450) to start to receive or process any data until the initialization process is complete. So by creating readers in a disabled state, your application can make sure that no data is received until the rest of the application initialization is complete, and at that time, enable the them.

Note: if an entity is disabled, then all of the child entities it creates will be disabled, too, regardless of the setting of this QoS policy. However, enabling a disabled entity will enable all of its children if this QoS policy is set to automatically enable children entities.

This policy is mutable. A change in the policy affects only the entities created after the change, not any previously created entities.

8.109.3 Member Data Documentation

8.109.3.1 autoenable_created_entities

```
boolean autoenable_created_entities
```

Specifies whether the entity acting as a factory automatically enables the instances it creates.

The setting of `autoenable_created_entities` to com.rti.dds.infrastructure.true indicates that the factory `create_<entity>` operation(s) will automatically invoke the **com.rti.dds.infrastructure.Entity.enable** (p. 1032) operation each time a new **com.rti.dds.infrastructure.Entity** (p. 1029) is created. Therefore, the **com.rti.dds.↩ infrastructure.Entity** (p. 1029) returned by `create_<entity>` will already be enabled. A setting of com.rti.↩ dds.infrastructure.false indicates that the **com.rti.dds.infrastructure.Entity** (p. 1029) will not be automatically enabled. Your application will need to call **com.rti.dds.infrastructure.Entity.enable** (p. 1032) itself.

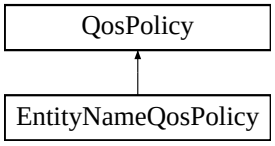
The default setting of `autoenable_created_entities` = com.rti.dds.infrastructure.true means that, by default, it is not necessary to explicitly call **com.rti.dds.infrastructure.Entity.enable** (p. 1032) on newly created entities.

[default] com.rti.dds.infrastructure.true

8.110 EntityNameQosPolicy Class Reference

Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.↩ Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.↩ rti.dds.subscription.Subscriber** (p. 1730), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Inheritance diagram for EntityNameQosPolicy:



Public Attributes

- String **name**

  The name of the entity.
- String **role_name**

  The entity role name.

8.110.1 Detailed Description

Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.↩ Publisher** (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). Except for **com.rti.dds.publication.Publisher** (p. 1466) and **com.↩ rti.dds.subscription.Subscriber** (p. 1730), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.↩ rti.dds.publication.Publisher** (p. 1466), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.↩ publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO;
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.110.2 Usage

The name and role name can be at most 255 characters in length.

8.110.3 Member Data Documentation

8.110.3.1 name

```
String name
```

The name of the entity.

[default] null

[range] Null terminated string with length not exceeding 255. It can be null.

8.110.3.2 role_name

```
String role_name
```

The entity role name.

With Durable Subscriptions this name is used to specify to which Durable Subscription the **com.rti.dds.subscription.↩ DataReader** (p. 450) belongs.

With Collaborative DataWriters this name is used to specify to which endpoint group the **com.rti.dds.publication.↩ DataWriter** (p. 553) belongs.

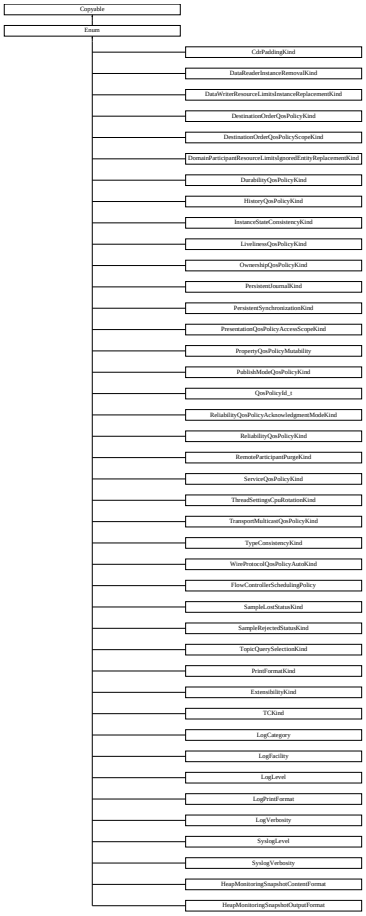
[range] Null terminated string with length not exceeding 255. It can be null.

[default] null

8.111 Enum Class Reference

A superclass for all type-safe enumerated types.

Inheritance diagram for Enum:



Public Member Functions

- final int **ordinal** ()

  The integral value of this enumerated constant.
- Object **copy_from** (Object src)
- final String **name** ()

  The name of this enum constant, as declared in the enum declaration.
- final String **toString** ()

  The string value of this enum constant.

Protected Member Functions

- **Enum** (String **name**, int **ordinal**)

  The constructor.
- final Object **clone** () throws CloneNotSupportedException

8.111.1 Detailed Description

A superclass for all type-safe enumerated types.

This class is not part of the DDS specification *per se*. It has been introduced to facilitate the implementation of the numerous enumerated types in the specification and is based on the Java enumeration JSR. See `http://www.jcp.org/aboutJava/communityprocess/jsr/tiger/enum.html`.

8.111.2 Constructor & Destructor Documentation

8.111.2.1 Enum()

```
Enum (
            String name,
            int ordinal )  [protected]
```

The constructor.

Parameters

ordinal	The value of the ordinal field of the new enumerated constant.
name	The value of the name field of the new enumerated constant.

See also

com.rti.dds.util.Enum.ordinal (p. 1040)

com.rti.dds.util.Enum.name (p. 1041)

References **Enum.name()**, and **Enum.ordinal()**.

8.111.3 Member Function Documentation

8.111.3.1 ordinal()

```
final int ordinal ( )
```

The integral value of this enumerated constant.

For example, in an IDL definition like this:

```
enum Foo {
```

```
    BAR = 2
};
```

...the value of `Foo.BAR.ordinal()` will be 2. If the assignment ("<code>= 2</code>") is omitted, the ordinal value will be 0.

Referenced by **TypeCodeFactory.create_enum_tc()**, **Enum.Enum()**, **DynamicData.from_string()**, **Logger.get_print_format_by_log_level()**, **Logger.get_verbosity_by_category()**, **Logger.set_print_format()**, **Logger.set_print_format_by_log_level()**, **Logger.set_verbosity()**, **Logger.set_verbosity_by_category()**, and **DynamicData.to_string()**.

8.111.3.2 copy_from()

```
Object copy_from (
             Object src )
```

This is the implementation of the `Copyable` interface. While this implementation is not strictly a copy it can have the same effect. In order to use it properly, assign the result of the operation to the member that is the target of the copy. So, for example: myEnumField = myEnumField.copy_from(anotherInstanceOfEnum); Since **Enum** (p. 1038)s are immutable there cannot be a true copy made but this method will return a reference to the same enumerate as `anotherInstanceOfEnum`.

Returns

returns `src`

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

8.111.3.3 name()

```
final String name ( )
```

The name of this enum constant, as declared in the enum declaration.

Most programmers should use the **com.rti.dds.util.Enum.toString** (p. 1041) method rather than accessing this field.

Referenced by **Enum.Enum()**.

8.111.3.4 toString()

```
final String toString ( )
```

The string value of this enum constant.

See also

com.rti.dds.util.Enum.name (p. 1041)

Returns

the name of this enum constant

8.111.3.5 clone()

```
final Object clone ( ) throws CloneNotSupportedException  [protected]
```

This method resolves the serialized data to an instance of this class. It is required functionality for all concrete subclasses but we've decided to keep it commented out for now as it will force customers to regenerate their Java code (if their IDL contains enumerations). We (RTI) should uncomment this method to check that all subclasses have implemented this method so we can avoid CORE-3759 and CORE-5292 from happening again.

Returns

One of the existing instance of the enumeration (to replace the instance created by the Java deserialization process).

Exceptions

ObjectStreamException	

8.112 EnumMember Class Reference

A description of a member of an enumeration.

Inherits Serializable.

Public Member Functions

- **EnumMember** (String **name**, int **ordinal**)

Public Attributes

- String **name**

  The name of the enumeration member.
- int **ordinal**

  The value associated the the enumeration member.

8.112.1 Detailed Description

A description of a member of an enumeration.

See also

com.rti.dds.typecode.TypeCodeFactory.create_enum_tc (p. 1928)

8.112.2 Constructor & Destructor Documentation

8.112.2.1 EnumMember()

```
EnumMember (
            String name,
            int ordinal )
```

Constructs an **EnumMember** (p. 1042) object initialized with the given values.

References **EnumMember.name**, and **EnumMember.ordinal**.

8.112.3 Member Data Documentation

8.112.3.1 name

```
String name
```

The name of the enumeration member.

Cannot be null.

Referenced by **EnumMember.EnumMember()**, and **TypeCode.member_name()**.

8.112.3.2 ordinal

```
int ordinal
```

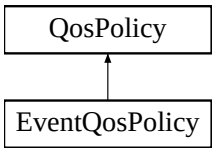
The value associated the the enumeration member.

Referenced by **EnumMember.EnumMember()**, and **TypeCode.member_ordinal()**.

8.113 EventQosPolicy Class Reference

Settings for event.

Inheritance diagram for EventQosPolicy:



Public Attributes

- final **ThreadSettings_t thread**
  Event thread QoS.
- int **initial_count**
  The initial number of events.
- int **max_count**
  The maximum number of events.

8.113.1 Detailed Description

Settings for event.

In a **com.rti.dds.domain.DomainParticipant** (p. 670), a thread is dedicated to handle all timed events, including checking for timeouts and deadlines and executing internal and user-defined timeout or exception handling routines/callbacks.

This QoS policy allows you to configure thread properties such as priority level and stack size. You can also configure the maximum number of events that can be posted to the event thread. By default, a **com.rti.dds.domain.Domain↩ Participant** (p. 670) will dynamically allocate memory as needed for events posted to the event thread. However, by setting a maximum value or setting the initial and maximum value to be the same, you can either bound the amount of memory allocated for the event thread or prevent a **com.rti.dds.domain.DomainParticipant** (p. 670) from dynamically allocating memory for the event thread after initialization.

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.113.2 Member Data Documentation

8.113.2.1 thread

```
final ThreadSettings_t thread
```

Event thread QoS.

There is only one event thread.

Priority:
[default] The actual value depends on your architecture:
For Windows: -2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

Stack Size:
[default] The actual value depends on your architecture:
For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

Mask:
[default] mask = **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_FLOATING_POINT** (p. 1797) | **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1797)

8.113.2.2 initial_count

```
int initial_count
```

The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

8.113.2.3 max_count

```
int max_count
```

The maximum number of events.

The maximum number of events. If the limit is reached, no new event can be added.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_count

8.114 ExpressionProperty Class Reference

Provides additional information about the filter expression passed to **com.rti.dds.topic.WriterContentFilter.writer_↩ compile** (p. 2003) .

Inherits Struct.

Public Attributes

- boolean **key_only_filter** = false

  Indicates if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.
- boolean **writer_side_filter_optimization** = false

  Indicates if the filter implementation can cache the filtering result for the provided expression.

8.114.1 Detailed Description

Provides additional information about the filter expression passed to **com.rti.dds.topic.WriterContentFilter.writer_↩ compile** (p. 2003) .

It is used by the filter implementation to indicate to the middleware whether or not the **com.rti.dds.topic.Writer↩ ContentFilter** (p. 2002) will cache the result of filter evaluation.

8.114.2 Member Data Documentation

8.114.2.1 key_only_filter

```
boolean key_only_filter = false
```

Indicates if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.

When this field is set to com.rti.dds.infrastructure.true, it indicates to RTI Connext that the filter expression is based only on key fields.

8.114.2.2 writer_side_filter_optimization

```
boolean writer_side_filter_optimization = false
```

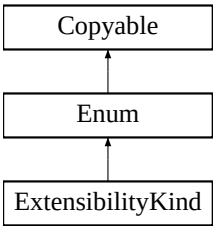
Indicates if the filter implementation can cache the filtering result for the provided expression.

When this field is set to com.rti.dds.infrastructure.true, RTI Connext will do no caching or explicit filter evaluation for the associated **com.rti.dds.subscription.DataReader** (p. 450). Instead, it will rely on the filter implementation to provide appropriate results.

8.115 ExtensibilityKind Class Reference

Type to indicate the extensibility of a type.

Inheritance diagram for ExtensibilityKind:



Static Public Attributes

- static final **ExtensibilityKind FINAL_EXTENSIBILITY**
  Specifies that a type has FINAL extensibility.
- static final **ExtensibilityKind EXTENSIBLE_EXTENSIBILITY**
  Specifies that a type has EXTENSIBLE extensibility.
- static final **ExtensibilityKind MUTABLE_EXTENSIBILITY**
  Specifies that a type has MUTABLE extensibility.

Additional Inherited Members

8.115.1 Detailed Description

Type to indicate the extensibility of a type.

8.116 FilterSampleInfo Class Reference

Provides meta information associated with the sample.

Inherits Struct.

Public Attributes

- **SampleIdentity_t related_sample_identity**
  The identity of another sample related to this one.
- **GUID_t related_reader_guid**
  Identifies a DataReader that is logically related to the sample.
- **GUID_t related_source_guid**
  Identifies the application logical data source that is related to the sample being written.

8.116.1 Detailed Description

Provides meta information associated with the sample.

This can be used by a content filter to perform filtering on meta data.

8.116.2 Member Data Documentation

8.116.2.1 related_sample_identity

```
SampleIdentity_t related_sample_identity
```

The identity of another sample related to this one.

The value of this field identifies another sample that is logically related to the one that is written. For example, the **com.rti.dds.publication.DataWriter** (p. 553) created by a Replier uses this field to associate the identity of the request sample with a reponse sample.

To specify that there is no related sample identity, use the value com.rti.dds.infrastructure.SampleIdentity_t.Sample↩ Identity_t.UNKNOWN_SAMPLE_IDENTITY.

A **com.rti.dds.subscription.DataReader** (p. 450) can inspect the related sample identity of a received sample by accessing the fields **com.rti.dds.subscription.SampleInfo.related_original_publication_virtual_guid** (p. 1645) and **com.rti.dds.subscription.SampleInfo.related_original_publication_virtual_sequence_number** (p. 1645).

8.116.2.2 related_reader_guid

```
GUID_t related_reader_guid
```

Identifies a DataReader that is logically related to the sample.

The related_reader_guid can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.related_reader↩ _guid** (p. 2001) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

8.116.2.3 related_source_guid

```
GUID_t related_source_guid
```

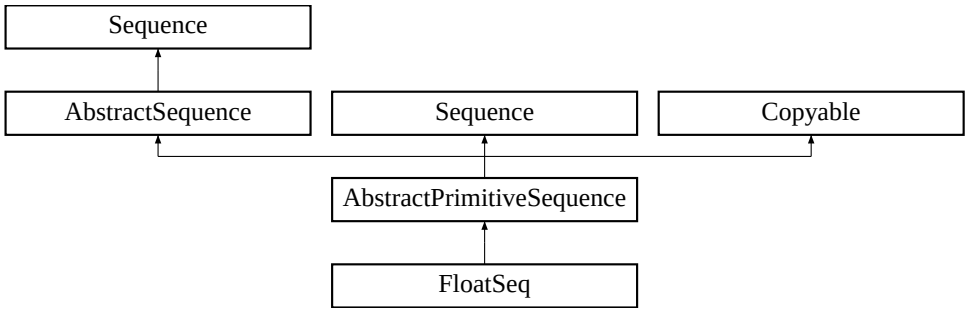
Identifies the application logical data source that is related to the sample being written.

The related_source_guid can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.related_source↩ _guid** (p. 2000) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

8.117 FloatSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < float >

Inheritance diagram for FloatSeq:



Public Member Functions

- **FloatSeq** ()

  Constructs an empty sequence of floats with an initial maximum of zero.
- **FloatSeq** (int initialMaximum)

  Constructs an empty sequence of floats with the given initial maximum.
- **FloatSeq** (float[ ] floats)

  Constructs a new sequence containing the given floats.
- boolean **addAllFloat** (float[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- boolean **addAllFloat** (float[ ] elements)
- void **addFloat** (float element)

  Append the element to the end of the sequence.
- void **addFloat** (int index, float element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- float **getFloat** (int index)

  Returns the float at the given index.
- float **setFloat** (int index, float element)

  Set the new float at the given index and return the old float.
- void **setFloat** (int dstIndex, float[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- float[ ] **toArrayFloat** (float[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  *A wrapper for **getFloat(int)** (p. 1052) that returns a java.lang.Float.*
- Object **set** (int index, Object element)

  *A wrapper for **setFloat()** (p. 1052).*
- void **add** (int index, Object element)

  A wrapper for addFloat(int, int).

8.117.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < float >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

float

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.117.2 Constructor & Destructor Documentation

8.117.2.1 FloatSeq() [1/3]

```
FloatSeq ( )
```

Constructs an empty sequence of floats with an initial maximum of zero.

8.117.2.2 FloatSeq() [2/3]

```
FloatSeq (
            int initialMaximum )
```

Constructs an empty sequence of floats with the given initial maximum.

8.117.2.3 FloatSeq() [3/3]

```
FloatSeq (
            float[] floats )
```

Constructs a new sequence containing the given floats.

Parameters

floats	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **FloatSeq.addAllFloat()**.

8.117.3 Member Function Documentation

8.117.3.1 addAllFloat() [1/2]

```
boolean addAllFloat (
            float[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **FloatSeq.addAllFloat()**, and **FloatSeq.FloatSeq()**.

8.117.3.2 addAllFloat() [2/2]

```
boolean addAllFloat (
            float[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **FloatSeq.addAllFloat()**.

8.117.3.3 addFloat() [1/2]

```
void addFloat (
            float element )
```

Append the element to the end of the sequence.

Referenced by **FloatSeq.add()**.

8.117.3.4 addFloat() [2/2]

```
void addFloat (
            int index,
            float element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.117.3.5 getFloat()

```
float getFloat (
            int index )
```

Returns the float at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **FloatSeq.get()**.

8.117.3.6 setFloat() [1/2]

```
float setFloat (
            int index,
            float element )
```

Set the new float at the given index and return the old float.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **FloatSeq.set()**.

8.117.3.7 setFloat() [2/2]

```
void setFloat (
            int dstIndex,
            float[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.117.3.8 toArrayFloat()

```
float[] toArrayFloat (
            float[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.117.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. `1055`), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_float_seq()**, and **DynamicData.set_float_seq()**.

8.117.3.10 get()

```
Object get (
             int index )
```

A wrapper for **getFloat(int)** (p. 1052) that returns a java.lang.Float.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **FloatSeq.getFloat()**.

8.117.3.11 set()

```
Object set (
             int index,
             Object element )
```

A wrapper for **setFloat()** (p. 1052).

Exceptions

ClassCastException	if the element is not of type Float.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **FloatSeq.setFloat()**.

8.117.3.12 add()

```
void add (
             int index,
             Object element )
```

A wrapper for addFloat(int, int).

Exceptions

ClassCastException	if the element is not of type Float.

See also

java.util.List::add(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **FloatSeq.addFloat()**.

8.118 FlowController Interface Reference

<<***interface***>> *(*p. *156)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances are allowed to write data.

Public Member Functions

- String **get_name** ()

  *Returns the name of the **com.rti.dds.publication.FlowController*** (p. *1055*).
- **DomainParticipant get_participant** ()

  *Returns the **com.rti.dds.domain.DomainParticipant*** (p. *670*) *to which the **com.rti.dds.publication.FlowController*** (p. *1055*) *belongs.*
- void **set_property** (**FlowControllerProperty_t** prop)

  *Sets the **com.rti.dds.publication.FlowController*** (p. *1055*) *property.*
- void **get_property** (**FlowControllerProperty_t** prop)

  *Gets the **com.rti.dds.publication.FlowController*** (p. *1055*) *property.*
- void **trigger_flow** ()

  *Provides an external trigger to the **com.rti.dds.publication.FlowController*** (p. *1055*).

Static Public Attributes

- static final String **DEFAULT_FLOW_CONTROLLER_NAME**

  [default] *Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** (p. *1497*) *that refers to the built-in default flow controller.*
- static final String **FIXED_RATE_FLOW_CONTROLLER_NAME**

  *Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** (p. *1497*) *that refers to the built-in fixed-rate flow controller.*
- static final String **ON_DEMAND_FLOW_CONTROLLER_NAME**

  *Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** (p. *1497*) *that refers to the built-in on-demand flow controller.*

8.118.1 Detailed Description

<<***interface***>> (p. *156*) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances are allowed to write data.

QoS:

com.rti.dds.publication.FlowControllerProperty_t (p. 1059)

8.118.2 Member Function Documentation

8.118.2.1 get_name()

```
String get_name ( )
```

Returns the name of the **com.rti.dds.publication.FlowController** (p. 1055).

Returns

The name of the **com.rti.dds.publication.FlowController** (p. 1055).

8.118.2.2 get_participant()

```
DomainParticipant get_participant ( )
```

Returns the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.publication.FlowController** (p. 1055) belongs.

Returns

The **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.publication.FlowController** (p. 1055) belongs.

8.118.2.3 set_property()

```
void set_property (
            FlowControllerProperty_t prop )
```

Sets the **com.rti.dds.publication.FlowController** (p. 1055) property.

This operation modifies the property of the **com.rti.dds.publication.FlowController** (p. 1055).

Once a **com.rti.dds.publication.FlowController** (p. 1055) has been instantiated, only the **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060) can be changed. The **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059) is immutable.

A new **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 1065) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters

prop	<<***in***>> *(*p. *156)* The new **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059). Property must be consistent. Immutable fields cannot be changed after **com.rti.dds.publication.FlowController** (p. 1055) has been created. The special value **com.rti.dds.domain.DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 49) can be used to indicate that the property of the **com.rti.dds.publication.FlowController** (p. 1055) should be changed to match the current default **com.rti.dds.publication.FlowControllerProperty_t** (p. 1059) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.publication.FlowControllerProperty_t (p. 1059) for rules on consistency among property values.

8.118.2.4 get_property()

```
void get_property (
            FlowControllerProperty_t prop )
```

Gets the **com.rti.dds.publication.FlowController** (p. 1055) property.

Parameters

prop	<<***in***>> *(p. 156)* **com.rti.dds.publication.FlowController** (p. 1055) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.118.2.5 trigger_flow()

```
void trigger_flow ( )
```

Provides an external trigger to the **com.rti.dds.publication.FlowController** (p. 1055).

Typically, a **com.rti.dds.publication.FlowController** (p. 1055) uses an internal trigger to periodically replenish its tokens. The period by which this trigger is called is determined by the **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 1065) property setting.

This function provides an additional, external trigger to the **com.rti.dds.publication.FlowController** (p. 1055). This trigger adds **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.tokens_added_per_period** (p. 1064) tokens each time it is called (subject to the other property settings of the **com.rti.dds.publication.FlowController** (p. 1055)).

An *on-demand* **com.rti.dds.publication.FlowController** (p. 1055) can be created with a **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) as **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 1065), in which case the only trigger source is external (i.e. the **FlowController** (p. 1055) is solely triggered by the user on demand).

com.rti.dds.publication.FlowController.trigger_flow (p. 1058) can be called on both strict *on-demand* **FlowController** (p. 1055) and hybrid **FlowController** (p. 1055) (internally and externally triggered).

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.119 FlowControllerProperty_t Class Reference

Determines the flow control characteristics of the **com.rti.dds.publication.FlowController** (p. 1055).

Inherits Struct.

Public Attributes

- **FlowControllerSchedulingPolicy scheduling_policy**
  Scheduling policy.
- **FlowControllerTokenBucketProperty_t token_bucket**
  Settings for the token bucket.

8.119.1 Detailed Description

Determines the flow control characteristics of the **com.rti.dds.publication.FlowController** (p. 1055).

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances are serviced and how much data they are allowed to send.

Note that these settings apply directly to the **com.rti.dds.publication.FlowController** (p. 1055), and do not depend on the number of **com.rti.dds.publication.DataWriter** (p. 553) instances the **com.rti.dds.publication.FlowController** (p. 1055) is servicing. For instance, the specified flow rate does *not* double simply because two **com.rti.dds.↩ publication.DataWriter** (p. 553) instances are waiting to write.

Entity:

com.rti.dds.publication.FlowController (p. 1055)

Properties:

RxO (p. 256) = N/A

Changeable (p. 256) = **NO** (p. 256) for **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059), **YES** (p. 256) for **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 1060). However, the special value of **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) as **com.rti.dds.publication.↩ FlowControllerTokenBucketProperty_t.period** (p. 1065) is strictly used to create an *on-demand* **com.rti.dds.↩ publication.FlowController** (p. 1055). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.119.2 Member Data Documentation

8.119.2.1 scheduling_policy

```
FlowControllerSchedulingPolicy scheduling_policy
```

Scheduling policy.

Determines the scheduling policy for servicing the **com.rti.dds.publication.DataWriter** (p. 553) instances associated with the **com.rti.dds.publication.FlowController** (p. 1055).

[default] com.rti.dds.publication.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy.EDF_FLOW_↩ CONTROLLER_SCHED_POLICY

8.119.2.2 token_bucket

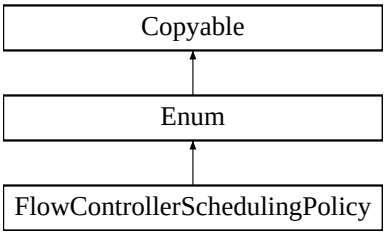
```
FlowControllerTokenBucketProperty_t token_bucket
```

Settings for the token bucket.

8.120 FlowControllerSchedulingPolicy Class Reference

Kinds of flow controller scheduling policy.

Inheritance diagram for FlowControllerSchedulingPolicy:



Static Public Attributes

- static final **FlowControllerSchedulingPolicy RR_FLOW_CONTROLLER_SCHED_POLICY**
  Indicates to flow control in a round-robin fashion.
- static final **FlowControllerSchedulingPolicy EDF_FLOW_CONTROLLER_SCHED_POLICY**
  Indicates to flow control in an earliest-deadline-first fashion.
- static final **FlowControllerSchedulingPolicy HPF_FLOW_CONTROLLER_SCHED_POLICY**
  Indicates to flow control in a highest-priority-first fashion.

Additional Inherited Members

8.120.1 Detailed Description

Kinds of flow controller scheduling policy.

Samples written by an asynchronous **com.rti.dds.publication.DataWriter** (p. 553) are not sent in the context of the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) call. Instead, the middleware puts the samples in a queue for future processing. The **com.rti.dds.publication.FlowController** (p. 1055) associated with each asynchronous **Data↩ Writer** (p. 553) instance determines when the samples are actually sent.

Each **com.rti.dds.publication.FlowController** (p. 1055) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **com.rti.dds.publication.DataWriter** (p. 553) instances associated with the flow controller, are placed in the queues that correspond to the intended destinations of the sample.

When tokens become available, a flow controller must decide which queue(s) to grant tokens first. This is determined by the flow controller's scheduling policy. Once a queue has been granted tokens, it is serviced by the asynchronous publishing thread. The queued up samples will be coalesced and sent to the corresponding destination. The number of samples sent depends on the data size and the number of tokens granted.

QoS:

com.rti.dds.publication.FlowControllerProperty_t (p. 1059)

8.120.2 Member Data Documentation

8.120.2.1 RR_FLOW_CONTROLLER_SCHED_POLICY

```
final FlowControllerSchedulingPolicy RR_FLOW_CONTROLLER_SCHED_POLICY [static]
```

Indicates to flow control in a round-robin fashion.

Whenever tokens become available, the flow controller distributes the tokens uniformly across all of its (non-empty) destination queues. No destinations are prioritized. Instead, all destinations are treated equally and are serviced in a round-robin fashion.

8.120.2.2 EDF_FLOW_CONTROLLER_SCHED_POLICY

```
final FlowControllerSchedulingPolicy EDF_FLOW_CONTROLLER_SCHED_POLICY [static]
```

Indicates to flow control in an earliest-deadline-first fashion.

A sample's deadline is determined by the time it was written plus the latency budget of the **DataWriter** (p. 553) at the time of the write call (as specified in the **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1231)). The relative priority of a flow controller's destination queue is determined by the earliest deadline across all samples it contains.

When tokens become available, the **com.rti.dds.publication.FlowController** (p. 1055) distributes tokens to the destination queues in order of their deadline priority. In other words, the queue containing the sample with the earliest deadline is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal deadline value, the corresponding destination queues are serviced in a round-robin fashion.

Hence, under the default **com.rti.dds.infrastructure.LatencyBudgetQosPolicy.duration** (p. 1232) setting, an EDF↩ _FLOW_CONTROLLER_SCHED_POLICY **com.rti.dds.publication.FlowController** (p. 1055) preserves the order in which the user calls **com.rti.ndds.example.FooDataWriter.write** (p. 1105) across the DataWriters associated with the flow controller.

Since the **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1231) is mutable, a sample written second may contain an earlier deadline than the sample written first if the **com.rti.dds.infrastructure.LatencyBudgetQosPolicy.↩ duration** (p. 1232) value is sufficiently decreased in between writing the two samples. In that case, if the first sample is not yet written (still in queue waiting for its turn), it inherits the priority corresponding to the (earlier) deadline from the second sample.

In other words, the priority of a destination queue is always determined by the earliest deadline among all samples contained in the queue. This priority inheritance approach is required in order to both honor the updated **com.rti.dds.↩ infrastructure.LatencyBudgetQosPolicy.duration** (p. 1232) and adhere to the **com.rti.dds.publication.DataWriter** (p. 553) in-order data delivery guarantee.

[default] for **com.rti.dds.publication.DataWriter** (p. 553)

8.120.2.3 HPF_FLOW_CONTROLLER_SCHED_POLICY

```
final FlowControllerSchedulingPolicy HPF_FLOW_CONTROLLER_SCHED_POLICY [static]
```

Indicates to flow control in a highest-priority-first fashion.

Determines the next destination queue to service as determined by the publication priority of the **com.rti.dds.↩ publication.DataWriter** (p. 553), channel of multi-channel **DataWriter** (p. 553), or individual sample.

The relative priority of a flow controller's destination queue is determined by the highest publication priority of all samples it contains.

When tokens become available, the **com.rti.dds.publication.FlowController** (p. 1055) distributes tokens to the destination queues in order of their publication priority. In other words, the queue containing the sample with the highest publication priority is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal publication priority, the corresponding destination queues are serviced in a round-robin fashion.

This priority inheritance approach is required in order to both honor the designated publication priority and adhere to the **com.rti.dds.publication.DataWriter** (p. 553) in-order data delivery guarantee.

8.121 FlowControllerTokenBucketProperty_t Class Reference

com.rti.dds.publication.FlowController (p. 1055) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Inherits Struct.

Public Attributes

- int **max_tokens**

  Maximum number of tokens than can accumulate in the token bucket.
- int **tokens_added_per_period**

  The number of tokens added to the token bucket per specified period.
- int **tokens_leaked_per_period**

  The number of tokens removed from the token bucket per specified period.
- **Duration_t period**

  Period for adding tokens to and removing tokens from the bucket.
- int **bytes_per_token**

  Maximum number of bytes allowed to send for each token available.

8.121.1 Detailed Description

com.rti.dds.publication.FlowController (p. 1055) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Asynchronously published samples are queued up and transmitted based on the token bucket flow control scheme. The token bucket contains tokens, each of which represents a number of bytes. Samples can be sent only when there are sufficient tokens in the bucket. As samples are sent, tokens are consumed. The number of tokens consumed is proportional to the size of the data being sent. Tokens are replenished on a periodic basis.

The rate at which tokens become available and other token bucket properties determine the network traffic flow.

Note that if the same sample must be sent to multiple destinations, separate tokens are required for each destination. Only when multiple samples are destined to the same destination will they be co-alesced and sent using the same token(s). In other words, each token can only contribute to a single network packet.

Entity:

com.rti.dds.publication.FlowController (p. 1055)

Properties:

RxO (p. 256) = N/A

Changeable (p. 256) = **YES** (p. 256). However, the special value of **com.rti.dds.infrastructure.Duration_t.↩ DURATION_INFINITE** (p. 846) as **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 1065) is strictly used to create an *on-demand* **com.rti.dds.publication.FlowController** (p. 1055). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.121.2 Member Data Documentation

8.121.2.1 max_tokens

```
int max_tokens
```

Maximum number of tokens than can accumulate in the token bucket.

The number of tokens in the bucket will never exceed this value. Any excess tokens are discarded. This property value, combined with **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.bytes_per_token** (p. 1065), determines the maximum allowable data burst.

Use **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.121.2.2 tokens_added_per_period

```
int tokens_added_per_period
```

The number of tokens added to the token bucket per specified period.

com.rti.dds.publication.FlowController (p. 1055) transmits data only when tokens are available. Tokens are periodically replenished. This field determines the number of tokens added to the token bucket with each periodic replenishment.

Available tokens are distributed to associated **com.rti.dds.publication.DataWriter** (p. 553) instances based on the **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059).

Use **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) to add the maximum number of tokens allowed by **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.max_tokens** (p. 1064).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.121.2.3 tokens_leaked_per_period

```
int tokens_leaked_per_period
```

The number of tokens removed from the token bucket per specified period.

com.rti.dds.publication.FlowController (p. 1055) transmits data only when tokens are available. When tokens are replenished and there are sufficient tokens to send all samples in the queue, this property determines whether any or all of the leftover tokens remain in the bucket.

Use **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) to remove all excess tokens from the token bucket once all samples have been sent. In other words, no token accumulation is allowed. When new samples are written after tokens were purged, the earliest point in time at which they can be sent is at the next periodic replenishment.

[default] 0

[range] [0,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.121.2.4 period

```
Duration_t period
```

Period for adding tokens to and removing tokens from the bucket.

com.rti.dds.publication.FlowController (p. 1055) transmits data only when tokens are available. This field determines the period by which tokens are added or removed from the token bucket.

The special value **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) can be used to create an *on-demand* **com.rti.dds.publication.FlowController** (p. 1055), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **com.rti.dds.publication.FlowController.trigger_flow** (p. 1058). This external trigger adds **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.tokens_added_per_period** (p. 1064) tokens each time it is called (subject to the other property settings).

[default] 1 second

[range] [1 nanosec, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.121.2.5 bytes_per_token

```
int bytes_per_token
```

Maximum number of bytes allowed to send for each token available.

com.rti.dds.publication.FlowController (p. 1055) transmits data only when tokens are available. This field determines the number of bytes that can actually be transmitted based on the number of tokens.

Tokens are always consumed in whole by each **com.rti.dds.publication.DataWriter** (p. 553). That is, in cases where **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.bytes_per_token** (p. 1065) is greater than the sample size, multiple samples may be sent to the same destination using a single token (regardless of **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 1059)).

Where fragmentation is required, the fragment size will be **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.bytes_per_token** (p. 1065) or the minimum largest message size across all transports installed with the **com.rti.dds.publication.DataWriter** (p. 553), whichever is less.

Use **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) to indicate that an unlimited number of bytes can be transmitted per token. In other words, a single token allows the recipient **com.rti.dds.publication.DataWriter** (p. 553) to transmit all its queued samples to a single destination. A separate token is required to send to each additional destination.

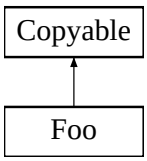
[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1024,**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)]

8.122 Foo Class Reference

A representative user-defined data type.

Inheritance diagram for Foo:



Public Member Functions

- Object **copy_from** (Object src)

8.122.1 Detailed Description

A representative user-defined data type.

Foo (p. 1066) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 1066) is usually defined using IDL syntax and placed in a ".idl" file that is then processed using rtiddsgen. The rtiddsgen utility generates the helper classes com.rti.dds.infrastructure.com.rti.dds.util.Sequence as well as the necessary code for DDS to manipulate the type (serialize it so that it can be sent over the network) as well as the implied **com.rti.ndds.example.FooDataReader** (p. 1067) and **com.rti.ndds.example.FooDataWriter** (p. 1097) types that allow the application to send and receive data of this type.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence, **com.rti.ndds.example.FooDataWriter** (p. 1097), **com.↩ rti.ndds.example.FooDataReader** (p. 1067), **com.rti.ndds.example.FooTypeSupport** (p. 1118), the `Code Generator User's Manual`

8.122.2 Member Function Documentation

8.122.2.1 copy_from()

```
Object copy_from (
            Object src )
```

This is the implementation of the `Copyable` interface. This method will perform a deep copy of `src`. This method could be placed into **`FooTypeSupport`** (p. `1118`).

rather than here by using the **-noCopyable** option to rtiddsgen.

Parameters

src	The Object which contains the data to be copied.

Returns

Returns **this**

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not the same type as `this`.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

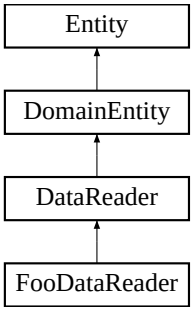
Implements **Copyable** (p. 445).

Referenced by **FooTypeSupport.copy_data()**.

8.123 FooDataReader Class Reference

<<***interface***>> *(p. 156)* <<***generic***>> *(p. 156)* User data type-specific data reader.

Inheritance diagram for FooDataReader:



Public Member Functions

- void **read** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_↩ states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_↩ states, int instance_states)

*Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **read_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.read** (p. 1069) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **take_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Analogous to **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) except it accesses samples via the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation.*

- void **read_next_sample** (**Foo** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **take_next_sample** (**Foo** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **read_instance** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **take_instance** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **read_next_instance** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **take_next_instance** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **read_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *<<**extension**>> (p. 155) Accesses via **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **take_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *<<**extension**>> (p. 155) Accesses via **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **read_next_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **take_next_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **return_loan** (**FooSeq** received_data, **SampleInfoSeq** info_seq)

  *Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450).*

- void **get_key_value** (**Foo** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.

- **InstanceHandle_t lookup_instance** (**Foo** key_holder)

  Retrieves the instance `handle` that corresponds to an instance `key_holder`.

Additional Inherited Members

8.123.1 Detailed Description

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* User data type-specific data reader.

Defines the user data type specific reader interface generated for each application class.

The concrete user data type reader automatically generated by the implementation is an incarnation of this class.

See also

com.rti.dds.subscription.DataReader (p. 450)

com.rti.ndds.example.Foo (p. 1066)

com.rti.ndds.example.FooDataWriter (p. 1097)

the `Code Generator User's Manual`

8.123.2 Member Function Documentation

8.123.2.1 read()

```
void read (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation offers the same functionality and API as **com.rti.ndds.example.FooDataReader.take** (p. 1071) except that the samples returned remain in the **com.rti.dds.subscription.DataReader** (p. 450) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **com.rti.ndds.example.FooDataReader.take()** (p. 1071) for details on the number of samples returned within the received_data and info_seq as well as the order in which the samples appear in these sequences.

The act of reading a sample changes its `sample_state` to com.rti.dds.subscription.SampleStateKind.SampleState↩Kind.READ_SAMPLE_STATE. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to be com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the instance.

Once the application completes its use of the samples, it must 'return the loan' to the **com.rti.dds.subscription.Data↩Reader** (p. 450) by calling the **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **com.rti.dds.subscription.SampleInfo** (p. 1634) objects after the call to **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While the loan must be returned at some point, you do *not* have to do so before the next **com.rti.ndds.example.FooDataReader.take** (p. 1071) call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader** (p. 450) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **com.rti.dds.subscription.DataReader** (p. 450) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529).

Important: If the samples "returned" by this method are loaned from RTI Connext (see **com.rti.ndds.example.FooDataReader.take** (p. 1071) for more information on memory loaning), it is important that their contents not be changed. Because the memory in which the data is stored belongs to the middleware, any modifications made to the data will be seen the next time the same samples are read or taken; the samples will no longer reflect the state that was received from the network.

Parameters

received_data	<<***inout***>> *(*p. *156)* User data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(*p. *156)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
sample_states	<<***in***>> *(*p. *156)* Data samples matching one of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(*p. *156)* Data samples matching one of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(*p. *156)* Data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076),

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_untyped()**.

8.123.2.2 take()

```
void take (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

The operation will return the list of samples received by the **com.rti.dds.subscription.DataReader** (p. 450) since the last **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation that matches the specified com.rti.dds.↩ subscription.SampleStateMask, com.rti.dds.subscription.ViewStateMask and com.rti.dds.subscription.InstanceState↩ Mask.

This operation may fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595) if the **com.rti.dds.↩ infrastructure.DataReaderResourceLimitsQosPolicy.max_outstanding_reads** (p. 534) limit has been exceeded.

The actual number of samples returned depends on the information that has been received by the middleware as well as the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590), **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529) and the characteristics of the data-type that is associated with the **com.rti.dds.subscription.DataReader** (p. 450):

- In the case where the **com.rti.dds.infrastructure.HistoryQosPolicy.kind** (p. 1146) is com.rti.dds.↩ infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, the call will return at most **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) samples for each ALIVE instance and (**com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) +1) samples for each NOT_ALIVE instance. The extra sample is an invalid sample (**com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643) is FALSE) that is used to indicate the instance state transition from ALIVE to NOT_ALIVE.

- The maximum number of samples returned is limited by **com.rti.dds.infrastructure.ResourceLimitsQos↩ Policy.max_samples** (p. 1592), and by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.↩ max_samples_per_read** (p. 534).

- For multiple instances, the number of samples returned is additionally limited by the product (**com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)
  - **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592))

- If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_infos** (p. 532) is limited, the number of samples returned may also be limited if insufficient **com.rti.dds.subscription.SampleInfo** (p. 1634) resources are available.

If the read or take succeeds and the number of samples returned has been limited (by means of a maximum limit, as listed above, or insufficient **com.rti.dds.subscription.SampleInfo** (p. 1634) resources), the call will complete successfully and provide those samples the reader is able to return. The user may need to make additional calls, or return outstanding loaned buffers in the case of insufficient resources, in order to access remaining samples.

Note that in the case where the **com.rti.dds.topic.Topic** (p. 1807) associated with the **com.rti.dds.subscription.DataReader** (p. 450) is bound to a data-type that has no key definition, then there will be at most one instance in the **com.rti.dds.subscription.DataReader** (p. 450). So the per-sample limits will apply.

The act of *taking* a sample removes it from RTI Connext so it cannot be read or taken again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the sample's instance to com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the sample's instance.

After **com.rti.ndds.example.FooDataReader.take** (p. 1071) completes, `received_data` and `info_seq` will be of the same length and contain the received data.

If the sequences are empty (maximum size equals 0) when the **com.rti.ndds.example.FooDataReader.take** (p. 1071) is called, the samples returned in the `received_data` and the corresponding `info_seq` are 'loaned' to the application from buffers provided by the **com.rti.dds.subscription.DataReader** (p. 450). The application can use them as desired and has guaranteed exclusive access to them.

Once the application completes its use of the samples it must 'return the loan' to the **com.rti.dds.subscription.DataReader** (p. 450) by calling the **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) operation.

Important: When you loan data from the middleware, you *must not* keep any pointers to any part of the data samples or the **com.rti.dds.subscription.SampleInfo** (p. 1634) objects after the call to **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While you must call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) at some point, you do *not* have to do so before the next **com.rti.ndds.example.FooDataReader.take** (p. 1071) call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader** (p. 450) of the buffers it needs to receive new samples and eventually samples will start to be lost. The total number of buffers available to the **com.rti.dds.subscription.DataReader** (p. 450) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when **com.rti.ndds.example.FooDataReader.take** (p. 1071) is called, samples are copied to received_data and info_seq. The application will not need to call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

The order of the samples returned to the caller depends on the **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1379).

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS, the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.true, then the returned collection is a list where the relative order of samples as they were written by a DataWriter is preserved also across different instances. In other words, changes made by a single DataWriter are made available to subscribers in the same order in which they occur. Samples belonging to the same instance may or may not be consecutive. This is because, to preserve order within a single DataWriter, it may be necessary to mix samples from different instances.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.false, then the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is com.rti.dds.infrastructure.↩ PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION↩ _QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) is set to com.rti.↩ dds.infrastructure.true, then changes made to instances by a set of DataWriters attached to a common Publisher are made available in the order in which they were written. For this to happen, the application must take the samples using the Subscriber's **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) and **com.↩ rti.dds.subscription.Subscriber.end_access** (p. 1750) operations (see the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the `Core Libraries User's Manual`).

In all of the above cases, the relative order between the samples of one instance is consistent with the **DESTINATION↩ _ORDER** (p. 218) policy:

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is com.rti.dds.infrastructure.↩ DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER↩ _QOS, samples belonging to the same instances will appear in the relative order in which there were received (FIFO, earlier samples ahead of the later samples).

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is com.rti.dds.infrastructure.↩ DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER↩ _QOS, samples belonging to the same instances will appear in the relative order implied by the `source_↩ timestamp` (FIFO, smaller values of `source_timestamp` ahead of the larger values).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

In addition to the collection of samples, the read and take operations also use a collection of **com.rti.dds.↩ subscription.SampleInfo** (p. 1634) structures.

The initial (input) properties of the `received_data` and `info_seq` collections will determine the precise behavior of the read or take operation. For the purposes of this description, the collections are modeled as having these properties:

- the current-length (len, see com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size())
- the maximum length (max_len, see com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.get↩ Maximum)

The initial values of the len and max_len properties for the received_data and info_seq collections govern the behavior of the read and take operations as specified by the following rules:

1. The values of len and max_len properties for the two collections must be identical. Otherwise read/take will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).
2. On successful output, the values of len and max_len will be the same for both collections.
3. If the initial max_len==0, then the received_data and info_seq collections will be filled with elements that are loaned by the **com.rti.dds.subscription.DataReader** (p. 450). On output, len will be set to the number of values returned, and max_len will be set to a value verifying max_len >= len. The use of this variant allows for Zero Copy access to the data and the application will need to return the loan to the **com.rti.dds.↩ publication.DataWriter** (p. 553) using **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).
4. If initial max_len>0 then the read or take operation will copy the received_data values and **com.rti.dds.↩ subscription.SampleInfo** (p. 1634) values into the elements already inside the collections. On output, len will be set to the number of values copied and max_len will remain unchanged. The use of this variant forces a copy but the application can control where the copy is placed and the application will not need to return the loan. The number of samples copied depends on the relative values of max_len and max_samples:
   - If max_samples == LENGTH_UNLIMITED, then at most max_len values will be copied. The use of this variant lets the application limit the number of samples returned to what the sequence can accommodate.
   - If max_samples <= max_len, then at most max_samples values will be copied. The use of this variant lets the application limit the number of samples returned to fewer that what the sequence can accommodate.
   - If max_samples > max_len, then the read or take operation will fail with **com.rti.dds.infrastructure.↩ RETCODE_PRECONDITION_NOT_MET** (p. 1598). This avoids the potential confusion where the application expects to be able to access up to max_samples, but that number can never be returned, even if they are available in the **com.rti.dds.subscription.DataReader** (p. 450), because the output sequence cannot accommodate them.

As described above, upon completion, the received_data and info_seq collections may contain elements loaned from the **com.rti.dds.subscription.DataReader** (p. 450). If this is the case, the application will need to use **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) to return the loan once it is no longer using the received_data in the collection. When **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) completes, the collection will have max_len=0. The application can determine whether it is necessary to return the loan or not based on how the state of the collections when the read/take operation was called However, in many cases it may be simpler to always call **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094), as this operation is harmless (i.e., it leaves all elements unchanged) if the collection does not have a loan.

On output, the collection of **com.rti.ndds.example.Foo** (p. 1066) values and the collection of **com.rti.dds.↩ subscription.SampleInfo** (p. 1634) structures are of the same length and are in a one-to-one correspondence. Each **com.rti.dds.subscription.SampleInfo** (p. 1634) provides information, such as the source_timestamp, the sample_state, view_state, and instance_state, etc., about the corresponding sample. Some elements in the returned collection may not have valid data. If the instance_state in the **com.rti.dds.subscription.SampleInfo** (p. 1634) is **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) or **com.↩ rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161), then the last

sample for that instance in the collection (that is, the one whose **com.rti.dds.subscription.SampleInfo** (p. 1634) has sample_rank==0) does not contain valid data.

Samples that contain no data do not count towards the limits imposed by the **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy** (p. 1590). The act of reading/taking a sample sets its sample_state to com.rti.dds.subscription.↩ SampleStateKind.SampleStateKind.READ_SAMPLE_STATE.

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the instance_state of the instance.

This operation must be provided on the specialized class that is generated for the particular application data-type that is being read (**com.rti.ndds.example.Foo** (p. 1066)).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the operations fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). For an example on how `take` can be used, please refer to the **Access received data via a reader** (p. 134) "receive example".

Parameters

received_data	<<***inout***>> *(p. 156)* User data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described above.
sample_states	<<***in***>> *(p. 156)* Data samples matching one of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(p. 156)* Data samples matching one of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* Data samples matching one of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076), **com.rti.ndds.example.FooData↩ Reader.take_w_condition** (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_untyped()**.

8.123.2.3 read_w_condition()

```
void read_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.read** (p. 1069) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1510) to filter data samples based on the content.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1514) must be attached to the **com.rti.dds.↩ subscription.DataReader** (p. 450); otherwise the operation will fail with **com.rti.dds.infrastructure.RETCODE_↩ PRECONDITION_NOT_MET** (p. 1598).

In case the **com.rti.dds.subscription.ReadCondition** (p. 1514) is a plain **com.rti.dds.subscription.ReadCondition** (p. 1514) and not the specialized **com.rti.dds.subscription.QueryCondition** (p. 1510), the operation is equivalent to calling **com.rti.ndds.example.FooDataReader.read** (p. 1069) and passing as `sample_states`, `view_states` and `instance_states` the value of the corresponding attributes in the `read_condition`. Using this operation, the application can avoid repeating the same parameters specified when creating the **com.rti.dds.subscription.↩ ReadCondition** (p. 1514).

The samples are accessed with the same semantics as **com.rti.ndds.example.FooDataReader.read** (p. 1069).

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the operation will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
condition	<<***in***>> *(p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.ndds.example.FooDataReader.take (p. 1071), **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_w_condition_untyped()**.

8.123.2.4 take_w_condition()

```
void take_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) except it accesses samples via the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation.

This operation is analogous to **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) except that it accesses samples via the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1514) must be attached to the **com.rti.dds.subscription.DataReader** (p. 450); otherwise the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

The samples are accessed with the same semantics as **com.rti.ndds.example.FooDataReader.take** (p. 1071).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1510) to filter data samples based on the content.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
condition	<<***in***>> *(p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076), **com.rti.ndds.example.FooDataReader.read** (p. 1069)

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_w_condition_untyped()**.

8.123.2.5 read_next_sample()

```
void read_next_sample (
                Foo received_data,
                SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450). This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634). The implied order among the samples stored in the **com.rti.dds.subscription.DataReader** (p. 450) is the same as for the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation.

The **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) operation provides a simplified API to 'read' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **com.rti.dds.subscription.DataReader** (p. 450), the operation will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) and nothing is copied.

Note

Calling **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) from the **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) callback reads only one sample of potentially many samples in the reader queue, because **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) in a loop within the on_data_available callback until **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) returns **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use **com.rti.ndds.example.FooDataReader.read** (p. 1069) rather than **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) in order to read more than one sample at a time.)

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific **com.rti.ndds.example.Foo** (p. 1066) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL **Foo** (p. 1066). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_info	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfo** (p. 1634) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1634). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_next_sample_untyped()**.

8.123.2.6 take_next_sample()

```
void take_next_sample (
            Foo received_data,
            SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' it from the **com.rti.dds.subscription.DataReader** (p. 450) so that it is no longer accessible. This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634). This operation is analogous to the **com.rti.ndds.example.FooDataReader.read_next_sample** (p. 1078) except for the fact that the sample is removed from the **com.rti.dds.subscription.DataReader** (p. 450).

The **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation, where the input data sequences has max_len=1, the sample_states=NOT_READ, the view_states=ANY_VIEW_STATE, and the instance_states=ANY_INSTANCE_STATE.

The **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) operation provides a simplified API to 'take' samples, avoiding the need for the application to manage sequences and specify states.

If there is no unread data in the **com.rti.dds.subscription.DataReader** (p. 450), the operation will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) and nothing is copied.

Note

Calling **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) from the **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) callback retrieves only one sample of potentially many samples in the reader queue, because **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 500) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) in a loop within the on_data_available callback until **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) returns **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use the **com.rti.ndds.example.FooDataReader.take** (p. 1071) rather than **com.rti.ndds.example.FooDataReader.take_next_sample** (p. 1079) in order to take more than one sample at a time.)

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific **com.rti.ndds.example.Foo** (p. 1066) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL **Foo** (p. 1066). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_info	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfo** (p. 1634) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1634). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261),

com.rti.dds.infrastructure.RETCODE_NO_DATA (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_↩ ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_next_sample_untyped()**.

8.123.2.7 read_instance()

```
void read_instance (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read** (p. 1069), except that all samples returned belong to the single specified instance whose handle is `a_handle`.

Upon successful completion, the data collection will contain samples all belonging to the same instance. The corresponding **com.rti.dds.subscription.SampleInfo** (p. 1634) verifies **com.rti.dds.subscription.SampleInfo.instance↩ _handle** (p. 1640) == `a_handle`.

The **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation is semantically equivalent to the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation, except in building the collection, the **com.rti.dds.↩ subscription.DataReader** (p. 450) will check that the sample belongs to the specified instance and otherwise it will not place the sample in the returned collection.

The behavior of the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.↩ read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the **com.rti.dds.↩ infrastructure.InstanceHandle_t** (p. 1152) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

received_data	<<***inout***>> *(*p. *156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(*p. *156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	<<***in***>> *(*p. *156)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).
sample_states	<<***in***>> *(*p. *156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(*p. *156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(*p. *156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_instance_untyped()**.

8.123.2.8 take_instance()

```
void take_instance (
            FooSeq received_data,
```

```
 SampleInfoSeq info_seq,
int max_samples,
 InstanceHandle_t a_handle,
int sample_states,
int view_states,
int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.take** (p. 1071), except for that all samples returned belong to the single specified instance whose handle is `a_handle`.

The semantics are the same for the **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 450) will check that the sample belongs to the specified instance, and otherwise it will not place the sample in the returned collection.

The behavior of the **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

received_data	<<***inout***>> *(*p. *156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(*p. *156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	<<***in***>> *(*p. *156)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL. The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

sample_states	<<***in***>> *(p. 156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(p. 156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(p. 156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_instance_untyped()**.

8.123.2.9 read_next_instance()

```
void read_next_instance (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) where all the samples belong to a single instance. The behavior is similar to **com.rti.ndds.example.FooDataReader.read_↩ instance** (p. 1081), except that the actual instance is not directly specified. Rather, the samples will all belong to the 'next' instance with `instance_handle` 'greater' than the specified 'previous_handle' that has available samples.

This operation implies the existence of a total order 'greater-than' relationship between the instance handles. The specifics of this relationship are not all important and are implementation specific. The important thing is that, according to the middleware, all instances are ordered relative to each other. This ordering is between the instance handles; It should not depend on the state of the instance (e.g. whether it has data or not) and must be defined even for instance

handles that do not correspond to instances currently managed by the **com.rti.dds.subscription.DataReader** (p. 450). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) is 'as if' the **com.rti.dds.↩ subscription.DataReader** (p. 450) invoked **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081), passing the smallest instance_handle among all the ones that: (a) are greater than `previous_handle`, and (b) have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **com.rti.dds.↩ infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

Note

The operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **com.rti.↩ dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), examines the samples returned, and then uses the `instance_handle` returned in the **com.rti.dds.subscription.SampleInfo** (p. 1634) as the value of the `previous_handle` argument to the next call to **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084). The iteration continues until **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597). This application-driven iteration is required to ensure that all samples on the reader queue are read.

Note that it is possible to call the **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) operation with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.↩ subscription.DataReader** (p. 450). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **com.rti.dds.subscription.DataReader** (p. 450). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **com.rti.dds.subscription.↩ InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) instance, returns the loan (at which point the instance information may be removed, and thus the handle becomes invalid), and tries to read the next instance. The behavior of the **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.↩ read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	<<***in***>> *(*p. *156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_states	<<***in***>> *(*p. *156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(*p. *156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(*p. *156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_next_instance_untyped()**.

8.123.2.10 take_next_instance()

```
void take_next_instance (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450).

This operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Note

Like **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), this operation is intended to be used in an application-driven iteration until all samples on the reader queue are taken. The iteration continues until **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) fails with the value **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) .

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(*p. *156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	<<***inout***>> *(*p. *156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	<<***in***>> *(*p. *156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	<<***in***>> *(*p. *156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
sample_states	<<***in***>> *(*p. *156)* data samples matching ones of these `sample_states` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.SampleStateKind** (p. 1663).
view_states	<<***in***>> *(*p. *156)* data samples matching ones of these `view_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.ViewStateKind** (p. 1970).
instance_states	<<***in***>> *(*p. *156)* data samples matching ones of these `instance_state` are returned. See the valid values for this parameter here: **com.rti.dds.subscription.InstanceStateKind** (p. 1160).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_next_instance_untyped()**.

8.123.2.11 read_instance_w_condition()

```
void read_instance_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

<<***extension***>> *(*p. *155)* Accesses via **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read_instance** (p. 1081), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

The behavior of the **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooData↩ Reader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.Foo↩ DataReader.return_loan** (p. 1094).

Similar to **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_instance_w_condition_untyped()**.

8.123.2.12 take_instance_w_condition()

```
void take_instance_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

*<<**extension**>> (p. 155)* Accesses via **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.take_instance** (p. 1082), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to belong the single specified instance whose handle is `a_handle`, and for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

The operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_instance_w_condition** (p. 1088), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

The behavior of the **com.rti.ndds.example.FooDataReader.take_instance_w_condition** (p. 1089) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_instance_w_condition** (p. 1089) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
a_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_instance_w_condition_untyped()**.

8.123.2.13 read_next_instance_w_condition()

```
void read_next_instance_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
              ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450). The behavior is identical to **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), except that all returned samples satisfy the specified condition. In other words, on success, all returned samples belong to the same instance, and the instance is the instance with 'smallest' `instance_handle` among the ones that verify: (a) `instance_handle` >= `previous_handle`, and (b) have samples for which the specified **com.rti.dds.subscription.ReadCondition** (p. 1514) evaluates to TRUE.

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	<<***inout***>> *(p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.read_next_instance_w_condition_untyped()**.

8.123.2.14 take_next_instance_w_condition()

```
void take_next_instance_w_condition (
              FooSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t previous_handle,
              ReadCondition condition )
```

Accesses via **com.rti.ndds.example.FooDataReader.take_next_instance** (p. 1086) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 450) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 450).

The operation has the same behavior as **com.rti.ndds.example.FooDataReader.read_next_instance_w_condition** (p. 1091), except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 450) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **com.rti.ndds.example.FooDataReader.read_next_instance** (p. 1084), it is possible to call **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 450).

The behavior of the **com.rti.ndds.example.FooDataReader.take_next_instance_w_condition** (p. 1092) operation follows the same rules as the **com.rti.ndds.example.FooDataReader.read** (p. 1069) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to **com.rti.ndds.↩ example.FooDataReader.read** (p. 1069), the **com.rti.ndds.example.FooDataReader.take_next_instance_w_↩ condition** (p. 1092) operation may 'loan' elements to the output collections, which must then be returned by means of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094).

Similar to the **com.rti.ndds.example.FooDataReader.read** (p. 1069), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **com.rti.dds.subscription.DataReader** (p. 450) has no samples that meet the constraints, the method will fail with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

received_data	*<<**inout**>> (p. 156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
info_seq	*<<**inout**>> (p. 156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
max_samples	*<<**in**>> (p. 156)* The maximum number of samples to be returned. If the special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) is provided, as many samples will be returned as are available, up to the limits described in the documentation for **com.rti.ndds.example.FooDataReader.take()** (p. 1071).
previous_handle	*<<**in**>> (p. 156)* The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.
condition	*<<**in**>> (p. 156)* the **com.rti.dds.subscription.ReadCondition** (p. 1514) to select samples of interest. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

References **DataReader.take_next_instance_w_condition_untyped()**.

8.123.2.15 return_loan()

```
void return_loan (
              FooSeq received_data,
              SampleInfoSeq info_seq )
```

Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.↩ subscription.DataReader** (p. 450).

This operation indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.↩ rti.dds.subscription.DataReader** (p. 450).

The `received_data` and `info_seq` must belong to a single related "pair"; that is, they should correspond to a pair returned from a single call to read or take. The `received_data` and `info_seq` must also have been obtained from the same **com.rti.dds.subscription.DataReader** (p. 450) to which they are returned. If either of these conditions is not met, the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

The operation **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) allows implementations of the read and take operations to "loan" buffers from the **com.rti.dds.subscription.DataReader** (p. 450) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **com.rti.dds.subscription.DataReader** (p. 450) will guarantee that the data and sample-information are not modified.

It is not necessary for an application to return the loans immediately after the read or take calls. However, as these buffers correspond to internal resources inside the **com.rti.dds.subscription.DataReader** (p. 450), the application should not retain them indefinitely.

The use of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) is only necessary if the read or take calls "loaned" buffers to the application. This only occurs if the `received_data` and `info_Seq` collections had `max_↩ len=0` at the time read or take was called.

If the collections had a loan, upon completion of **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094), the collections will have `max_len=0`.

Similar to read, this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

Parameters

received_data	<<***in***>> *(*p. *156)* user data type-specific com.rti.dds.infrastructure.com.rti.dds.util.Sequence object where the received data samples was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450). Must be a valid non-NULL **FooSeq** (p. 1116). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Parameters

info_seq	<<***in***>> *(*p. *156)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1647) object where the received sample info was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450). Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1647). The method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if it is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

References **DataReader.return_loan_untyped()**.

8.123.2.16 get_key_value()

```
void get_key_value (
              Foo key_holder,
              InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

key_holder	<<***inout***>> *(*p. *156)* a user data type specific key holder, whose `key` fields are filled by this operation. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `key_holder` is NULL.
handle	<<***in***>> *(*p. *156)* the `instance` whose key is to be retrieved. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` must represent an existing instance of type **com.rti.ndds.example.Foo** (p. 1066) known to the **com.rti.dds.subscription.DataReader** (p. 450). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

If **com.rti.ndds.example.Foo** (p. 1066) has a key and `handle` represents an instance of another type or an instance of type **com.rti.ndds.example.Foo** (p. 1066) that has been unregistered, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataReader.get_key_value_untyped()**.

8.123.2.17 lookup_instance()

```
InstanceHandle_t lookup_instance (
            Foo key_holder )
```

Retrieves the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. If the instance is unknown to the DataReader, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

Parameters

key_holder	<<***in***>> *(*p. *156)* a user data type specific key holder.

Returns

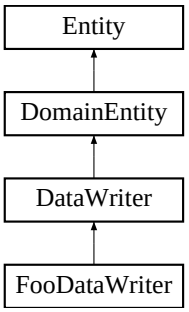
the instance handle associated with this instance. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect and returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156)

References **DataReader.lookup_instance_untyped()**.

8.124 FooDataWriter Class Reference

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* User data type specific data writer.

Inheritance diagram for FooDataWriter:



Public Member Functions

- **InstanceHandle_t register_instance** (**Foo** instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- **InstanceHandle_t register_instance_w_timestamp** (**Foo** instance_data, **Time_t** source_timestamp)

  Performs the same functions as register_instance except that the application provides the value for the `source_timestamp`*.*
- **InstanceHandle_t register_instance_w_params** (**Foo** instance_data, **WriteParams_t** params)

  *Performs the same function as **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098) and **com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp** (p. 1099) except that it also provides the values contained in* `params`*.*
- void **unregister_instance** (**Foo** instance_data, **InstanceHandle_t** handle)

  *Reverses the action of **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098).*
- void **unregister_instance_w_timestamp** (**Foo** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *Performs the same function as **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) except that it also provides the value for the* `source_timestamp`*.*
- void **unregister_instance_w_params** (**Foo** instance_data, **WriteParams_t** params)

  *Performs the same function as **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) and com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp except that it also provides the values contained in* `params`*.*
- void **write** (**Foo** instance_data, **InstanceHandle_t** handle)

  Modifies the value of a data instance.
- void **write_w_timestamp** (**Foo** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *Performs the same function as **com.rti.ndds.example.FooDataWriter.write** (p. 1105) except that it also provides the value for the* `source_timestamp`*.*
- void **write_w_params** (**Foo** instance_data, **WriteParams_t** params)

  *Performs the same function as **com.rti.ndds.example.FooDataWriter.write** (p. 1105) and **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) except that it also provides the values contained in* `params`*.*
- void **dispose** (**Foo** instance_data, **InstanceHandle_t** instance_handle)

  Requests the middleware to delete the instance.

- void **dispose_w_timestamp** (**Foo** instance_data, **InstanceHandle_t** instance_handle, **Time_t** source_↩ timestamp)

  Performs the same functions as dispose except that the application provides the value for the `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(*p. *450) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(*p. *1634).*
- void **dispose_w_params** (**Foo** instance_data, **WriteParams_t** params)

  Performs the same function as ***com.rti.ndds.example.FooDataWriter.dispose*** *(*p. *1111) and* ***com.rti.ndds.example.↩ FooDataWriter.dispose_w_timestamp*** *(*p. *1113) except that it also provides the values contained in* `params`*.*
- void **get_key_value** (**Foo** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` *that corresponds to an instance* `handle`*.*
- **InstanceHandle_t lookup_instance** (**Foo** key_holder)

  Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*

8.124.1 Detailed Description

<<***interface***>> *(*p. *156)* <<***generic***>> *(*p. *156)* User data type specific data writer.

Defines the user data type specific writer interface generated for each application class.

The concrete user data type writer automatically generated by the implementation is an incarnation of this class.

See also

com.rti.dds.publication.DataWriter (p. 553)

com.rti.ndds.example.Foo (p. 1066)

com.rti.ndds.example.FooDataReader (p. 1067)

the `Code Generator User's Manual`

8.124.2 Member Function Documentation

8.124.2.1 register_instance()

```
InstanceHandle_t register_instance (
            Foo instance_data )
```

Informs RTI Connext that the application will be modifying a particular instance.

This operation is only useful for keyed data types. Using it for non-keyed types causes no effect and returns **com.↩ rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). The operation takes as a parameter an instance (of which only the key value is examined) and returns a `handle` that can be used in successive **write()** (p. 1105) or **dispose()** (p. 1111) operations.

The operation gives RTI Connext an opportunity to pre-configure itself to improve performance.

The use of this operation by an application is optional even for keyed types. If an instance has not been pre-registered, the application can use the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) as the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) parameter to the write or dispose operation and RTI Connext will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **com.rti.ndds.example.FooDataWriter.write** (p. 1105), **com.rti.ndds.example.FooDataWriter.write_w_↩ timestamp** (p. 1109), **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) and **com.rti.ndds.example.Foo↩ DataWriter.dispose_w_timestamp** (p. 1113) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if **com.rti.↩ dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) limit has been exceeded.

The operation is **idempotent**. If it is called for an already registered instance, it just returns the already allocated handle. This may be used to lookup and retrieve the handle allocated to a given instance.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 553) has been enabled. Otherwise, **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will be returned.

Parameters

instance_data	*<<**in**>> (p. 156)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.

Returns

For keyed data type, a handle that can be used in the calls that take a **com.rti.dds.infrastructure.Instance↩ Handle_t** (p. 1152), such as write, dispose, unregister_instance, or return **com.rti.dds.infrastructure.Instance↩ Handle_t.HANDLE_NIL** (p. 1156) on failure. If the instance_data is of a data type that has no keys, this function always returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100), **com.rti.ndds.example.FooData↩ Writer.get_key_value** (p. 1114), **Relationship between registration, liveliness and ownership** (p. 1344)

References **DataWriter.register_instance_untyped()**.

8.124.2.2 register_instance_w_timestamp()

```
InstanceHandle_t register_instance_w_timestamp (
            Foo instance_data,
            Time_t source_timestamp )
```

Performs the same functions as register_instance except that the application provides the value for the `source_↩ timestamp`.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 218) QoS policy for details.

This operation may fail and return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) limit has been exceeded.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 553) has been enabled. Otherwise, **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) will be returned.

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Returns

For keyed data type, return a handle that can be used in the calls that take a **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152), such as write, dispose, unregister_instance, or return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) on failure. If the instance_data is of a data type that has no keys, this function always return **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100), **com.rti.ndds.example.FooDataWriter.get_key_value** (p. 1114)

References **DataWriter.register_instance_w_timestamp_untyped()**.

8.124.2.3 register_instance_w_params()

```
InstanceHandle_t register_instance_w_params (
            Foo instance_data,
            WriteParams_t params )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098) and **com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp** (p. 1099) except that it also provides the values contained in `params`.

See also

com.rti.ndds.example.FooDataWriter.write_w_params (p. 1110)

8.124.2.4 unregister_instance()

```
void unregister_instance (
              Foo instance_data,
              InstanceHandle_t handle )
```

Reverses the action of **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098).

This operation is useful only for keyed data types. Using it for non-keyed types causes no effect and reports no error. The operation takes as a parameter an instance (of which only the key value is examined) and a handle.

This operation should only be called on an instance that is currently registered. This includes instances that have been auto-registered by calling operations such as write or dispose as described in **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098). Otherwise, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

This only need be called just once per instance, regardless of how many times register_instance was called for that instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Connext that the **com.rti.dds.publication.DataWriter** (p. 553) is no longer going to provide any information about the instance. This operation also indicates that RTI Connext can locally remove all information regarding that instance. The application should not attempt to use the `handle` previously allocated to that instance after calling this function.

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `handle`. This indicates that the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the handle is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **unregister_instance()** (p. 1100) operation is for the instance as indicated by the `handle`.

If, after a **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100), the application wants to modify (**com.rti.ndds.example.FooDataWriter.write** (p. 1105) or **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111)) an instance, it has to register it again, or else use the special `handle` value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

This operation does not indicate that the instance is deleted (that is the purpose of **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111)). The operation **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) just indicates that the **com.rti.dds.publication.DataWriter** (p. 553) no longer has anything to say about the instance. **com.rti.dds.subscription.DataReader** (p. 450) entities that are reading the instance may receive a sample with **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) for the instance, unless there are other **com.rti.dds.publication.DataWriter** (p. 553) objects writing that same instance.

com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances (p. 2007) controls whether instances are automatically disposed when they are unregistered.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 244)). If the **com.rti.dds.↩ publication.DataWriter** (p. 553) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1100) will relinquish that ownership.

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) is set to **com.rti.dds.infrastructure.Reliability↩ QosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) and the unregistration would overflow the resource limits of this writer or of a reader, this operation may block for up to **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_↩ blocking_time** (p. 1528); if this writer is still unable to unregister after that period, this method will fail with **com.rti.↩ dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) . If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
handle	<<***in***>> *(p. 156)* represents the `instance` to be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` is not used and `instance` is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will report the error **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597)

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

Relationship between registration, liveliness and ownership (p. 1344)

References **DataWriter.unregister_instance_untyped()**.

8.124.2.5 unregister_instance_w_timestamp()

```
void unregister_instance_w_timestamp (
                Foo instance_data,
                InstanceHandle_t handle,
                Time_t source_timestamp )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) except that it also provides the value for the `source_timestamp`.

The provided `source_timestamp` potentially affects the relative order in which readers observe events from multiple writers. Refer to **DESTINATION_ORDER** (p. 218) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.

This operation may block and may time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for the unregister_instance operation.

Parameters

instance_data	<<***in***>> *(p. 156)* The instance that should be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
handle	<<***in***>> *(p. 156)* represents the `instance` to be unregistered. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` is not used and `instance` is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will report the error **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataWriter.unregister_instance_w_timestamp_untyped()**.

8.124.2.6 unregister_instance_w_params()

```
void unregister_instance_w_params (
              Foo instance_data,
              WriteParams_t params )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) and com.rti.↩ ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp except that it also provides the values contained in `params`.

See also

com.rti.ndds.example.FooDataWriter.write_w_params (p. 1110)

com.rti.ndds.example.FooDataWriter.dispose_w_params (p. 1114)

8.124.2.7 write()

```
void write (
              Foo instance_data,
              InstanceHandle_t handle )
```

Modifies the value of a data instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1634) and **DESTINATION_ORDER** (p. 218) QoS policy for details).

As a side effect, this operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 553) itself, the **com.↩ rti.dds.publication.Publisher** (p. 1466) and the **com.rti.dds.domain.DomainParticipant** (p. 670).

Note that the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `handle`. This indicates the identity of the instance should be automatically deduced from the `instance↩ _data` (by means of the `key`).

If `handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the handle is any value other than **com.rti.dds.infrastructure.Instance↩ Handle_t.HANDLE_NIL** (p. 1156), corresponds to an instance that has been registered, but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). RTI Connext will treat as if the **write()** (p. 1105) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 258) `kind` is set to **com.rti.dds.infrastructure.ReliabilityQos↩ PolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 259) to be exceeded.

This operation will not block when using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532). If you are using BEST_EFFORT Reliability in combination with com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS, then instead of being blocked, samples that are queued to be sent by the asynchronous publishing thread will be overwritten when the number of DDS samples that are currently queued has reached the `depth` QoS value in the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144).

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) elapses before the **com.rti.dds.publication.DataWriter** (p. 553) can store the modification without exceeding the limits, the operation will fail and return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) for KEEP_ALL configurations.

Here is how the write operation behaves when com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) are used:

- The send window size is determined by the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1616) fields in the **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires.
- Then, the **com.rti.dds.publication.DataWriter** (p. 553) will try to add the new sample to the writer history.
- If the instance associated with the sample is present in the writer history and there are `depth` (in the **HISTORY** (p. 237)) samples in the instance, the DataWriter will replace the oldest sample of that instance independently of that sample's acknowledged status, and the write operation will return com.rti.dds.infrastructure.RETCODE_OK. Otherwise, no sample will be replaced and the write operation will continue.
- If the instance associated with the sample is not present in the writer history and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629) (see **com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind** (p. 623)).
  - If no instance can be replaced, the write operation returns **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).
- If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) is exceeded, the DataWriter will try to drop a sample from a different instance as follows:
  - The DataWriter will try first to remove a fully ACKed (2) sample from a different instance 'I' as long as that sample is not the last remaining sample for the instance 'I'. To find this sample, the DataWriter starts iterating from the oldest sample in the writer history to the newest sample.
  - If no such sample is found, the DataWriter will replace the oldest sample in the writer history.
- The sample is added to the writer history, and the write operation returns com.rti.dds.infrastructure.RETCODE_OK.

Here is how the write operation behaves when com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS and **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) are used:

- The send window size is determined by the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1616) fields in the **DATA_WRITER_PROTOCOL** (p. 215). If a send window is specified (max_send_window_size is not UNLIMITED) and the window is full, the write operation will block until one of the samples in the send window is protocol-acknowledged (ACKed) (1) or until the **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires.
  - If the max_blocking_time expires, the write operation returns **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
- When a sample is protocol-ACKed (1) before max_blocking_time expires, the DataWriter will try to add the sample to the writer history as follows:
  - If the instance associated with the sample is not present in the writer history and max_instances is exceeded, the DataWriter will try to replace an existing instance (and its samples) according to the value of **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement** (p. 629) (see **com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind** (p. 623)).
    - If no instance can be replaced, the write operation returns **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).
  - If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) is exceeded, the DataWriter will go through the samples in the order in which they were added, and it will replace the first sample that is fully ACKed (2).
    - If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528) expires. If the max_blocking_time expires, the write operation will return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
  - If **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) is exceeded, the DataWriter will go through the samples of the instance in the order in which they were added, and it will replace the first sample that is fully ACKed.
    - If no fully ACKed sample is found, the DataWriter will block (3) until a sample is fully ACKed and can be replaced or the max_blocking_time expires. If the max_blocking_time expires, the write operation will return **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).
  - The sample is added to the writer history, and the write operation returns com.rti.dds.infrastructure.RETCODE_OK.

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help freeing up some resources.

This operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if the timestamp is less than the timestamp used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp).

See **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind** (p. 1530) for more information on the following notes:

(1) A sample in the writer history is considered "protocol ACKed" when the sample has been individually ACKed at the RTPS protocol level by each one of the DataReaders that matched the DataWriter at the moment the sample was added to the writer queue.

- Late joiners do not change the protocol ACK state of a sample. If a sample is marked as protocol ACKed because it has been acknowledged by all the matching DataReaders and a DataReader joins later on, the historical sample is still considered protocol ACKed even if it has not been received by the late joiner.

- If a sample 'S1' is protocol ACKed and a TopicQuery is received, triggering the publication of 'S1', the sample is still considered protocol ACKed. If a sample 'S1' is not ACKed and a TopicQuery is received triggering the publication of 'S1', the DataWriter will require that both the matching DataReaders on the live RTPS channel and the DataReader on the TopicQuery channel individually protocol ACK the sample in order to consider the sample protocol ACKed.

(2) A sample in the writer history is considered "fully ACKed" when all of the following conditions are met:

- The sample is protocol-ACKed.
- The sample has been "application-level ACKed" by all the DataReaders matching the DataWriter that have their **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) set to **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 1531) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgment↩ ModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530). Once the sample is application-level ACKed, it cannot change its status to not ACKed after new DataReaders are matched. (Application-level ACK occurs when the application acknowledges receipt of a sample.)
- If required subscriptions are enabled (see **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 343)), the sample must also be ACKed by all the required subscriptions configured on the DataWriter.

(3) It is possible within a single call to the write operation for a DataWriter to block both when the send window is full and then again when **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) or **com.rti.↩ dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) is exceeded. This can happen because blocking on the send window only considers protocol-ACKed samples, while blocking based on resource limits considers fully-ACKed samples. In any case, the total max blocking time of a single call to the write operation will not exceed **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1528).

Parameters

instance_data	<<***in***>> *(*p. *156)* The data to write.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_data` is NULL.

Parameters

handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

MT Safety:

It is UNSAFE to modify `instance_data` before the operation has finished. The operation is otherwise SAFE.

See also

com.rti.dds.subscription.DataReader (p. 450)

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

DESTINATION_ORDER (p. 218)

References **DataWriter.write_untyped()**.

8.124.2.8 write_w_timestamp()

```
void write_w_timestamp (
                Foo instance_data,
                InstanceHandle_t handle,
                Time_t source_timestamp )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.write** (p. 1105) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1634) and **DESTINATION_ORDER** (p. 218) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.write** (p. 1105) operation.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_↩ RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help free up some resources.

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) under the same circumstances described for the write operation.

Parameters

instance_data	<<***in***>> *(p. 156)* The data to write. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_data` is NULL.
handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.
source_timestamp	<<***in***>> *(p. 156)* When using com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.↩ DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the timestamp of the previous operation but the difference is less than the **com.rti.dds.infrastructure.DestinationOrderQosPolicy.source_timestamp_tolerance** (p. 637), the timestamp of the previous operation will be used as the source timestamp of this sample. Otherwise, if the difference is greater than **com.rti.dds.infrastructure.DestinationOrderQosPolicy.source_timestamp_tolerance** (p. 637), the function will return **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

com.rti.dds.subscription.DataReader (p. 450)

DESTINATION_ORDER (p. 218)

References **DataWriter.write_w_timestamp_untyped()**.

8.124.2.9 write_w_params()

```
void write_w_params (
            Foo instance_data,
            WriteParams_t params )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.write** (p. 1105) and **com.rti.ndds.example.↩ FooDataWriter.write_w_timestamp** (p. 1109) except that it also provides the values contained in `params`.

Allows provision of the sample identity, related sample identity, source timestamp, instance handle, and publication priority contained in `params`.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumstances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF↩ _RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance_w_params** (p. 1104) may help free up some resources.

This operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) under the same circumstances described for the **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

Parameters

instance_data	<<***in***>> *(*p. *156)* The data to write. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_data` is NULL.
params	<<***inout***>> *(*p. *156)* The write parameters. Note that this is an inout parameter if you activate com.rti.dds.infrastructure.WriteParams_t.replace_auto; otherwise it won't be modified. This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `params` is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

com.rti.dds.subscription.DataReader (p. 450)

8.124.2.10 dispose()

```
void dispose (
                Foo instance_data,
                InstanceHandle_t instance_handle )
```

Requests the middleware to delete the instance.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

When an instance is disposed, the **com.rti.dds.publication.DataWriter** (p. 553) communicates this state change to **com.rti.dds.subscription.DataReader** (p. 450) objects by propagating a dispose sample. When the instance changes to a disposed state, you can see the state change on the DataReader by looking at **com.rti.dds.subscription.SampleInfo.instance_state** (p. 1640). Disposed instances have the value **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).

The resources allocated to dispose instances on the DataWriter are not removed by default. The removal of the resources allocated to a dispose instance on the DataWriter queue can be controlled by using the QoS **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autopurge_disposed_instances_delay** (p. 2008).

Likewise, on the DataReader, the removal of the resources associated with an instance in the dispose state can be controlled by using the QoS **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy.autopurge_disposed_instances_delay** (p. 1522).

This operation does not modify the value of the instance. The `instance_data` parameter is passed just for the purposes of identifying the instance.

When this operation is used, RTI Connext will automatically supply the value of the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the source_timestamp attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) operation.

The special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) can be used for the parameter `instance_handle`. This indicates the identity of the instance should be automatically deduced from the `instance_data` (by means of the `key`).

If `instance_handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

RTI Connext will not detect the error when the `instance_handle` is any value other than **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), and the `instance_handle` corresponds to an instance that has been registered but does not correspond to the instance deduced from the `instance_data` (by means of the `key`). In this case, the instance that will be disposed is the instance corresponding to the `instance_handle`, not to the `instance_data`.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumstances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help free up some resources.

Parameters

instance_data	*<<**in**>> (p. 156)* The data to dispose. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Parameters

instance_handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `instance_handle` is not used and it is deduced from `instance_data`. If **com.rti.ndds.example.Foo** (p. 1066) has no key, `instance_handle` is not used. If `instance_handle` is used, it must represent an instance of type **com.rti.ndds.example.Foo** (p. 1066) that has been written or registered with this writer. Otherwise, this method fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `instance_handle` is NULL. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_handle` cannot be **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) if `instance_data` is NULL. Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

Relationship between registration, liveliness and ownership (p. 1344)

References **DataWriter.dispose_untyped()**.

8.124.2.11 dispose_w_timestamp()

```
void dispose_w_timestamp (
              Foo instance_data,
              InstanceHandle_t instance_handle,
              Time_t source_timestamp )
```

Performs the same functions as dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) operation.

This operation may block and time out (**com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)) under the same circumtances described for **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

If there are no instance resources left, this operation may fail with **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598). Calling **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) may help freeing up some resources.

Parameters

instance_data	<<***in***>> *(p. 156)* The data to dispose. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `instance_handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).
instance_handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is not **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **DataWriter.dispose_w_timestamp_untyped()**.

8.124.2.12 dispose_w_params()

```
void dispose_w_params (
            Foo instance_data,
            WriteParams_t params )
```

Performs the same function as **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) and **com.rti.ndds.↩ example.FooDataWriter.dispose_w_timestamp** (p. 1113) except that it also provides the values contained in `params`.

See also

com.rti.ndds.example.FooDataWriter.write_w_params (p. 1110)

8.124.2.13 get_key_value()

```
void get_key_value (
                Foo key_holder,
                InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

Useful for keyed data types.

The operation will only fill the fields that form the `key` inside the `key_holder` instance. If the type has no keys, this method has no effect and exits with no error.

For keyed data types, this operation may fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

key_holder	<<***inout***>> *(p. 156)* a user data type specific key holder, whose `key` fields are filled by this operation. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect.

This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `key_holder` is NULL.

Parameters

handle	<<***in***>> *(p. 156)* the `instance` whose key is to be retrieved. If **com.rti.ndds.example.Foo** (p. 1066) *has* a key, `handle` must represent a registered instance of type **com.rti.ndds.example.Foo** (p. 1066). Otherwise, this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). If **com.rti.ndds.example.Foo** (p. 1066) *has* a key and `handle` is **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156), this method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594). This method will fail with **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) if `handle` is NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

References **DataWriter.get_key_value_untyped()**.

8.124.2.14 lookup_instance()

```
InstanceHandle_t lookup_instance (
            Foo key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

Useful for keyed data types.

This operation takes as a parameter an instance and returns a handle that can be used in subsequent operations that accept an instance handle as an argument. The instance parameter is only used for the purpose of examining the fields that define the key. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason RTI Connext is unable to provide an instance handle, RTI Connext will return the special value HANDLE_NIL.

Parameters

key_holder	*<<**in**>> (p. 156)* a user data type specific key holder.

Returns

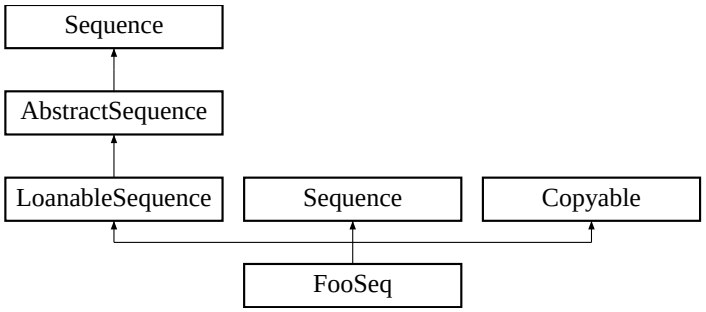
the instance handle associated with this instance. If **com.rti.ndds.example.Foo** (p. 1066) has no key, this method has no effect and returns **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156)

References **DataWriter.lookup_instance_untyped()**.

8.125 FooSeq Class Reference

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. 1066).

Inheritance diagram for FooSeq:



Public Member Functions

- Object **copy_from** (Object src)

8.125.1 Detailed Description

<<***interface***>> *(p. 156)* <<***generic***>> *(p. 156)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **com.rti.ndds.example.Foo** (p. 1066).

For users who define data types in OMG IDL, this type corresponds to the IDL express sequence<**Foo** (p. 1066)>.

For any user-data type **Foo** (p. 1066) that an application defines for the purpose of data-distribution with RTI Connext, a **FooSeq** (p. 1116) is generated. We refer to an IDL sequence<**Foo** (p. 1066)> as **FooSeq** (p. 1116).

A sequence is a type-safe List that makes a distinction between its allocated size and its logical size (much like the ArrayList class). The Collection.size() method returns the logical size.

A new sequence is created for elements of a particular Class, which does not change throughout the lifetime of a sequence instance.

To add an element to a sequence, use the **add()** (p. 329) method inherited from the standard interface java.↩ util.List; this will implicitly increase the sequence's size. Or, to pre-allocate space for several elements at once, use com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

An attempt to add an element to a sequence that is not of the correct element type will result in a ClassCastException. (Note that null is considered to belong to any type.)

See also

com.rti.ndds.example.FooDataWriter (p. 1097), **com.rti.ndds.example.FooDataReader** (p. 1067), **com.rti.↩ ndds.example.FooTypeSupport** (p. 1118), the Code Generator User's Manual

8.125.2 Member Function Documentation

8.125.2.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy data into this object from another. The result of this method is that both this and src will be the same size and contain the same data.

Parameters

src	The Object which contains the data to be copied

Returns

`this`

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not a `Sequence` OR if one of the objects contained in the `Sequence` is not of the expected type.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

References **AbstractSequence.add()**, **FooSeq.copy_from()**, **LoanableSequence.getMaximum()**, **Loanable↩ Sequence.setMaximum()**, and **LoanableSequence.size()**.

Referenced by **FooSeq.copy_from()**.

8.126 FooTypeSupport Class Reference

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* User data type specific interface.

Inherits TypeSupportImpl.

Public Member Functions

- Object **copy_data** (Object destination, Object source)
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **Foo** src)

  *<<**extension**>> (p. 155) Serializes the input sample into a CDR buffer of octets.*
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **Foo** src, short representation)

  *<<**extension**>> (p. 155) Serializes the input sample into a buffer of octets.*
- void **deserialize_from_cdr_buffer** (**Foo** dst, byte[ ] buffer, long length)

  *<<**extension**>> (p. 155) Deserializes a sample from a buffer of octets.*
- String **data_to_string** (**Foo** src, **PrintFormatProperty** formatProperty)

  *<<**extension**>> (p. 155) Transforms a data sample into a human-readable string representation.*
- String **data_to_string** (**Foo** src)

  *<<**extension**>> (p. 155) Transforms a data sample into a human-readable string representation using the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387).*
- **Foo data_from_string** (String str, **PrintFormatKind** printFormat)

  *<<**extension**>> (p. 155) Creates a new sample from a string.*
- **Foo data_from_string** (String str)

  *<<**extension**>> (p. 155) Creates a new **Foo** (p. 1066) sample from a JSON string.*
- **DynamicData data_to_dynamicdata** (Object data)

  *<<**extension**>> (p. 155) Creates a new `DynamicData` sample from an existing **Foo** (p. 1066) sample*
- **Foo data_from_dynamicdata** (**DynamicData** dynamicData)

  *<<**extension**>> (p. 155) Creates a new **Foo** (p. 1066) sample from an existing `DynamicData` sample.*

Static Public Member Functions

- static String **get_type_name** ()

  Get the default name for this type.
- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of a data type.
- static **TypeCode** **getTypeCode** ()

  *<<**extension**>> (p. 155) Retrieves the TypeCode for the Type.*

8.126.1 Detailed Description

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* User data type specific interface.

Defines the user data type specific interface generated for each application class.

The concrete user data type automatically generated by the implementation is an incarnation of this class.

See also

the `Code Generator User's Manual`

8.126.2 Member Function Documentation

8.126.2.1 get_type_name()

```
static String get_type_name ( )  [static]
```

Get the default name for this type.

Can be used for calling **com.rti.ndds.example.FooTypeSupport.register_type** (p. 1119) or creating **com.rti.dds.topic.Topic** (p. 1807)

Returns

default name for this type

See also

com.rti.ndds.example.FooTypeSupport.register_type (p. 1119)

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.126.2.2 register_type()

```
static void register_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to communicate to RTI Connext the existence of a data type.

The *generated* implementation of the operation embeds all the knowledge that has to be communicated to the middleware in order to make it able to manage the contents of data of that type. This includes in particular the key definition that will allow RTI Connext to distinguish different instances of the same type.

The same **com.rti.dds.topic.TypeSupport** (p. 1941) can be registered multiple times with a **com.rti.dds.domain.↩ DomainParticipant** (p. 670) using the same or different values for the `type_name`. If `register_type` is called multiple times on the same **com.rti.dds.topic.TypeSupport** (p. 1941) with the same **com.rti.dds.domain.Domain↩ Participant** (p. 670) and `type_name`, the second (and subsequent) registrations are ignored but the operation returns com.rti.dds.infrastructure.RETCODE_OK.

Precondition

Cannot use the same `type_name` to register two different **com.rti.dds.topic.TypeSupport** (p. 1941) with the same **com.rti.dds.domain.DomainParticipant** (p. 670), or else the operation will fail and **com.rti.dds.↩ infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) will be returned.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to register the data type **Foo** (p. 1066) with. Cannot be NULL.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type **Foo** (p. 1066) is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1807). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706).) The name may not be NULL or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.126.2.3 getTypeCode()

```
static  TypeCode getTypeCode ( )  [static]
```

<<***extension***>> *(*p. *155)* Retrieves the TypeCode for the Type.

This method retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) for the Type. A **com.rti.dds.typecode.TypeCode** (p. 1873) is a mechanism for representing a type at runtime. RTI Connext can use type codes to send type definitions on the network. A **com.rti.dds.typecode.TypeCode** (p. 1873) value consists of a type code *kind* (represented by the **com.rti.dds.typecode.TCKind** (p. 1786) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **com.rti.dds.typecode.TypeCode** (p. 1873), and in the case of complex types (structures, arrays, and so on), these contained type codes contain their own members.

Returns

The TypeCode for this type

8.126.2.4 copy_data()

```
Object copy_data (
             Object destination,
             Object source )
```

This is a concrete implementation of this method inherited from the base class. This method will perform a deep copy of `source` into `destination`.

Parameters

source	The Object which contains the data to be copied.
destination	The object where data will be copied to.

Returns

Returns `destination`.

Exceptions

NullPointerException	If `destination` or `source` is null.
ClassCastException	If either `destination` or `this` is not a **`Foo`** (p. `1066`) type.

References **Foo.copy_from()**.

8.126.2.5 serialize_to_cdr_buffer() [1/2]

```
long serialize_to_cdr_buffer (
            byte[] buffer,
            long length,
             Foo src )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a CDR buffer of octets.

This method serializes a sample into a buffer of octets using Common Data Representation (CDR). Calling this method is equivalent to calling **com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer** (p. 1121) with **com.rti.dds.infrastructure.DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION** (p. 214) as the representation.

The input buffer must be big enough to store the serialized representation of the sample. Otherwise, the method will return an error.

To determine the minimum size of the input buffer, the user must call this method with the buffer set to null.

For more information about CDR in Connext, see `Data Representation` in the Core Libraries Extensible Types Guide.

Parameters

src	<<***in***>> *(p. 156)*. Input sample. Cannot be null.
buffer	<<***out***>> *(p. 156)*. Serialization buffer.
length	<<***in***>> *(p. 156)*. When `buffer` is not null, `length` must contain the size of the input buffer when the method is invoked. This parameter is ignored when `buffer` is set to null.

Returns

When `buffer` is set to null, this method will return a buffer size big enough to hold the serialized data. Notice that the size is conservative and it may be greater than the actual size after serialization. When `buffer` is different than null, this method will return the actual size of the serialized content. This is the size that must be provided to the **com.rti.ndds.example.FooTypeSupport.deserialize_from_cdr_buffer** (p. 1123).

Exceptions

One	of the **Standard Return Codes** (p. 261) , java.lang.IllegalStateException

8.126.2.6 serialize_to_cdr_buffer() [2/2]

```
long serialize_to_cdr_buffer (
            byte[] buffer,
            long length,
             Foo src,
            short representation )
```

<<***extension***>> *(*p. *155)* Serializes the input sample into a buffer of octets.

This method serializes a sample into a buffer of octets using the input data representation. See **com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer** (p. 1121) for details.

Parameters

src	<<***in***>> *(*p. *156)*. Input sample. Cannot be null.
buffer	<<***out***>> *(*p. *156)*. Serialization buffer.
length	<<***in***>> *(*p. *156)*. Serialization buffer length.
representation	<<***in***>> *(*p. *156)*. Representation used to serialize the data.

Returns

Size of the serialized content.

Exceptions

One	of the **Standard Return Codes** (p. 261) , java.lang.IllegalStateException

8.126.2.7 deserialize_from_cdr_buffer()

```
void deserialize_from_cdr_buffer (
              Foo dst,
             byte[] buffer,
             long length )
```

<<***extension***>> *(*p. *155)* Deserializes a sample from a buffer of octets.

This method deserializes a sample from a CDR buffer of octets.

The content of the buffer generated by the method **com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer** (p. 1121) can be provided to this method to get the sample back.

Parameters

dst	<<***out***>> *(*p. *156)*. Output sample. Cannot be null.
buffer	<<***in***>> *(*p. *156)*. Deserialization buffer. Cannot be null.
length	<<***in***>> *(*p. *156)*. Length of the serialized representation of the sample in the buffer.

Exceptions

One	of the **Standard Return Codes** (p. 261) , java.lang.IllegalStateException

8.126.2.8 data_to_string() [1/2]

```
String data_to_string (
            Foo src,
            PrintFormatProperty formatProperty )
```

<<***extension***>> *(*p. *155)* Transforms a data sample into a human-readable string representation.

This method takes a data sample and creates a string representation of the data.

If the size of the output string is longer than the size of an unsigned 32-bit integer, this operation will fail with **com.rti.↩ dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

This method is only provided for types that were generated with typecodes.

Parameters

src	<<***out***>> *(*p. *156)*. The sample to get the string representation of. Cannot be null.
formatProperty	<<***in***>> *(*p. *156)*. Properties describing what the format of the output string should be.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598) , java.lang.IllegalStateException

8.126.2.9 data_to_string() [2/2]

```
String data_to_string (
            Foo src )
```

<<***extension***>> *(*p. *155)* Transforms a data sample into a human-readable string representation using the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387).

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

8.126.2.10 data_from_string() [1/2]

```
Foo data_from_string (
            String str,
             PrintFormatKind printFormat )
```

<<***extension***>> *(*p. *155)* Creates a new sample from a string.

This method takes a string representation of the data and creates a new data sample.

Parameters

str	<<***in***>> *(p. 156)*. The JSON string representation
printFormat	<<***in***>> *(p. 156)*. Must be set to **com.rti.dds.topic.PrintFormatKind.JSON_PRINT_FORMAT** (p. 1386), the only format currently supported.

8.126.2.11 data_from_string() [2/2]

```
Foo data_from_string (
            String str )
```

<<***extension***>> *(p. 155)* Creates a new **Foo** (p. 1066) sample from a JSON string.

This method is an overload that does not take a PrintFormatKind.

Parameters

str	<<***in***>> *(p. 156)*. The JSON string representation

See also

com.rti.ndds.example.FooTypeSupport.data_from_string (p. 1124)

8.126.2.12 data_to_dynamicdata()

```
DynamicData data_to_dynamicdata (
            Object data )
```

<<***extension***>> *(p. 155)* Creates a new DynamicData sample from an existing **Foo** (p. 1066) sample

This method takes a **Foo** (p. 1066) sample and creates a new DynamicData sample from it. The resulting Dynamic↩ Data sample will have the same type and contain the same values as the **Foo** (p. 1066) sample.

Parameters

data	<<***in***>> *(p. 156)*. The **Foo** (p. 1066) sample that will be transformed into a DynamicData sample.

See also

com.rti.ndds.example.FooTypeSupport.data_from_dynamicdata (p. 1126).

8.126.2.13 data_from_dynamicdata()

```
Foo data_from_dynamicdata (
            DynamicData dynamicData )
```

<<***extension***>> *(p. 155)* Creates a new **Foo** (p. 1066) sample from an existing DynamicData sample.

This method takes a DynamicData sample and uses it to create a new **Foo** (p. 1066) sample.

A precondition for this operation is that the type of the DynamicData need to be assignable to **Foo** (p. 1066). If this not the case, the behavior is undefined.

Parameters

dynamicData	<<***in***>> *(p. 156)*. The DynamicData sample to be transformed into a **Foo** (p. 1066) sample.

See also

com.rti.ndds.example.FooTypeSupport.data_to_dynamicdata (p. 1126)

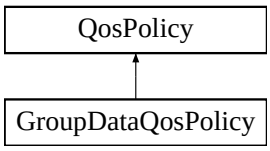
com.rti.ndds.example.FooTypeSupport.getTypeCode (p. 1120)

com.rti.dds.typecode.TypeCode.TypeCode.assignable

8.127 GroupDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Inheritance diagram for GroupDataQosPolicy:



Public Attributes

- final **ByteSeq value**

  a sequence of octets

8.127.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Entity:

com.rti.dds.publication.Publisher (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **YES** (p. 256)

See also

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.127.2 Usage

The additional information is attached to a **com.rti.dds.publication.Publisher** (p. 1466) or **com.rti.dds.↩ subscription.Subscriber** (p. 1730). This extra data is not used by RTI Connext itself. When a remote application discovers the **com.rti.dds.publication.Publisher** (p. 1466) or **com.rti.dds.subscription.Subscriber** (p. 1730), it can access that information and use it for its own purposes.

Use cases for this QoS policy, as well as the **com.rti.dds.infrastructure.TopicDataQosPolicy** (p. 1819) and **com.↩ rti.dds.infrastructure.UserDataQosPolicy** (p. 1959), are often application-to-application identification, authentication, authorization, and encryption purposes. For example, applications can use Group or User Data to send security certificates to each other for RSA-type security.

In combination with **com.rti.dds.subscription.DataReaderListener** (p. 497), **com.rti.dds.publication.DataWriter↩ Listener** (p. 589) and operations such as **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 725) and **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 726), this QoS policy can help an application to define and enforce its own security policies. For example, an application can implement matching policies similar to those of the **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365), except the decision can be made based on an application-defined policy.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.publisher_group_data↩ _max_length** (p. 815) and **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.subscriber_↩ group_data_max_length** (p. 815).

8.127.3 Member Data Documentation

8.127.3.1 value

```
final  ByteSeq value
```

a sequence of octets

[default] Empty (zero-sized)

[range] Octet sequence of length [0,max_length]

8.128 GuardCondition Class Reference

<<***interface***>> *(*p. *156)* A specific **com.rti.dds.infrastructure.Condition** (p. 429) whose `trigger_value` is completely under the control of the application.

Inherits AbstractNativeObject, AutoCloseable, and NativeCondition.

Public Member Functions

- **GuardCondition** ()
  No argument constructor.
- void **set_trigger_value** (boolean value)
  Set the guard condition trigger value.
- boolean **get_trigger_value** ()
  Retrieve the `trigger_value`*.*
- void **delete** ()
  Destructor.
- void **close** ()
  See ***delete()*** *(*p. *1131).*

8.128.1 Detailed Description

<<***interface***>> *(*p. *156)* A specific **com.rti.dds.infrastructure.Condition** (p. 429) whose `trigger_value` is completely under the control of the application.

The **com.rti.dds.infrastructure.GuardCondition** (p. 1129) provides a way for an application to manually wake up a **com.rti.dds.infrastructure.WaitSet** (p. 1973). This is accomplished by attaching the **com.rti.dds.infrastructure.↩ GuardCondition** (p. 1129) to the **com.rti.dds.infrastructure.WaitSet** (p. 1973) and then setting the `trigger_value` by means of the **com.rti.dds.infrastructure.GuardCondition.set_trigger_value** (p. 1130) operation.

Important: **GuardCondition** (p. 1129) allocates native resources. When a **GuardCondition** (p. 1129) is no longer being used, **close()** (p. 1131) must be called.

See also

com.rti.dds.infrastructure.WaitSet (p. 1973)

8.128.2 Constructor & Destructor Documentation

8.128.2.1 GuardCondition()

```
GuardCondition ( )
```

No argument constructor.

Construct a new guard condition with trigger value com.rti.dds.infrastructure.false.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Exceptions

com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES *(p. 1598)*	if a new **com.rti.dds.infrastructure.GuardCondition** (p. 1129) could not be allocated.

Important: The **com.rti.dds.infrastructure.GuardCondition** (p. 1129) allocates native resources. When **com.rti.dds.infrastructure.GuardCondition** (p. 1129) is no longer being used, user should call **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1131) explicitly to properly cleanup all native resources.

8.128.3 Member Function Documentation

8.128.3.1 set_trigger_value()

```
void set_trigger_value (
            boolean value )
```

Set the guard condition trigger value.

Parameters

value	<<***in***>> *(p. 156)* the new trigger value.

8.128.3.2 get_trigger_value()

```
boolean get_trigger_value ( )
```

Retrieve the `trigger_value`.

Returns

the trigger value.

Implements **Condition** (p. 430).

8.128.3.3 delete()

```
void delete ( )
```

Destructor.

Releases the resources associated with this object.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Calling this method multiple times on the same object is safe; subsequent deletions will have no effect.

Referenced by **GuardCondition.close()**.

8.128.3.4 close()

```
void close ( )
```

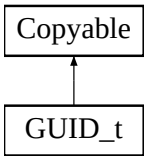
See **delete()** (p. 1131).

References **GuardCondition.delete()**.

8.129 GUID_t Class Reference

Type for *GUID* (Global Unique Identifier) representation.

Inheritance diagram for GUID_t:



Public Member Functions

- **GUID_t** (**GUID_t** guid)
  Copy constructor.
- **GUID_t** (byte[ ] **value**)
  Constructor.
- Object **copy_from** (Object src)
  Copy value of a data type from source.

Public Attributes

- byte[ ] **value** = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
  A 16 byte array containing the GUID value.

Static Public Attributes

- static final **GUID_t GUID_UNKNOWN**
  Unknown GUID.
- static final **GUID_t GUID_AUTO**
  Indicates that RTI Connext should choose an appropriate virtual GUID.
- static final **GUID_t GUID_ZERO**
  Zero GUID.

8.129.1 Detailed Description

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

8.129.2 Constructor & Destructor Documentation

8.129.2.1 GUID_t() [1/2]

```
GUID_t (
            GUID_t guid )
```

Copy constructor.

Parameters

guid	The GUID instance to copy. It must not be null.

References **GUID_t.value**.

8.129.2.2 GUID_t() [2/2]

```
GUID_t (
            byte[] value )
```

Constructor.

Parameters

value	GUID value as a 16 byte array. It must not be null.

References **GUID_t.value**.

8.129.3 Member Function Documentation

8.129.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

8.129.4 Member Data Documentation

8.129.4.1 GUID_UNKNOWN

```
final GUID_t GUID_UNKNOWN [static]
```

Unknown GUID.

8.129.4.2 GUID_AUTO

```
final GUID_t GUID_AUTO [static]
```

Initial value:

```
=
        new GUID_t(new byte[]{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0})
```

Indicates that RTI Connext should choose an appropriate virtual GUID.

If this special value is assigned to **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid** (p. 598) or **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 502), RTI Connext will assign the virtual GUID automatically based on the RTPS or physical GUID.

8.129.4.3 GUID_ZERO

```
final GUID_t GUID_ZERO [static]
```

Zero GUID.

8.129.4.4 value

```
byte [] value = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

A 16 byte array containing the GUID value.

Referenced by **BuiltinTopicKey_t.from_guid()**, **GUID_t.GUID_t()**, and **BuiltinTopicKey_t.to_guid()**.

8.130 HeapMonitoring Class Reference

Heap Monitoring APIs.

Static Public Member Functions

- static native boolean **enable** ()

  Starts monitoring the heap memory used by RTI Connext.
- static boolean **enable** (**HeapMonitoringParams** params)

  Starts monitoring the heap memory used by RTI Connext.
- static native void **disable** ()

  Stops monitoring the heap memory used by RTI Connext.
- static native boolean **pause** ()

  Pauses heap monitoring.
- static native boolean **resume** ()

  Resumes heap monitoring.
- static native boolean **take_heap_snapshot** (String filename, boolean print_details)

  Saves the current heap memory usage in a file.

8.130.1 Detailed Description

Heap Monitoring APIs.

8.130.2 Member Function Documentation

8.130.2.1 enable() [1/2]

```
static native boolean enable ( )  [static]
```

Starts monitoring the heap memory used by RTI Connext.

This function must be called before any other function in the RTI Connext library is called.

Once heap monitoring is enabled, you can take heap snapshots by using **com.rti.ndds.utility.HeapMonitoring.take_heap_snapshot** (p. 1137).

Use this method only for debugging purposes, since it may introduce a significant performance impact.

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another heap-related method, including this one.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.HeapMonitoring.disable (p. 1136)

8.130.2.2 enable() [2/2]

```
static boolean enable (
              HeapMonitoringParams params )  [static]
```

Starts monitoring the heap memory used by RTI Connext.

Perfoms the same function as **com.rti.ndds.utility.HeapMonitoring.enable** (p. 1135) except that it also provides the values in params. Those values will set the format used in the snapshot **com.rti.ndds.utility.HeapMonitoring.take_heap_snapshot** (p. 1137).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.HeapMonitoring.disable (p. 1136)

8.130.2.3 disable()

```
static native void disable ( )  [static]
```

Stops monitoring the heap memory used by RTI Connext.

This method must be the last method called from RTI Connext.

See also

com.rti.ndds.utility.HeapMonitoring.enable (p. 1135)

8.130.2.4 pause()

```
static native boolean pause ( )  [static]
```

Pauses heap monitoring.

New memory allocations will not be monitored and they will not appear in the snapshot generated by **com.rti.ndds.↩ utility.HeapMonitoring.take_heap_snapshot** (p. 1137).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.HeapMonitoring.resume (p. 1137)

8.130.2.5 resume()

```
static native boolean resume ( )  [static]
```

Resumes heap monitoring.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.HeapMonitoring.pause (p. 1137)

8.130.2.6 take_heap_snapshot()

```
static native boolean take_heap_snapshot (
            String filename,
            boolean print_details )  [static]
```

Saves the current heap memory usage in a file.

After **com.rti.ndds.utility.HeapMonitoring.enable** (p. 1135) is called, you may invoke this method periodically to save the current heap memory usage to a file.

By comparing two snapshots, you can tell if new memory has been allocated and in many cases where. This is why this operation can be used to debug unexpected memory growth.

The format of a snapshot is as follows:

First, there is a memory usage summary like this:

```
<P>
                    Product Version: NDDSCORE_BUILD_6.0.0.0_20200316T123411Z_RTI_ENG

                    Process virtual memory: 2552352768

                    Process physical memory: 16187392

                    Current application heap usage: 10532131

                    Approximate total heap usage: 203331110

                    High watermark: 10532131

                    Alloc count: 17634

                    Free count: 3518
```

- Process virtual memory: The amount of virtual memory in bytes taken by the process. This memory includes RTI Connext and non-RTI Connext memory.
- Process virtual memory: The amount of physical memory in bytes taken by the process.
- Current application heap usage: The amount of heap memory in bytes used by the middleware. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap. This value does not include overhead memory allocations that are used by the Heap Monitoring utility. It therefore provides the heap usage that is used when Heap Monitoring is disabled and does not reflect the actual amount of memory that has been allocated by the middleware. That value is accounted for in 'Approximate total heap usage'.
- Approximate total heap usage: The amount of heap memory in bytes used by the middleware, including overhead allocations from the Heap Monitoring utility. When the Heap Monitoring utility is enabled, every allocation has an additional overhead number of bytes allocated so that the middleware can keep track of the meta-data that is output in the heap snapshots. This overhead is not accounted for in the 'Current application heap usage' summary field, but is included in this field. For Java and .NET APIs, this memory only accounts for unmanaged RTI Connext memory, not memory living in the managed heap.
- High watermark: The maximum amount of heap usage by RTI Connext since **com.rti.ndds.utility.Heap↩ Monitoring.enable** (p. 1135) was invoked.
- Alloc count: The number of invocations to malloc, realloc, or calloc operations done by RTI Connext.
- Free count: The number of invocations to the free operation done by RTI Connext.

After the previous summary, and only if you set the parameter print_details to com.rti.dds.infrastructure.true, the method will print the details of every single outstanding heap allocation done by RTI Connext. For example:

```
<P>
                block_id, timestamp, block_size, alloc_method_name, type_name, pool_alloc, pool_buffer_size,
    pool_buffer_count, topic_name, function_name, activity_context

                23087, 1586943520, 16, RTIOsapiHeap_allocateArray, struct RTIEncapsulationInfo, MALLOC, 0,
    0, PRESServiceRequest, PRESWriterHistoryDriver_new,
    "0X101175A,0X76DD63D7,0X984377BC:0X1C1{Name=ShapeTypeParticipant,Domain=110}|CREATE
    Participant|ENABLE|:0X80000088{Entity=Pu,Domain=110}|CREATE Writer WITH TOPIC PRESServiceRequest"
```

- block_id: Block ID of the allocation. This number increases with every allocation.
- timestamp: Timestamp in UTC seconds corresponding to the time where the allocation was done.
- block_size: The number of bytes allocated.
- alloc_method_name: The allocation RTI Connext method name.
- type_name: The allocation typename.
- pool_alloc: Indicates if the heap allocation is a RTI Connext pool allocation (POOL) or a regular allocation (MALLOC).
- pool_buffer_size: For pool allocations, this number indicates the size of the elements in the pool in number of bytes. block_size is equal to (pool_buffer_size $*$ pool_buffer_count).
- pool_buffer_count: For pool allocations, this number indicates the number of buffers allocated for the pool. block↩ _size is equal to (pool_buffer_size $*$ pool_buffer_count).
- topic_name: The topic name associated with the allocation or 'n/a' if it is not available.
- function_name: function name associated with the allocation or 'n/a' if it is not available.
- activity_context: **Activity Context** (p. 288)

Parameters

filename	<<***in***>> *(p. 156)*. Name of file in which to store the snapshot.
print_details	<<***in***>> *(p. 156)*. Indicates if the snapshot will contain only the memory usage summary or the details of the individual allocations.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

8.131 HeapMonitoringParams Class Reference

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

Inherits Struct.

Public Attributes

- **HeapMonitoringSnapshotOutputFormat snapshot_output_format**
  Specify the format of the output of the snapshot.
- **HeapMonitoringSnapshotContentFormat snapshot_content_format**
  It is a bitmap to decide which information of the snapshot will be displayed.

8.131.1 Detailed Description

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

8.131.2 Member Data Documentation

8.131.2.1 snapshot_output_format

```
HeapMonitoringSnapshotOutputFormat snapshot_output_format
```

Initial value:

```
=
        HeapMonitoringSnapshotOutputFormat.NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_STANDARD
```

Specify the format of the output of the snapshot.

The enum can take two options:

- **com.rti.ndds.utility.HeapMonitoringSnapshotOutputFormat.NDDS_UTILITY_HEAP_MONITORING_↩ SNAPSHOT_OUTPUT_FORMAT_STANDARD** (p. 1143):
  The output of the snapshot will be in plain text.

- **com.rti.ndds.utility.HeapMonitoringSnapshotOutputFormat.NDDS_UTILITY_HEAP_MONITORING_↩ SNAPSHOT_OUTPUT_FORMAT_COMPRESSED** (p. 1144):
  The output of the snapshot will be compressed using Zlib techonology.
  The file can be uncompressed using zlib-flate.

[default] com.rti.ndds.utility.HeapMonitoringSnapshotOutputFormat.NDDS_UTILITY_HEAP_MONITORING_↩ SNAPSHOT_OUTPUT_FORMAT_STANDARD (p. 1143).

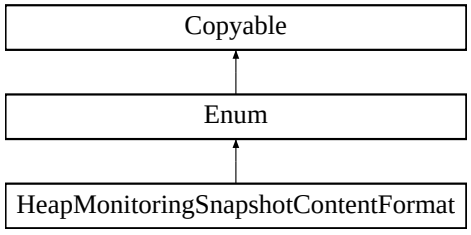
8.131.2.2 snapshot_content_format

```
HeapMonitoringSnapshotContentFormat snapshot_content_format
```

Initial value:

```
=
        HeapMonitoringSnapshotContentFormat.NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT
```

It is a bitmap to decide which information of the snapshot will be displayed.

```suggestion Bitmap that specifies the field to display in the snapshot.

You can combine these values by logically ORing them together. For example: (**com.rti.ndds.utility.HeapMonitoring↩ SnapshotContentFormat.NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC** (p. 1142) | com.rti.ndds.utility.HeapMonitoringSnapshotContentFormat.NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_↩ CONTENT_BIT_FUNCTION)

**[default] com.rti.ndds.utility.HeapMonitoringSnapshotContentFormat.NDDS_UTILITY_HEAP_MONITORING_↩ SNAPSHOT_CONTENT_DEFAULT** (p. 1142)

## 8.132 HeapMonitoringSnapshotContentFormat Class Reference

Bitmap used to decide which information of the snapshot will be displayed.

Inheritance diagram for HeapMonitoringSnapshotContentFormat:




### Static Public Attributes

- static final **HeapMonitoringSnapshotContentFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_BIT_TOPIC**

  *Add the topic to the snapshot of heap monitoring.*
- static final **HeapMonitoringSnapshotContentFormat RTI_OSAPI_HEAP_SNAPSHOT_CONTENT_BIT_↩ FUNCTION**

  *Add the function name to the snapshot of heap monitoring.*
- static final **HeapMonitoringSnapshotContentFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_BIT_ACTIVITY**

  *Add the activity context to the snapshot of heap monitoring. The user can select the information that will be part of the activity context by using the API **com.rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).*
- static final **HeapMonitoringSnapshotContentFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_DEFAULT**

  *Add all the optional attributes to the snapshot of heap monitoring.*
- static final **HeapMonitoringSnapshotContentFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↩ _CONTENT_MINIMAL**

  *Not add any optional attribute to the snapshot of heap monitoring.*
```

Additional Inherited Members

8.132.1 Detailed Description

Bitmap used to decide which information of the snapshot will be displayed.

8.132.2 Member Data Documentation

8.132.2.1 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC

final **HeapMonitoringSnapshotContentFormat** NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC [static]

Add the topic to the snapshot of heap monitoring.

8.132.2.2 RTI_OSAPI_HEAP_SNAPSHOT_CONTENT_BIT_FUNCTION

final **HeapMonitoringSnapshotContentFormat** RTI_OSAPI_HEAP_SNAPSHOT_CONTENT_BIT_FUNCTION [static]

Add the function name to the snapshot of heap monitoring.

8.132.2.3 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_ACTIVITY

final **HeapMonitoringSnapshotContentFormat** NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_↩ ACTIVITY [static]

Add the activity context to the snapshot of heap monitoring. The user can select the information that will be part of the activity context by using the API **com.rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).

8.132.2.4 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT

final **HeapMonitoringSnapshotContentFormat** NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT [static]

Add all the optional attributes to the snapshot of heap monitoring.

8.132.2.5 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MINIMAL

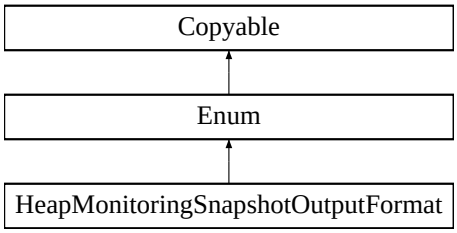
```
final HeapMonitoringSnapshotContentFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MINIMAL
[static]
```

Not add any optional attribute to the snapshot of heap monitoring.

8.133 HeapMonitoringSnapshotOutputFormat Class Reference

Specify the format of the output of the snapshot. RTI Connext.

Inheritance diagram for HeapMonitoringSnapshotOutputFormat:



Static Public Attributes

- static final **HeapMonitoringSnapshotOutputFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_↩ OUTPUT_FORMAT_STANDARD**

  The output of the snapshot will be in plain text.
- static final **HeapMonitoringSnapshotOutputFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_↩ OUTPUT_FORMAT_COMPRESSED**

  The output of the snapshot will be compressed using Zlib techonology.

Additional Inherited Members

8.133.1 Detailed Description

Specify the format of the output of the snapshot. RTI Connext.

8.133.2 Member Data Documentation

8.133.2.1 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_STANDARD

```
final HeapMonitoringSnapshotOutputFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_↩
STANDARD [static]
```

The output of the snapshot will be in plain text.

8.133.2.2 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_COMPRESSED

```
final HeapMonitoringSnapshotOutputFormat NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_↩
COMPRESSED [static]
```

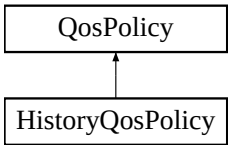
The output of the snapshot will be compressed using Zlib techonology.

The file can be uncompressed using zlib-flate.

8.134 HistoryQosPolicy Class Reference

Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Inheritance diagram for HistoryQosPolicy:



Public Attributes

- **HistoryQosPolicyKind kind**

  Specifies the kind of history to be kept.
- int **depth**

  Specifies the number of samples per instance to be kept, when the `kind` *is com.rti.dds.infrastructure.HistoryQosPolicy↩ Kind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS.*

8.134.1 Detailed Description

Specifies the behavior of RTI Connext in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

This QoS policy specifies how much data must to stored by RTI Connext for a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). It controls whether RTI Connext should deliver only the most recent value, attempt to deliver all intermediate values, or do something in between.

On the publishing side, this QoS policy controls the samples that should be maintained by the **com.rti.dds.↩ publication.DataWriter** (p. 553) on behalf of existing **com.rti.dds.subscription.DataReader** (p. 450) entities. The behavior with regards to a **com.rti.dds.subscription.DataReader** (p. 450) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 230) policy.

On the subscribing side, this QoS policy controls the samples that should be maintained until the application "takes" them from RTI Connext.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

See also

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

8.134.2 Usage

This policy controls the behavior of RTI Connext when the value of an instance changes before it is finally communicated to **com.rti.dds.subscription.DataReader** (p. 450) entities.

When a **com.rti.dds.publication.DataWriter** (p. 553) sends data, or a **com.rti.dds.subscription.DataReader** (p. 450) receives data, the data sent or received is stored in a cache whose contents are controlled by this QoS policy. This QoS policy interacts with **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) by controlling whether RTI Connext guarantees that *all* of the sent data is received (com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicy↩ Kind.KEEP_ALL_HISTORY_QOS) or if only the last N data values sent are guaranteed to be received (com.rti.dds.↩ infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS)–this is a reduced level of reliability.

The amount of data that is sent to new DataReaders who have configured their **com.rti.dds.infrastructure.Durability↩ QosPolicy** (p. 830) to receive previously published data is controlled by **com.rti.dds.infrastructure.DurabilityQos↩ Policy.writer_depth** (p. 834), not by the History QoS policy.

Note that the History QoS policy does not control the *physical* sizes of the send and receive queues. The memory allocation for the queues is controlled by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

If `kind` is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS (the default), then RTI Connext will only attempt to keep the latest values of the instance and discard the older ones. In this case, the value of `depth` regulates the maximum number of values (up to and including the most current one) RTI Connext will maintain and deliver until the samples are fully acknowledged. After N values have been sent or received, any new data will overwrite the oldest data in the queue. Thus the queue acts like a circular buffer of length *N*.

For keyed-data, there is different behavior on the publishing and subscribing sides associated with how invalid samples representing the disposal of or unregistration from an instance affect history.

On the publishing side, unregistering from or disposing of an instance creates an invalid sample that is accounted for in the history depth. This means that an invalid sample may replace a value that is currently being stored in the writer queue.

On the subscribing side, however, invalid samples do not count towards history depth and will not replace a value that is being stored in the reader queue.

On both the publishing and subscribing sides, there can only ever be one invalid sample per-instance and that one sample can be in different states depending on whether the instance has been disposed, unregistered, or both.

The default (and most common setting) for `depth` is 1, indicating that only the most recent value should be delivered.

If `kind` is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS, then RTI Connext will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that RTI Connext can use to keep this history are limited by the settings of the **RESOURCE_LIMITS** (p. 259). If the limit is reached, then the behavior of RTI Connext will depend on the **RELIABILITY** (p. 258). If the Reliability `kind` is **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), then the old values will be discarded. If Reliability `kind` is RELIABLE, then RTI Connext will block the **com.rti.dds.publication.DataWriter** (p. 553) until it can deliver the necessary old values to all subscribers.

8.134.3 Consistency

This QoS policy's `depth` must be consistent with the **RESOURCE_LIMITS** (p. 259) `max_samples_per_instance`. For these two QoS to be consistent, they must verify that *depth* <= *max_samples_per_instance*.

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.134.4 Member Data Documentation

8.134.4.1 kind

HistoryQosPolicyKind kind

Specifies the kind of history to be kept.

For DataWriters, the samples are only kept either until they are fully acknowledged by all matching DataReaders or until they are replaced or removed. Samples can be replaced or removed for a number of reasons, including but not limited to **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1234) expiration, replacement because the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) has been exceeded, or one of the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590) settings has been exceeded.

[default] com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS

8.134.4.2 depth

int depth

Specifies the number of samples per instance to be kept, when the kind is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS.

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 259) policy. That is:

depth <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)

When the kind is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS, the depth has no effect. Its implied value is infinity (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 259) policy).

When a DataWriter responds to a TopicQuery, the samples that are evaluated against the TopicQuery filter are only those samples that fall within the **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834), not this depth.

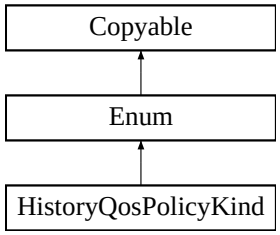
[default] 1

[range] [1,100 million], <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)

8.135 HistoryQosPolicyKind Class Reference

Kinds of history.

Inheritance diagram for HistoryQosPolicyKind:



Static Public Attributes

- static final **HistoryQosPolicyKind KEEP_LAST_HISTORY_QOS**

  [default] *Keep the last* `depth` *samples.*
- static final **HistoryQosPolicyKind KEEP_ALL_HISTORY_QOS**

  Keep all the samples.

Additional Inherited Members

8.135.1 Detailed Description

Kinds of history.

QoS:

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

8.135.2 Member Data Documentation

8.135.2.1 KEEP_LAST_HISTORY_QOS

```
final HistoryQosPolicyKind KEEP_LAST_HISTORY_QOS [static]
```

[default] Keep the last `depth` samples.

On the publishing side, RTI Connext will only attempt to keep the most recent `depth` samples of each instance of data (identified by its key) managed by the **com.rti.dds.publication.DataWriter** (p. 553). Invalid samples representing a disposal or unregistration of an instance count towards the depth and may replace other DDS samples currently in the DataWriter queue for the same instance.

On the subscribing side, the **com.rti.dds.subscription.DataReader** (p. 450) will only attempt to keep the most recent `depth` samples received for each instance (identified by its key) until the application takes them via the **com.rti.dds.↩ subscription.DataReader** (p. 450) 's **take()** operation.
Invalid samples representing a disposal or unregistration of an instance do not count towards the history depth and will not replace other DDS samples currently in the DataReader queue for the same instance.

8.135.2.2 KEEP_ALL_HISTORY_QOS

```
final HistoryQosPolicyKind KEEP_ALL_HISTORY_QOS [static]
```

Keep *all* the samples.

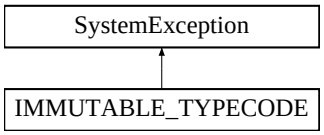
On the publishing side, RTI Connext will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the **com.rti.dds.publication.DataWriter** (p. 553) until they can be delivered to all subscribers.

On the subscribing side, RTI Connext will attempt to keep all samples of each instance of data (identified by its key) managed by the **com.rti.dds.subscription.DataReader** (p. 450). These samples are kept until the application takes them from RTI Connext via the **take()** operation.

8.136 IMMUTABLE_TYPECODE Class Reference

An attempt was made to modify a **com.rti.dds.typecode.TypeCode** (p. 1873) that was received from a remote object.

Inheritance diagram for IMMUTABLE_TYPECODE:



8.136.1 Detailed Description

An attempt was made to modify a **com.rti.dds.typecode.TypeCode** (p. 1873) that was received from a remote object.

The built-in publication and subscription readers provide access to information about the remote **com.rti.dds.↩ publication.DataWriter** (p. 553) and **com.rti.dds.subscription.DataReader** (p. 450) entities in the distributed system. Among other things, the data from these built-in readers contains the **com.rti.dds.typecode.TypeCode** (p. 1873) for these entities. Modifying this received **com.rti.dds.typecode.TypeCode** (p. 1873) is not permitted.

8.137 InconsistentTopicStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of the pairs (**com.rti.dds.subscription.DataReader** (p. 450),**com.rti.dds.publication.DataWriter** (p. 553)) whose topic names match the **com.rti.dds.topic.Topic** (p. 1807) to which this status is attached and whose types are inconsistent according to the rules defined in **com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy** (p. 1935).*
- int **total_count_change**

  *The incremental number of inconsistent pairs (**com.rti.dds.subscription.DataReader** (p. 450),**com.rti.dds.↩ publication.DataWriter** (p. 553)) for the **com.rti.dds.topic.Topic** (p. 1807) to which this status is attached, that have been discovered since the last time this status was read.*

8.137.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS

Entity:

com.rti.dds.topic.Topic (p. 1807)

Listener:

com.rti.dds.topic.TopicListener (p. 1822)

Every time a **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.publication.DataWriter** (p. 553) with the same **com.rti.dds.topic.Topic** (p. 1807) do not match because the type-consistency enforcement policy fails, the inconsistent topic status is updated for the **com.rti.dds.topic.Topic** (p. 1807).

See also

com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy (p. 1935)

8.137.2 Member Data Documentation

8.137.2.1 total_count

```
int total_count
```

Total cumulative count of the pairs (**com.rti.dds.subscription.DataReader** (p. 450),**com.rti.dds.publication.Data↩ Writer** (p. 553)) whose topic names match the **com.rti.dds.topic.Topic** (p. 1807) to which this status is attached and whose types are inconsistent according to the rules defined in **com.rti.dds.infrastructure.TypeConsistency↩ EnforcementQosPolicy** (p. 1935).

8.137.2.2 total_count_change

```
int total_count_change
```

The incremental number of inconsistent pairs (**com.rti.dds.subscription.DataReader** (p. 450),**com.rti.dds.↩ publication.DataWriter** (p. 553)) for the **com.rti.dds.topic.Topic** (p. 1807) to which this status is attached, that have been discovered since the last time this status was read.

8.138 InetAddressSeq Class Reference

Declares IDL sequence< com.rti.dds.infrastructure.java.net.InetAddress >

Inherits ArraySequence.

Public Member Functions

- **InetAddressSeq** ()
  Construct a new empty sequence.
- **InetAddressSeq** (int initial_maximum)
  Construct a new empty sequence with the given maximum.
- **InetAddressSeq** (Collection addresses)
  Construct a new sequence containing all of the given addresses.

8.138.1 Detailed Description

Declares IDL sequence< com.rti.dds.infrastructure.java.net.InetAddress >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.java.net.InetAddress

8.138.2 Constructor & Destructor Documentation

8.138.2.1 InetAddressSeq() [1/3]

```
InetAddressSeq ( )
```

Construct a new empty sequence.

8.138.2.2 InetAddressSeq() [2/3]

```
InetAddressSeq (
            int initial_maximum )
```

Construct a new empty sequence with the given maximum.

8.138.2.3 InetAddressSeq() [3/3]

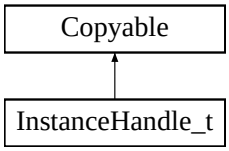
```
InetAddressSeq (
            Collection addresses )
```

Construct a new sequence containing all of the given addresses.

8.139 InstanceHandle_t Class Reference

Type definition for an instance handle.

Inheritance diagram for InstanceHandle_t:



Public Member Functions

- **InstanceHandle_t** ()
- **InstanceHandle_t** (**InstanceHandle_t** src)
- boolean **is_nil** ()

  Compare this handle to ***com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL*** *(*p. *1156).*
- Object **copy_from** (Object src)

  Copy value of a data type from source.
- boolean **equals** (Object other)

  Compares this instance handle with another handle for equality.
- int **compare** (**InstanceHandle_t** other)

  Compares this instance handle with another handle.

Static Public Attributes

- static final **InstanceHandle_t** **HANDLE_NIL**

  The NIL instance handle.

8.139.1 Detailed Description

Type definition for an instance handle.

Handle to identify different instances of the same **com.rti.dds.topic.Topic** (p. 1807) of a certain type.

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

com.rti.dds.subscription.SampleInfo.instance_handle (p. 1640)

8.139.2 Constructor & Destructor Documentation

8.139.2.1 InstanceHandle_t() [1/2]

```
InstanceHandle_t ( )
```

Construct a new instance handle equal to the nil handle.

See also

HANDLE_NIL (p. 1156)

Referenced by **InstanceHandle_t.copy_from()**, and **InstanceHandle_t.equals()**.

8.139.2.2 InstanceHandle_t() [2/2]

```
InstanceHandle_t (
            InstanceHandle_t src )
```

Construct a new instance handle equal to the given handle.

Exceptions

NullPointerException	if src is null

References **InstanceHandle_t.copy_from()**.

8.139.3 Member Function Documentation

8.139.3.1 is_nil()

```
boolean is_nil ( )
```

Compare this handle to **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

Returns

com.rti.dds.infrastructure.true if the given instance handle is equal to **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) or com.rti.dds.infrastructure.false otherwise.

See also

com.rti.dds.infrastructure.InstanceHandle_t.equals (p. 1154)

8.139.3.2 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

References **InstanceHandle_t.InstanceHandle_t()**.

Referenced by **SampleInfo.copy_from()**, **WriteParams_t.copy_from()**, **InstanceHandle_t.InstanceHandle_t()**, and **WriteParams_t.WriteParams_t()**.

8.139.3.3 equals()

```
boolean equals (
            Object other )
```

Compares this instance handle with another handle for equality.

Parameters

other	<<***in***>> *(*p. *156)* The other handle to be compared with this handle. Cannot be null.

Returns

com.rti.dds.infrastructure.true if the two handles have equal values, or com.rti.dds.infrastructure.false otherwise.

See also

com.rti.dds.infrastructure.InstanceHandle_t.is_nil (p. 1153)

References **InstanceHandle_t.equals()**, and **InstanceHandle_t.InstanceHandle_t()**.

Referenced by **InstanceHandle_t.equals()**, and **WriteParams_t.WriteParams_t()**.

8.139.3.4 compare()

```
int compare (
             InstanceHandle_t other )
```

Compares this instance handle with another handle.

Parameters

other	<<***in***>> (p. *156)* The other handle to be compared with this handle. Cannot be null.

Returns

If the two handles are equal, the function returns 0. If self is greater than other the function returns a positive number; otherwise, it returns a negative number.

See also

com.rti.dds.infrastructure.InstanceHandle_t.is_nil (p. 1153)

8.139.4 Member Data Documentation

8.139.4.1 HANDLE_NIL

```
final InstanceHandle_t HANDLE_NIL [static]
```

The NIL instance handle.

Special **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) value

See also

com.rti.dds.infrastructure.InstanceHandle_t.is_nil (p. 1153)

Referenced by **InstanceHandleSeq.fill()**, **DynamicDataWriter.lookup_instance()**, **DynamicDataWriter.register_instance()**, **DynamicDataWriter.register_instance_w_timestamp()**, and **WriteParams_t.WriteParams_t()**.

8.140 InstanceHandleSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) > .

Inherits ArraySequence.

Public Member Functions

- **InstanceHandleSeq** ()

  *Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.*
- **InstanceHandleSeq** (int initial_maximum)

  *Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.*
- **InstanceHandleSeq** (Collection<?> elements)

  *Construct a new sequence containing the given **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.*
- void **fill** (int size)

  Fill this sequence with the given number of instance handles.

8.140.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) > .

When reading into this sequence (as with **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 566) or **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 465)), the contents of any existing handles in this sequence will be overwritten to avoid the expense of allocating new handles. Therefore, it is generally not a good idea to add handles to a sequence that you obtained elsewhere (e.g. from a Status object or as a result of calling **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098)). Any null elements will be replaced by new handles. To avoid allocating new handles on the fly, use the method **com.rti.dds.infrastructure.InstanceHandleSeq.fill** (p. 1158).

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.InstanceHandle_t (p. 1152)

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.140.2 Constructor & Destructor Documentation

8.140.2.1 InstanceHandleSeq() [1/3]

```
InstanceHandleSeq ( )
```

Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.

The maximum of the sequence will be set to a default value.

8.140.2.2 InstanceHandleSeq() [2/3]

```
InstanceHandleSeq (
            int initial_maximum )
```

Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.

The maximum of the sequence will be set to the given value.

Parameters

initial_maximum	<<***in***>> *(*p. *156)* Maximum length of sequence.

8.140.2.3 InstanceHandleSeq() [3/3]

```
InstanceHandleSeq (
            Collection<?> elements )
```

Construct a new sequence containing the given **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) elements.

The maximum of the sequence will be set to the size of the given collection.

Parameters

elements	<<***in***>> *(p. 156)* Elements to construct a sequence with.

8.140.3 Member Function Documentation

8.140.3.1 fill()

```
void fill (
            int size )
```

Fill this sequence with the given number of instance handles.

Ensure that this sequence has at least the given size and that all elements up to that count are non-null.

Parameters

size	<<***in***>> *(p. 156)* Size of sequence to ensure.

Exceptions

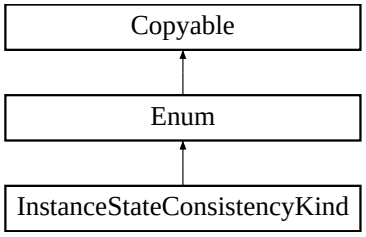
RETCODE_BAD_PARAMETER *(p. 1594)*	If size < 0

References **InstanceHandle_t.HANDLE_NIL**.

8.141 InstanceStateConsistencyKind Class Reference

<<***extension***>> *(p. 155)* Whether instance state consistency is enabled.

Inheritance diagram for InstanceStateConsistencyKind:



Static Public Attributes

- static final **InstanceStateConsistencyKind NO_RECOVER_INSTANCE_STATE_CONSISTENCY**

  Instance state is not restored on a DataReader after reconnecting with a DataWriter until the DataWriter sends a new sample.
- static final **InstanceStateConsistencyKind RECOVER_INSTANCE_STATE_CONSISTENCY**

  Instance state is restored on the DataReader after it reconnects with a DataWriter that has regained liveliness, even before the DataWriter sends a new sample.

Additional Inherited Members

8.141.1 Detailed Description

<<***extension***>> *(*p. *155)* Whether instance state consistency is enabled.

QoS:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

8.141.2 Member Data Documentation

8.141.2.1 NO_RECOVER_INSTANCE_STATE_CONSISTENCY

```
final InstanceStateConsistencyKind NO_RECOVER_INSTANCE_STATE_CONSISTENCY [static]
```

Instance state is not restored on a DataReader after reconnecting with a DataWriter until the DataWriter sends a new sample.

When DataReaders rediscover DataWriters, they will not request updated instance state data. DataWriters always provide instance state data alongside each sample update regardless of this setting.

8.141.2.2 RECOVER_INSTANCE_STATE_CONSISTENCY

```
final InstanceStateConsistencyKind RECOVER_INSTANCE_STATE_CONSISTENCY [static]
```

Instance state is restored on the DataReader after it reconnects with a DataWriter that has regained liveliness, even before the DataWriter sends a new sample.

When DataReaders rediscover DataWriters, they will request updated instance state data. DataWriters will respond to requests for updated instance state data and publish updates on the ServiceRequest channel. DataWriters still provide instance state data alongside each sample update regardless of this setting.

The following limitation applies to using RECOVER_INSTANCE_STATE_CONSISTENCY: If the **com.rti.dds.infrastructure.DestinationOrderQosPolicyKind** (p. 638) is set to com.rti.dds.infrastructure.DestinationOrderQosPolicyKind.DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, RECOVER_INSTANCE_STATE_CONSISTENCY can only be used if scope is set to idref_DestinationOrderQosPolicyScopeKind_INSTANCE_SCOPE_DESTINATIONORDER_QOS (the default).

8.142 InstanceStateKind Class Reference

Indicates if the samples are from a live **com.rti.dds.publication.DataWriter** (p. 553) or not.

Static Public Attributes

- static final int **ALIVE_INSTANCE_STATE** = 0x0001 << 0

  Instance is currently in existence.
- static final int **NOT_ALIVE_DISPOSED_INSTANCE_STATE** = 0x0001 << 1

  Not alive disposed instance. The instance has been disposed by a DataWriter.
- static final int **NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** = 0x0001 << 2

  *Not alive no writers for instance. None of the **com.rti.dds.publication.DataWriter** (p. 553) objects that are currently alive (according to the **LIVELINESS** (p. 239)) are writing the instance.*
- static final int **ANY_INSTANCE_STATE** = 0xffff

  Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.
- static final int **NOT_ALIVE_INSTANCE_STATE** = 0x006

  Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

8.142.1 Detailed Description

Indicates if the samples are from a live **com.rti.dds.publication.DataWriter** (p. 553) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161) indicates that (a) samples have been received for the instance, (b) there are live **com.rti.dds.publication.DataWriter** (p. 553) entities writing the instance, and (c) the instance has not been explicitly disposed (or else more samples have been received after it was disposed).

- **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) indicates the instance was explicitly disposed by a **com.rti.dds.publication.DataWriter** (p. 553) by means of the dispose operation.

- **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) indicates the instance has been declared as not-alive by the **com.rti.dds.subscription.DataReader** (p. 450) because it detected that there are no live **com.rti.dds.publication.DataWriter** (p. 553) entities writing that instance.

The precise behavior events that cause the instance state to change depends on the setting of the OWNERSHIP QoS:

- If **OWNERSHIP** (p. 244) is set to **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_↩ OWNERSHIP_QOS** (p. 1348), then the instance state becomes **com.rti.dds.subscription.InstanceState↩ Kind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) only if the **com.rti.dds.publication.DataWriter** (p. 553) that "owns" the instance explicitly disposes it. The instance state becomes **com.rti.dds.subscription.↩ InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161) again only if the **com.rti.dds.publication.DataWriter** (p. 553) that owns the instance writes it.

- If **OWNERSHIP** (p. 244) is set to **com.rti.dds.infrastructure.OwnershipQosPolicyKind.SHARED_↩ OWNERSHIP_QOS** (p. 1347), then the instance state becomes **com.rti.dds.subscription.InstanceState↩ Kind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) if any **com.rti.dds.publication.DataWriter** (p. 553) explicitly disposes the instance. The instance state becomes **com.rti.dds.subscription.Instance↩ StateKind.ALIVE_INSTANCE_STATE** (p. 1161) as soon as any **com.rti.dds.publication.DataWriter** (p. 553) writes the instance again.

The instance state available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) is a snapshot of the instance state of the instance at the time the collection was obtained (i.e. at the time read or take was called). The instance state is therefore the same for all samples in the returned collection that refer to the same instance.

8.142.2 Member Data Documentation

8.142.2.1 ALIVE_INSTANCE_STATE

```
final int ALIVE_INSTANCE_STATE = 0x0001 << 0  [static]
```

Instance is currently in existence.

8.142.2.2 NOT_ALIVE_DISPOSED_INSTANCE_STATE

```
final int NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0001 << 1  [static]
```

Not alive disposed instance. The instance has been disposed by a DataWriter.

8.142.2.3 NOT_ALIVE_NO_WRITERS_INSTANCE_STATE

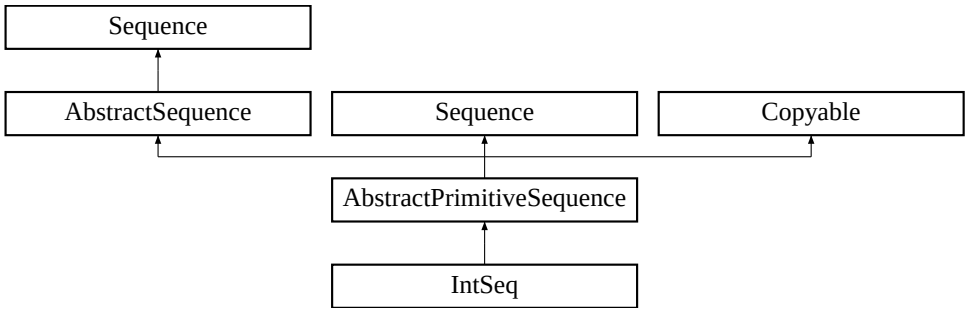
```
final int NOT_ALIVE_NO_WRITERS_INSTANCE_STATE = 0x0001 << 2  [static]
```

Not alive no writers for instance. None of the **com.rti.dds.publication.DataWriter** (p. 553) objects that are currently alive (according to the **LIVELINESS** (p. 239)) are writing the instance.

8.143 IntSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ int >

Inheritance diagram for IntSeq:



Public Member Functions

- **IntSeq** ()

  Constructs an empty sequence of integers with an initial maximum of zero.
- **IntSeq** (int initialMaximum)

  Constructs an empty sequence of integers with the given initial maximum.
- **IntSeq** (int[ ] ints)

  Constructs a new sequence containing the given integers.
- boolean **addAllInt** (int[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- boolean **addAllInt** (int[ ] elements)
- void **addInt** (int element)

  Append the element to the end of the sequence.
- void **addInt** (int index, int element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- int **getInt** (int index)

  Returns the integer at the given index.
- int **setInt** (int index, int element)

  Set the new integer at the given index and return the old integer.
- void **setInt** (int dstIndex, int[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.

- int[ ] **toArrayInt** (int[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  *A wrapper for **getInt(int)** (p. 1165) that returns a java.lang.Integer.*
- Object **set** (int index, Object element)

  *A wrapper for **setInt()** (p. 1165).*
- void **add** (int index, Object element)

  *A wrapper for **addInt(int, int)** (p. 1165).*

8.143.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ int >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.int

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.143.2 Constructor & Destructor Documentation

8.143.2.1 IntSeq() [1/3]

```
IntSeq ( )
```

Constructs an empty sequence of integers with an initial maximum of zero.

8.143.2.2 IntSeq() [2/3]

```
IntSeq (
            int initialMaximum )
```

Constructs an empty sequence of integers with the given initial maximum.

8.143.2.3 IntSeq() [3/3]

```
IntSeq (
            int[] ints )
```

Constructs a new sequence containing the given integers.

Parameters

ints	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **IntSeq.addAllInt()**.

8.143.3 Member Function Documentation

8.143.3.1 addAllInt() [1/2]

```
boolean addAllInt (
            int[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **IntSeq.addAllInt()**, and **IntSeq.IntSeq()**.

8.143.3.2 addAllInt() [2/2]

```
boolean addAllInt (
            int[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **IntSeq.addAllInt()**.

8.143.3.3 addInt() [1/2]

```
void addInt (
             int element )
```

Append the element to the end of the sequence.

Referenced by **IntSeq.add()**.

8.143.3.4 addInt() [2/2]

```
void addInt (
             int index,
             int element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.143.3.5 getInt()

```
int getInt (
             int index )
```

Returns the integer at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **IntSeq.get()**, **DynamicData.get_uint_seq()**, and **DynamicData.set_ushort_seq()**.

8.143.3.6 setInt() [1/2]

```
int setInt (
             int index,
             int element )
```

Set the new integer at the given index and return the old integer.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **IntSeq.set()**.

8.143.3.7 setInt() [2/2]

```
void setInt (
            int dstIndex,
            int[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.143.3.8 toArrayInt()

```
int[] toArrayInt (
            int[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.143.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. `1168`), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_int_seq()**, **DynamicData.get_uint_seq()**, **DynamicData.get_ushort_seq()**, **DynamicData.set_int_seq()**, and **DynamicData.set_ushort_seq()**.

8.143.3.10 get()

```
Object get (
             int index )
```

A wrapper for **getInt(int)** (p. 1165) that returns a java.lang.Integer.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **IntSeq.getInt()**.

8.143.3.11 set()

```
Object set (
             int index,
             Object element )
```

A wrapper for **setInt()** (p. 1165).

Exceptions

ClassCastException	if the element is not of type Integer.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **IntSeq.setInt()**.

Referenced by **DynamicData.get_ushort_seq()**, and **DynamicData.set_uint_seq()**.

8.143.3.12 add()

```
void add (
            int index,
            Object element )
```

A wrapper for **addInt(int, int)** (p. 1165).

Exceptions

ClassCastException	if the element is not of type Integer.

See also

java.util.List::add(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **IntSeq.addInt()**.

8.144 Sample< T >.Iterator< T > Interface Template Reference

Provides access to a collection of middleware-loaned samples.

Inherits ListIterator< Sample< T > >, and Closeable.

Inherited by SampleIteratorImpl< T >.

Public Member Functions

- int **size** ()

  Returns the number of samples.
- boolean **hasNext** ()
- **Sample**< T > **next** ()
- boolean **hasPrevious** ()
- **Sample**< T > **previous** ()
- int **nextIndex** ()
- int **previousIndex** ()
- void **remove** ()
- void **set** (**Sample**< T > o)
- void **add** (**Sample**< T > o)
- void **close** ()

  Returns the loaned samples to the middleware.

8.144.1 Detailed Description

Provides access to a collection of middleware-loaned samples.

The samples in this container are loaned from the middleware and must be returned at some point.

To return the loan, use **com.rti.connext.infrastructure.Sample<T>.Iterator<T>.close()** (p. 1171) , from `java.io.Closeable` or (since Java 7) enclose this object within a `try-with-resources` block (see **Taking loaned samples** (p. 149)).

Alternatively, if the loan is not returned by the application code, it will be automatically returned when this object is garbage-collected.

The contents of this container should not be modified and references to the samples it contains are only valid before the loan is returned.

This interface extends `java.util.ListIterator<Sample<T>>`. However any operations that modify the underlying list are not supported.

Template Parameters

T	The data type of the contained Samples

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(int) (p. 1573)

com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequests(int) (p. 1550)

Taking loaned samples (p. 149)

Taking samples by copy (p. 150)

8.144.2 Member Function Documentation

8.144.2.1 size()

```
int size ( )
```

Returns the number of samples.

8.144.2.2 hasNext()

```
boolean hasNext ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.3 next()

```
Sample< T > next ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.4 hasPrevious()

```
boolean hasPrevious ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.5 previous()

```
Sample< T > previous ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.6 nextIndex()

```
int nextIndex ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.7 previousIndex()

```
int previousIndex ( )
```

From java.util.ListIterator<Sample<T>>

8.144.2.8 remove()

```
void remove ( )
```

Exceptions

UnsupportedOperationException	always.

8.144.2.9 set()

```
void set (
            Sample< T > o )
```

Exceptions

UnsupportedOperationException	always.

8.144.2.10 add()

```
void add (
            Sample< T > o )
```

Exceptions

UnsupportedOperationException	always.

8.144.2.11 close()

```
void close ( )
```

Returns the loaned samples to the middleware.

After calling this operation this object cannot be accessed again.

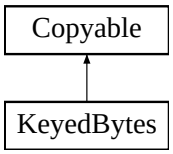
See also

com.rti.ndds.example.FooDataReader.return_loan (p. 1094) (for more information on how the middleware loans samples)

8.145 KeyedBytes Class Reference

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

Inheritance diagram for KeyedBytes:



Public Member Functions

- **KeyedBytes** ()

  Default Constructor.
- **KeyedBytes** (**KeyedBytes** src)

  Copy constructor.
- **KeyedBytes** (int theLength)

  Constructor that specifies the allocated sizes.
- Object **copy_from** (Object src)

  Copy src into this object.

Public Attributes

- String **key**

  Instance key associated with the specified value.
- int **length**

  Number of bytes to serialize.
- int **offset**

  Offset from which to start serializing bytes.
- byte[ ] **value**

  com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes array value.

8.145.1 Detailed Description

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

8.145.2 Constructor & Destructor Documentation

8.145.2.1 KeyedBytes() [1/3]

```
KeyedBytes ( )
```

Default Constructor.

The default constructor initializes the newly created object with empty key, null value, zero length, and zero offset.

Referenced by **KeyedBytes.copy_from()**.

8.145.2.2 KeyedBytes() [2/3]

```
KeyedBytes (
            KeyedBytes src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Object to copy from.

Exceptions

NullPointerException	if src is null.

References **KeyedBytes.copy_from()**.

8.145.2.3 KeyedBytes() [3/3]

```
KeyedBytes (
            int theLength )
```

Constructor that specifies the allocated sizes.

After this method is called, key is initialized with the empty string and length and offset are set to zero.

Parameters

theLength	<<***in***>> *(*p. *156)* Size of the allocated bytes array. bytes array.

Exceptions

IllegalArgumentException	if size is negative

References **KeyedBytes.length**, **KeyedBytes.offset**, and **KeyedBytes.value**.

8.145.3 Member Function Documentation

8.145.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy src into this object.

This method performs a deep copy of `src` and it allocates memory for the value if required.

Parameters

src	<<***in***>> *(*p. *156)* Object to copy from.

Returns

this if success. Otherwise, null.

Exceptions

NullPointerException	if `src` is null.

Implements **Copyable** (p. 445).

References **KeyedBytes.key**, **KeyedBytes.KeyedBytes()**, **KeyedBytes.length**, **KeyedBytes.offset**, and **Keyed↩ Bytes.value**.

Referenced by **KeyedBytes.KeyedBytes()**.

8.145.4 Member Data Documentation

8.145.4.1 key

```
String key
```

Instance key associated with the specified value.

Referenced by **KeyedBytes.copy_from()**, **KeyedBytesDataWriter.dispose()**, **KeyedBytesDataWriter.dispose_w_timestamp()**, **KeyedBytesDataReader.get_key_value()**, **KeyedBytesDataWriter.get_key_value()**, **KeyedBytesDataReader.lookup_instance()**, **KeyedBytesDataWriter.lookup_instance()**, **KeyedBytesDataWriter.register_instance()**, **KeyedBytesDataWriter.register_instance_w_timestamp()**, **KeyedBytesDataWriter.unregister_instance()**, **KeyedBytesDataWriter.unregister_instance_w_timestamp()**, **KeyedBytesDataWriter.write()**, and **KeyedBytesDataWriter.write_w_timestamp()**.

8.145.4.2 length

```
int length
```

Number of bytes to serialize.

Referenced by **KeyedBytes.copy_from()**, **KeyedBytes.KeyedBytes()**, **KeyedBytesDataWriter.write()**, and **KeyedBytesDataWriter.write_w_timestamp()**.

8.145.4.3 offset

```
int offset
```

Offset from which to start serializing bytes.

The first position of the bytes array has offset 0.

Referenced by **KeyedBytes.copy_from()**, **KeyedBytes.KeyedBytes()**, **KeyedBytesDataWriter.write()**, and **KeyedBytesDataWriter.write_w_timestamp()**.

8.145.4.4 value

```
byte [] value
```

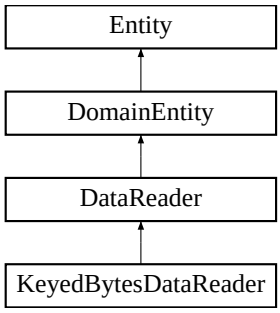
com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes array value.

Referenced by **KeyedBytes.copy_from()**, **KeyedBytes.KeyedBytes()**, **KeyedBytesDataWriter.write()**, and **KeyedBytesDataWriter.write_w_timestamp()**.

8.146 KeyedBytesDataReader Class Reference

<<***interface***>> *(*p. *156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.

Inheritance diagram for KeyedBytesDataReader:



Public Member Functions

- void **read** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **take** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **read_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩**Condition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (*p. *1514).*
- void **take_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩**Condition** condition)

  Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take operation.
- void **read_next_sample** (**KeyedBytes** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **take_next_sample** (**KeyedBytes** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **read_instance** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Instance**↩**Handle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **take_instance** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Instance**↩**Handle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **read_instance_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_↩samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (*p. *1514).*
- void **take_instance_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_↩samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

*Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*

- void **read_next_instance** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_next_instance** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_next_instance_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max↩ _samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_next_instance_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max↩ _samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **return_loan** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq)

  *Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.↩ subscription.DataReader** (p. 450).*
- void **get_key_value** (**KeyedBytes** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- String **get_key_value** (**InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Retrieve the instance `key` that corresponds to an instance `handle`.*
- **InstanceHandle_t lookup_instance** (**KeyedBytes** key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.
- **InstanceHandle_t lookup_instance** (String key)

  *<<**extension**>> (p. 155) Retrieve the instance `handle` that corresponds to an instance `key`.*

Additional Inherited Members

8.146.1 Detailed Description

<<***interface***>> *(p. 156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

com.rti.dds.subscription.DataReader (p. 450)

8.146.2 Member Function Documentation

8.146.2.1 read()

```
void read (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_untyped()**.

8.146.2.2 take()

```
void take (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_untyped()**.

8.146.2.3 read_w_condition()

```
void read_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076)

References **DataReader.read_w_condition_untyped()**.

8.146.2.4 take_w_condition()

```
void take_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take operation.

See also

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

References **DataReader.take_w_condition_untyped()**.

8.146.2.5 read_next_sample()

```
void read_next_sample (
              KeyedBytes received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_sample (p. 1078)

References **DataReader.read_next_sample_untyped()**.

8.146.2.6 take_next_sample()

```
void take_next_sample (
              KeyedBytes received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_next_sample (p. 1079)

References **DataReader.take_next_sample_untyped()**.

8.146.2.7 read_instance()

```
void read_instance (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_instance (p. 1081)

References **DataReader.read_instance_untyped()**.

8.146.2.8 take_instance()

```
void take_instance (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_instance (p. 1082)

References **DataReader.take_instance_untyped()**.

8.146.2.9 read_instance_w_condition()

```
void read_instance_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_instance_w_condition (p. 1088)

References **DataReader.read_instance_w_condition_untyped()**.

8.146.2.10 take_instance_w_condition()

```
void take_instance_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.take_instance_w_condition (p. 1089)

References **DataReader.take_instance_w_condition_untyped()**.

8.146.2.11 read_next_instance()

```
void read_next_instance (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

References **DataReader.read_next_instance_untyped()**.

8.146.2.12 take_next_instance()

```
void take_next_instance (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

References **DataReader.take_next_instance_untyped()**.

8.146.2.13 read_next_instance_w_condition()

```
void read_next_instance_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_next_instance_w_condition (p. 1091)

References **DataReader.read_next_instance_w_condition_untyped()**.

8.146.2.14 take_next_instance_w_condition()

```
void take_next_instance_w_condition (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.take_next_instance_w_condition (p. 1092)

References **DataReader.take_next_instance_w_condition_untyped()**.

8.146.2.15 return_loan()

```
void return_loan (
              KeyedBytesSeq received_data,
              SampleInfoSeq info_seq )
```

Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.return_loan (p. 1094)

References **DataReader.return_loan_untyped()**.

8.146.2.16 get_key_value() [1/2]

```
void get_key_value (
              KeyedBytes key_holder,
              InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

References **DataReader.get_key_value_untyped()**.

8.146.2.17 get_key_value() [2/2]

```
String get_key_value (
              InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

References **DataReader.get_key_value_untyped()**, and **KeyedBytes.key**.

8.146.2.18 lookup_instance() [1/2]

```
InstanceHandle_t lookup_instance (
            KeyedBytes key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

com.rti.ndds.example.FooDataReader.lookup_instance (p. 1096)

References **DataReader.lookup_instance_untyped()**.

8.146.2.19 lookup_instance() [2/2]

```
InstanceHandle_t lookup_instance (
           String key )
```

<<***extension***>> *(p. 155)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

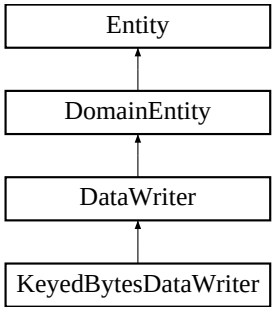
com.rti.ndds.example.FooDataReader.lookup_instance (p. 1096)

References **KeyedBytes.key**, and **DataReader.lookup_instance_untyped()**.

8.147 KeyedBytesDataWriter Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.

Inheritance diagram for KeyedBytesDataWriter:



Public Member Functions

- **InstanceHandle_t register_instance** (**KeyedBytes** instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- **InstanceHandle_t register_instance** (String key)

  *<<**extension**>> (p. 155) Informs RTI Connext that the application will be modifying a particular instance.*
- **InstanceHandle_t register_instance_w_timestamp** (**KeyedBytes** instance_data, **Time_t** source_timestamp)

  Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance except that the application provides the value for the `source_timestamp`.
- **InstanceHandle_t register_instance_w_timestamp** (String key, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes↩ DataWriter.KeyedBytesDataWriter.register_instance except that the application provides the value for the `source_`↩ `timestamp`.*
- void **unregister_instance** (**KeyedBytes** instance_data, **InstanceHandle_t** handle)

  Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance.
- void **unregister_instance** (String key, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesData↩ Writer.KeyedBytesDataWriter.register_instance.*
- void **unregister_instance_w_timestamp** (**KeyedBytes** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.unregister_instance except that it also provides the value for the `source_timestamp`.
- void **unregister_instance_w_timestamp** (String key, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ BytesDataWriter.KeyedBytesDataWriter.unregister_instance except that it also provides the value for the `source`↩ `_timestamp`.*
- void **write** (**KeyedBytes** instance_data, **InstanceHandle_t** handle)

  Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.
- void **write** (String key, byte[ ] octets, int offset, int length, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.*
- void **write** (String key, **ByteSeq** octets, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.*
- void **write_w_timestamp** (**KeyedBytes** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.write except that it also provides the value for the `source_timestamp`.
- void **write_w_timestamp** (String key, byte[ ] octets, int offset, int length, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.KeyedBytesData↩ Writer.write except that it also provides the value for the `source_timestamp`.
- void **write_w_timestamp** (String key, **ByteSeq** octets, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.KeyedBytesData↩ Writer.write except that it also provides the value for the `source_timestamp`.
- void **dispose** (**KeyedBytes** instance_data, **InstanceHandle_t** instance_handle)

  Requests the middleware to delete the data.
- void **dispose** (String key, **InstanceHandle_t** instance_handle)

  *<<**extension**>> (p. 155) Requests the middleware to delete the data.*
- void **dispose_w_timestamp** (**KeyedBytes** instance_data, **InstanceHandle_t** instance_handle, **Time_**↩ **t** source_timestamp)

Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.dispose except that the application provides the value for the `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 450) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1634).*

- void **dispose_w_timestamp** (String key, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.KeyedBytesDataWriter.dispose except that the application provides the value for the* `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 450) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1634).*
- void **get_key_value** (**KeyedBytes** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` *that corresponds to an instance* `handle`*.*
- String **get_key_value** (**InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Retrieve the instance* `key` *that corresponds to an instance* `handle`*.*
- **InstanceHandle_t lookup_instance** (**KeyedBytes** key_holder)

  Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*
- **InstanceHandle_t lookup_instance** (String key)

  *<<**extension**>> (p. 155) Retrieve the instance* `handle` *that corresponds to an instance* `key`*.*

8.147.1 Detailed Description

*<<**interface**>> (p. 156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.publication.DataWriter (p. 553)

8.147.2 Member Function Documentation

8.147.2.1 register_instance() [1/2]

```
InstanceHandle_t register_instance (
            KeyedBytes instance_data )
```

Informs RTI Connext that the application will be modifying a particular instance.

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

References **DataWriter.register_instance_untyped()**.

8.147.2.2 register_instance() [2/2]

```
InstanceHandle_t register_instance (
            String key )
```

<<***extension***>> *(p. 155)* Informs RTI Connext that the application will be modifying a particular instance.

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

References **KeyedBytes.key**, and **DataWriter.register_instance_untyped()**.

8.147.2.3 register_instance_w_timestamp() [1/2]

```
InstanceHandle_t register_instance_w_timestamp (
             KeyedBytes instance_data,
             Time_t source_timestamp )
```

Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance except that the application provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp (p. 1099)

References **DataWriter.register_instance_w_timestamp_untyped()**.

8.147.2.4 register_instance_w_timestamp() [2/2]

```
InstanceHandle_t register_instance_w_timestamp (
            String key,
             Time_t source_timestamp )
```

<<***extension***>> *(p. 155)* Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ BytesDataWriter.KeyedBytesDataWriter.register_instance except that the application provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp (p. 1099)

References **KeyedBytes.key**, and **DataWriter.register_instance_w_timestamp_untyped()**.

8.147.2.5 unregister_instance() [1/2]

```
void unregister_instance (
             KeyedBytes instance_data,
             InstanceHandle_t handle )
```

Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance.

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

References **DataWriter.unregister_instance_untyped()**.

8.147.2.6 unregister_instance() [2/2]

```
void unregister_instance (
            String key,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesData↩Writer.KeyedBytesDataWriter.register_instance.

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

References **KeyedBytes.key**, and **DataWriter.unregister_instance_untyped()**.

8.147.2.7 unregister_instance_w_timestamp() [1/2]

```
void unregister_instance_w_timestamp (
             KeyedBytes instance_data,
             InstanceHandle_t handle,
             Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.unregister_↩instance except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

References **DataWriter.unregister_instance_w_timestamp_untyped()**.

8.147.2.8 unregister_instance_w_timestamp() [2/2]

```
void unregister_instance_w_timestamp (
            String key,
             InstanceHandle_t handle,
             Time_t source_timestamp )
```

<<***extension***>> *(p. 155)* Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes↩ DataWriter.KeyedBytesDataWriter.unregister_instance except that it also provides the value for the `source_↩ timestamp`.

See also

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

References **KeyedBytes.key**, and **DataWriter.unregister_instance_w_timestamp_untyped()**.

8.147.2.9 write() [1/3]

```
void write (
             KeyedBytes instance_data,
             InstanceHandle_t handle )
```

Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **DataWriter.write_untyped()**.

8.147.2.10 write() [2/3]

```
void write (
            String key,
            byte[] octets,
            int offset,
            int length,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.

Parameters

key	<<***in***>> *(p. 156)* Instance key.
octets	<<***in***>> *(p. 156)* Array of bytes to be published.
offset	<<***in***>> *(p. 156)* Offset from which to start publishing.
length	<<***in***>> *(p. 156)* Number of bytes to be published.
handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance, or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). See **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **KeyedBytes.key**, **KeyedBytes.length**, **KeyedBytes.offset**, **KeyedBytes.value**, and **DataWriter.write_untyped()**.

8.147.2.11 write() [3/3]

```
void write (
             String key,
              ByteSeq octets,
              InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data instance.

Parameters

key	<<***in***>> *(p. 156)* Instance key.
octets	<<***in***>> *(p. 156)* Sequence of bytes to be published.
handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance, or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). See **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **KeyedBytes.key**, **KeyedBytes.length**, **KeyedBytes.offset**, **AbstractPrimitiveSequence.size()**, **KeyedBytes.value**, and **DataWriter.write_untyped()**.

8.147.2.12 write_w_timestamp() [1/3]

```
void write_w_timestamp (
              KeyedBytes instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.write except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **DataWriter.write_w_timestamp_untyped()**.

8.147.2.13 write_w_timestamp() [2/3]

```
void write_w_timestamp (
             String key,
             byte[] octets,
             int offset,
             int length,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.KeyedBytes↩ DataWriter.write except that it also provides the value for the `source_timestamp`.

Parameters

key	<<***in***>> *(p. 156)* Instance key.
octets	<<***in***>> *(p. 156)* Array of bytes to be published.
offset	<<***in***>> *(p. 156)* Offset from which to start publishing.
length	<<***in***>> *(p. 156)* Number of bytes to be published.
handle	<<***in***>> *(p. 156)* Either the handle returned by a previous call to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance, or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). See **com.rti.ndds.example.FooDataWriter.write** (p. 1105).
source_timestamp	<<***in***>> *(p. 156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109). Cannot be NULL.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **KeyedBytes.key**, **KeyedBytes.length**, **KeyedBytes.offset**, **KeyedBytes.value**, and **DataWriter.write_w_timestamp_untyped()**.

8.147.2.14 write_w_timestamp() [3/3]

```
void write_w_timestamp (
            String key,
             ByteSeq octets,
             InstanceHandle_t handle,
             Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.KeyedBytesDataWriter.write except that it also provides the value for the `source_timestamp`.

Parameters

key	<<***in***>> *(*p. *156)* Instance key.
octets	<<***in***>> *(*p. *156)* Sequence of bytes to be published.
handle	<<***in***>> *(*p. *156)* Either the handle returned by a previous call to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance, or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156). See **com.rti.ndds.example.FooDataWriter.write** (p. 1105).
source_timestamp	<<***in***>> *(*p. *156)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109). Cannot be NULL.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **KeyedBytes.key**, **KeyedBytes.length**, **KeyedBytes.offset**, **AbstractPrimitiveSequence.size()**, **KeyedBytes.value**, and **DataWriter.write_w_timestamp_untyped()**.

8.147.2.15 dispose() [1/2]

```
void dispose (
             KeyedBytes instance_data,
             InstanceHandle_t instance_handle )
```

Requests the middleware to delete the data.

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **DataWriter.dispose_untyped()**.

8.147.2.16 dispose() [2/2]

```
void dispose (
            String key,
             InstanceHandle_t instance_handle )
```

<<***extension***>> *(*p. *155)* Requests the middleware to delete the data.

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **DataWriter.dispose_untyped()**, and **KeyedBytes.key**.

8.147.2.17 dispose_w_timestamp() [1/2]

```
void dispose_w_timestamp (
             KeyedBytes instance_data,
             InstanceHandle_t instance_handle,
             Time_t source_timestamp )
```

Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesDataWriter.dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.↩ subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.↩ subscription.SampleInfo** (p. 1634).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

References **DataWriter.dispose_w_timestamp_untyped()**.

8.147.2.18 dispose_w_timestamp() [2/2]

```
void dispose_w_timestamp (
            String key,
             InstanceHandle_t instance_handle,
             Time_t source_timestamp )
```

<<***extension***>> *(*p. *155)* Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ BytesDataWriter.KeyedBytesDataWriter.dispose except that the application provides the value for the `source↩ _timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

References **DataWriter.dispose_w_timestamp_untyped()**, and **KeyedBytes.key**.

8.147.2.19 get_key_value() [1/2]

```
void get_key_value (
              KeyedBytes key_holder,
              InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataWriter.get_key_value_untyped()**.

8.147.2.20 get_key_value() [2/2]

```
String get_key_value (
              InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataWriter.get_key_value_untyped()**, and **KeyedBytes.key**.

8.147.2.21 lookup_instance() [1/2]

```
InstanceHandle_t lookup_instance (
              KeyedBytes key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

com.rti.ndds.example.FooDataWriter.lookup_instance (p. 1115)

References **DataWriter.lookup_instance_untyped()**.

8.147.2.22 lookup_instance() [2/2]

```
InstanceHandle_t lookup_instance (
            String key )
```

<<***extension***>> *(*p. *155)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

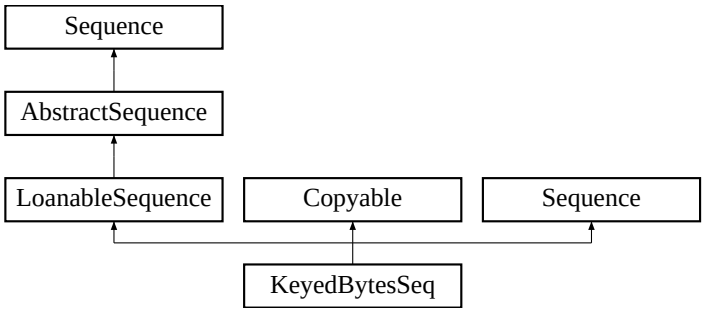
com.rti.ndds.example.FooDataWriter.lookup_instance (p. 1115)

References **KeyedBytes.key**, and **DataWriter.lookup_instance_untyped()**.

8.148 KeyedBytesSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.KeyedBytes >.

Inheritance diagram for KeyedBytesSeq:



Public Member Functions

- **KeyedBytesSeq** ()

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects with an initial maximum of zero.
- **KeyedBytesSeq** (int initialMaximum)

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects with the given initial maximum.
- **KeyedBytesSeq** (Collection elements)

  Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects.
- **KeyedBytes get** (int index)

  Returns the element at the specified position in this sequence.
- Object **copy_from** (Object src)

8.148.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes >.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes

8.148.2 Constructor & Destructor Documentation

8.148.2.1 KeyedBytesSeq() [1/3]

```
KeyedBytesSeq ( )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects with an initial maximum of zero.

8.148.2.2 KeyedBytesSeq() [2/3]

```
KeyedBytesSeq (
            int initialMaximum )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects with the given initial maximum.

8.148.2.3 KeyedBytesSeq() [3/3]

```
KeyedBytesSeq (
            Collection elements )
```

Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes objects.

Parameters

elements	the initial contents of this sequence.

Exceptions

NullPointerException	if the input collection is null

8.148.3 Member Function Documentation

8.148.3.1 get()

```
KeyedBytes get (
            int index )
```

Returns the element at the specified position in this sequence.

See also

java.util.List::get(int)

Reimplemented from **LoanableSequence** (p. 1252).

8.148.3.2 copy_from()

```
Object copy_from (
            Object src )
```

Copy data into `this` object from another. The result of this method is that both `this` and `src` will be the same size and contain the same data.

Parameters

src	The Object which contains the data to be copied

Returns

`this`

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not a `Sequence` OR if one of the objects contained in the `Sequence` is not of the expected type.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

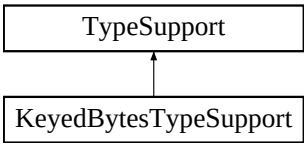
References **AbstractSequence.add()**, **KeyedBytesSeq.copy_from()**, **LoanableSequence.getMaximum()**, **LoanableSequence.setMaximum()**, and **LoanableSequence.size()**.

Referenced by **KeyedBytesSeq.copy_from()**.

8.149 KeyedBytesTypeSupport Class Reference

<<***interface***>> *(*p. *156)* com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type support.

Inheritance diagram for KeyedBytesTypeSupport:



Public Member Functions

- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **KeyedBytes** src)

  <<***extension***>> *(*p. *155) Serializes the input sample into a CDR buffer of octets.*
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **KeyedBytes** src, short representation)

  <<***extension***>> (p. *155) Serializes the input sample into a buffer of octets.*
- String **data_to_string** (**KeyedBytes** src, **PrintFormatProperty** property)

  <<***extension***>> *(*p. *155) Get the string representation of an input sample.*
- String **data_to_string** (**KeyedBytes** src)

  <<***extension***>> (p. *155) Get the string representation of an input sample.*
- void **deserialize_from_cdr_buffer** (**KeyedBytes** dst, byte[ ] buffer, long length)

  <<***extension***>> (p. *155) Deserializes a sample from a buffer of octets.*

Static Public Member Functions

- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data type.
- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.
- static String **get_type_name** ()

  Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type.

8.149.1 Detailed Description

<<***interface***>> *(*p. *156)* com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type support.

8.149.2 Member Function Documentation

8.149.2.1 register_type()

```
static void register_type (
               DomainParticipant participant,
              String type_name )  [static]
```

Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data type.

By default, The com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes built-in type is automatically registered when a DomainParticipant is created using the type_name returned by com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesTypeSupport.get_type_name. Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesTypeSupport with a **com.rti.dds.domain.DomainParticipant** (p. 670) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 670) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to register the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.Bytes with. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1807). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706).) The name may not be null or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.149.2.2 unregister_type()

```
static void unregister_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1807) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **com.rti.dds.topic.Topic** (p. 1807) is associated with the type, the operation will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to unregister the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes from. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) or **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595)

MT Safety:

SAFE.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type

8.149.2.3 get_type_name()

```
static String get_type_name ( )  [static]
```

Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type.

Can be used for calling com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type or creating **com.rti.dds.topic.Topic** (p. 1807).

Returns

default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytes type.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.149.2.4 serialize_to_cdr_buffer() [1/2]

```
long serialize_to_cdr_buffer (
             byte[] buffer,
             long length,
              KeyedBytes src )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a CDR buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

Referenced by **KeyedBytesTypeSupport.data_to_string()**.

8.149.2.5 serialize_to_cdr_buffer() [2/2]

```
long serialize_to_cdr_buffer (
            byte[] buffer,
            long length,
             KeyedBytes src,
            short representation )
```

<<***extension***>> *(*p. *155)* Serializes the input sample into a buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

8.149.2.6 data_to_string() [1/2]

```
String data_to_string (
             KeyedBytes src,
             PrintFormatProperty property )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **DynamicData.from_cdr_buffer()**, **DynamicData.PROPERTY_DEFAULT**, **KeyedBytesTypeSupport.↩ serialize_to_cdr_buffer()**, and **DynamicData.to_string()**.

Referenced by **KeyedBytesTypeSupport.data_to_string()**.

8.149.2.7 data_to_string() [2/2]

```
String data_to_string (
             KeyedBytes src )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

Use the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387) to create the output string.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **KeyedBytesTypeSupport.data_to_string()**.

8.149.2.8 deserialize_from_cdr_buffer()

```
void deserialize_from_cdr_buffer (
              KeyedBytes dst,
             byte[] buffer,
             long length )
```

<<***extension***>> *(*p. *155)* Deserializes a sample from a buffer of octets.

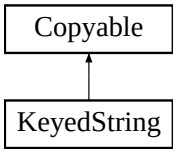
See also

com.rti.ndds.example.FooTypeSupport.deserialize_from_cdr_buffer (p. 1123)

8.150 KeyedString Class Reference

Keyed string built-in type.

Inheritance diagram for KeyedString:



Public Member Functions

- **KeyedString** ()
  Default Constructor.
- **KeyedString** (String **key**, String **value**)
- **KeyedString** (**KeyedString** src)
  Copy constructor.
- Object **copy_from** (Object src)
  Copy value of a data type from source.

Public Attributes

- String **key**
  Instance key associated with the specified value.
- String **value**
  String value.

8.150.1 Detailed Description

Keyed string built-in type.

8.150.2 Constructor & Destructor Documentation

8.150.2.1 KeyedString() [1/3]

```
KeyedString ( )
```

Default Constructor.

The default constructor initializes the newly created object with empty key and value.

Referenced by **KeyedString.copy_from()**.

8.150.2.2 KeyedString() [2/3]

```
KeyedString (
            String key,
            String value )
```

Creates a **KeyedString** (p. 1203) with a key and a value

References **KeyedString.key**, and **KeyedString.value**.

8.150.2.3 KeyedString() [3/3]

```
KeyedString (
             KeyedString src )
```

Copy constructor.

Parameters

src	<<***in***>> *(p. 156)* Object to copy from.

Exceptions

NullPointerException	if src is null.

References **KeyedString.copy_from()**.

8.150.3 Member Function Documentation

8.150.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

References **KeyedString.key**, **KeyedString.KeyedString()**, and **KeyedString.value**.

Referenced by **KeyedString.KeyedString()**.

8.150.4 Member Data Documentation

8.150.4.1 key

```
String key
```

Instance key associated with the specified value.

Referenced by **KeyedString.copy_from()**, **KeyedStringDataWriter.dispose()**, **KeyedStringDataWriter.dispose_w_timestamp()**, **KeyedStringDataReader.get_key_value()**, **KeyedStringDataWriter.get_key_value()**, **KeyedString.KeyedString()**, **KeyedStringDataReader.lookup_instance()**, **KeyedStringDataWriter.lookup_instance()**, **KeyedStringDataWriter.register_instance()**, **KeyedStringDataWriter.register_instance_w_timestamp()**, **KeyedStringDataWriter.unregister_instance()**, **KeyedStringDataWriter.unregister_instance_w_timestamp()**, **KeyedStringDataWriter.write()**, and **KeyedStringDataWriter.write_w_timestamp()**.

8.150.4.2 value

```
String value
```

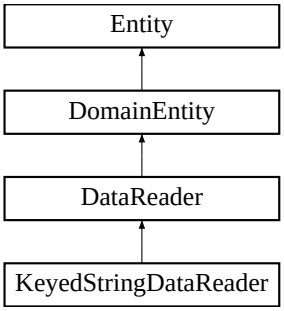
String value.

Referenced by **KeyedString.copy_from()**, **KeyedString.KeyedString()**, **KeyedStringDataWriter.write()**, and **KeyedStringDataWriter.write_w_timestamp()**.

8.151 KeyedStringDataReader Class Reference

<<***interface***>> *(*p. *156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.

Inheritance diagram for KeyedStringDataReader:



Public Member Functions

- void **read** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **take** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

  *Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (*p. *450).*
- void **read_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (*p. *1514).*

- void **take_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take operation.
- void **read_next_sample** (**KeyedString** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_next_sample** (**KeyedString** received_data, **SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_instance** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_instance** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_instance_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_instance_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **read_next_instance** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take_next_instance** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, int sample_states, int view_states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_next_instance_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_next_instance_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **return_loan** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq)

  *Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **get_key_value** (**KeyedString** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.
- String **get_key_value** (**InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Retrieve the instance `key` that corresponds to an instance `handle`.*
- **InstanceHandle_t lookup_instance** (**KeyedString** key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.
- **InstanceHandle_t lookup_instance** (String key)

  *<<**extension**>> (p. 155) Retrieve the instance `handle` that corresponds to an instance `key`.*

Additional Inherited Members

8.151.1 Detailed Description

<<***interface***>> *(*p. *156)* Instantiates `DataReader` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.

See also

com.rti.ndds.example.FooDataReader (p. 1067)
com.rti.dds.subscription.DataReader (p. 450)

8.151.2 Member Function Documentation

8.151.2.1 read()

```
void read (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_untyped()**.

8.151.2.2 take()

```
void take (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_untyped()**.

8.151.2.3 read_w_condition()

```
void read_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076)

References **DataReader.read_w_condition_untyped()**.

8.151.2.4 take_w_condition()

```
void take_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_w_condition except it accesses samples via the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take operation.

See also

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

References **DataReader.take_w_condition_untyped()**.

8.151.2.5 read_next_sample()

```
void read_next_sample (
              KeyedString received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_sample (p. 1078)

References **DataReader.read_next_sample_untyped()**.

8.151.2.6 take_next_sample()

```
void take_next_sample (
              KeyedString received_data,
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_next_sample (p. 1079)

References **DataReader.take_next_sample_untyped()**.

8.151.2.7 read_instance()

```
void read_instance (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_instance (p. 1081)

References **DataReader.read_instance_untyped()**.

8.151.2.8 take_instance()

```
void take_instance (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_instance (p. 1082)

References **DataReader.take_instance_untyped()**.

8.151.2.9 read_instance_w_condition()

```
void read_instance_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_instance_w_condition (p. 1088)

References **DataReader.read_instance_w_condition_untyped()**.

8.151.2.10 take_instance_w_condition()

```
void take_instance_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.take_instance_w_condition (p. 1089)

References **DataReader.take_instance_w_condition_untyped()**.

8.151.2.11 read_next_instance()

```
void read_next_instance (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_instance (p. 1084)

References **DataReader.read_next_instance_untyped()**.

8.151.2.12 take_next_instance()

```
void take_next_instance (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take_next_instance (p. 1086)

References **DataReader.take_next_instance_untyped()**.

8.151.2.13 read_next_instance_w_condition()

```
void read_next_instance_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_next_instance_w_condition (p. 1091)

References **DataReader.read_next_instance_w_condition_untyped()**.

8.151.2.14 take_next_instance_w_condition()

```
void take_next_instance_w_condition (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              InstanceHandle_t a_handle,
              ReadCondition condition )
```

Accesses via com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.take_next_instance_w_condition (p. 1092)

References **DataReader.take_next_instance_w_condition_untyped()**.

8.151.2.15 return_loan()

```
void return_loan (
              KeyedStringSeq received_data,
              SampleInfoSeq info_seq )
```

Indicates to the **com.rti.dds.subscription.DataReader** (p. 450) that the application is done accessing the collection of `received_data` and `info_seq` obtained by some earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.return_loan (p. 1094)

References **DataReader.return_loan_untyped()**.

8.151.2.16 get_key_value() [1/2]

```
void get_key_value (
              KeyedString key_holder,
              InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

References **DataReader.get_key_value_untyped()**.

8.151.2.17 get_key_value() [2/2]

```
String get_key_value (
              InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataReader.get_key_value (p. 1095)

References **DataReader.get_key_value_untyped()**, and **KeyedString.key**.

8.151.2.18 lookup_instance() [1/2]

```
InstanceHandle_t lookup_instance (
            KeyedString key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

com.rti.ndds.example.FooDataReader.lookup_instance (p. 1096)

References **DataReader.lookup_instance_untyped()**.

8.151.2.19 lookup_instance() [2/2]

```
InstanceHandle_t lookup_instance (
           String key )
```

<<***extension***>> *(p. 155)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

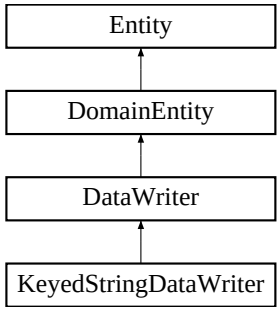
com.rti.ndds.example.FooDataReader.lookup_instance (p. 1096)

References **KeyedString.key**, and **DataReader.lookup_instance_untyped()**.

8.152 KeyedStringDataWriter Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.

Inheritance diagram for KeyedStringDataWriter:



Public Member Functions

- **InstanceHandle_t register_instance** (**KeyedString** instance_data)

  Informs RTI Connext that the application will be modifying a particular instance.
- **InstanceHandle_t register_instance** (String key)

  *<<**extension**>> (p. 155) Informs RTI Connext that the application will be modifying a particular instance.*
- **InstanceHandle_t register_instance_w_timestamp** (**KeyedString** instance_data, **Time_t** source_timestamp)

  Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance except that the application provides the value for the `source_timestamp`.
- **InstanceHandle_t register_instance_w_timestamp** (String key, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.KeyedStringDataWriter.register_instance except that the application provides the value for the `source_timestamp`.*
- void **unregister_instance** (**KeyedString** instance_data, **InstanceHandle_t** handle)

  Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance.
- void **unregister_instance** (String key, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.KeyedStringDataWriter.register_instance.*
- void **unregister_instance_w_timestamp** (**KeyedString** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.unregister_instance except that it also provides the value for the `source_timestamp`.
- void **unregister_instance_w_timestamp** (String key, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.KeyedStringDataWriter.unregister_instance except that it also provides the value for the `source_timestamp`.*
- void **write** (**KeyedString** instance_data, **InstanceHandle_t** handle)

  Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data instance.
- void **write** (String key, String str, **InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data instance.*
- void **write_w_timestamp** (**KeyedString** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.write except that it also provides the value for the `source_timestamp`.
- void **write_w_timestamp** (String key, String str, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *<<**extension**>> (p. 155) Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.KeyedStringDataWriter.write except that it also provides the value for the `source_timestamp`.*
- void **dispose** (**KeyedString** instance_data, **InstanceHandle_t** instance_handle)

  Requests the middleware to delete the data.
- void **dispose** (String key, **InstanceHandle_t** instance_handle)

  *<<**extension**>> (p. 155) Requests the middleware to delete the data.*
- void **dispose_w_timestamp** (**KeyedString** instance_data, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

  *Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).*
- void **dispose_w_timestamp** (String key, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

*<<**extension**>> (p. 155) Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString↩ DataWriter.KeyedStringDataWriter.dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).*

- void **get_key_value** (**KeyedString** key_holder, **InstanceHandle_t** handle)

  Retrieve the instance `key` that corresponds to an instance `handle`.

- String **get_key_value** (**InstanceHandle_t** handle)

  *<<**extension**>> (p. 155) Retrieve the instance `key` that corresponds to an instance `handle`.*

- **InstanceHandle_t lookup_instance** (**KeyedString** key_holder)

  Retrieve the instance `handle` that corresponds to an instance `key_holder`.

- **InstanceHandle_t lookup_instance** (String key)

  *<<**extension**>> (p. 155) Retrieve the instance `handle` that corresponds to an instance `key`.*

8.152.1 Detailed Description

*<<**interface**>> (p. 156)* Instantiates `DataWriter` < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString >.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.publication.DataWriter (p. 553)

8.152.2 Member Function Documentation

8.152.2.1 register_instance() [1/2]

```
InstanceHandle_t register_instance (
            KeyedString instance_data )
```

Informs RTI Connext that the application will be modifying a particular instance.

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

References **DataWriter.register_instance_untyped()**.

8.152.2.2 register_instance() [2/2]

```
InstanceHandle_t register_instance (
            String key )
```

<<***extension***>> *(p. 155)* Informs RTI Connext that the application will be modifying a particular instance.

See also

com.rti.ndds.example.FooDataWriter.register_instance (p. 1098)

References **KeyedString.key**, and **DataWriter.register_instance_untyped()**.

8.152.2.3 register_instance_w_timestamp() [1/2]

```
InstanceHandle_t register_instance_w_timestamp (
             KeyedString instance_data,
             Time_t source_timestamp )
```

Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance except that the application provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp (p. 1099)

References **DataWriter.register_instance_w_timestamp_untyped()**.

8.152.2.4 register_instance_w_timestamp() [2/2]

```
InstanceHandle_t register_instance_w_timestamp (
            String key,
             Time_t source_timestamp )
```

<<***extension***>> *(p. 155)* Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ StringDataWriter.KeyedStringDataWriter.register_instance except that the application provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.register_instance_w_timestamp (p. 1099)

References **KeyedString.key**, and **DataWriter.register_instance_w_timestamp_untyped()**.

8.152.2.5 unregister_instance() [1/2]

```
void unregister_instance (
             KeyedString instance_data,
             InstanceHandle_t handle )
```

Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance.

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

References **DataWriter.unregister_instance_untyped()**.

8.152.2.6 unregister_instance() [2/2]

```
void unregister_instance (
            String key,
             InstanceHandle_t handle )
```

<<***extension***>> *(p. 155)* Reverses the action of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringData↩ Writer.KeyedStringDataWriter.register_instance.

See also

com.rti.ndds.example.FooDataWriter.unregister_instance (p. 1100)

References **KeyedString.key**, and **DataWriter.unregister_instance_untyped()**.

8.152.2.7 unregister_instance_w_timestamp() [1/2]

```
void unregister_instance_w_timestamp (
             KeyedString instance_data,
             InstanceHandle_t handle,
             Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.unregister_↩ instance except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

References **DataWriter.unregister_instance_w_timestamp_untyped()**.

8.152.2.8 unregister_instance_w_timestamp() [2/2]

```
void unregister_instance_w_timestamp (
            String key,
             InstanceHandle_t handle,
             Time_t source_timestamp )
```

<<***extension***>> *(*p. *155)* Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString↩ DataWriter.KeyedStringDataWriter.unregister_instance except that it also provides the value for the `source_↩ timestamp`.

See also

com.rti.ndds.example.FooDataWriter.FooDataWriter.unregister_instance_w_timestamp

References **KeyedString.key**, and **DataWriter.unregister_instance_w_timestamp_untyped()**.

8.152.2.9 write() [1/2]

```
void write (
             KeyedString instance_data,
             InstanceHandle_t handle )
```

Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data instance.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **DataWriter.write_untyped()**.

8.152.2.10 write() [2/2]

```
void write (
            String key,
            String str,
             InstanceHandle_t handle )
```

<<***extension***>> *(*p. *155)* Modifies the value of a com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data instance.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **KeyedString.key**, **KeyedString.value**, and **DataWriter.write_untyped()**.

8.152.2.11 write_w_timestamp() [1/2]

```
void write_w_timestamp (
              KeyedString instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.write except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **DataWriter.write_w_timestamp_untyped()**.

8.152.2.12 write_w_timestamp() [2/2]

```
void write_w_timestamp (
             String key,
             String str,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

<<***extension***>> *(p. 155)* Performs the same function as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString↩ DataWriter.KeyedStringDataWriter.write except that it also provides the value for the `source_timestamp`.

See also

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **KeyedString.key**, **KeyedString.value**, and **DataWriter.write_w_timestamp_untyped()**.

8.152.2.13 dispose() [1/2]

```
void dispose (
              KeyedString instance_data,
              InstanceHandle_t instance_handle )
```

Requests the middleware to delete the data.

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **DataWriter.dispose_untyped()**.

8.152.2.14 dispose() [2/2]

```
void dispose (
            String key,
             InstanceHandle_t instance_handle )
```

<<***extension***>> *(*p. *155)* Requests the middleware to delete the data.

See also

com.rti.ndds.example.FooDataWriter.dispose (p. 1111)

References **DataWriter.dispose_untyped()**, and **KeyedString.key**.

8.152.2.15 dispose_w_timestamp() [1/2]

```
void dispose_w_timestamp (
             KeyedString instance_data,
             InstanceHandle_t instance_handle,
             Time_t source_timestamp )
```

Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringDataWriter.dispose except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.↩ subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.↩ subscription.SampleInfo** (p. 1634).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

References **DataWriter.dispose_w_timestamp_untyped()**.

8.152.2.16 dispose_w_timestamp() [2/2]

```
void dispose_w_timestamp (
            String key,
             InstanceHandle_t instance_handle,
             Time_t source_timestamp )
```

<<***extension***>> *(*p. *155)* Performs the same functions as com.rti.dds.type.builtin.com.rti.dds.type.builtin.Keyed↩ StringDataWriter.KeyedStringDataWriter.dispose except that the application provides the value for the `source↩ _timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1634).

See also

com.rti.ndds.example.FooDataWriter.dispose_w_timestamp (p. 1113)

References **DataWriter.dispose_w_timestamp_untyped()**, and **KeyedString.key**.

8.152.2.17 get_key_value() [1/2]

```
void get_key_value (
              KeyedString key_holder,
              InstanceHandle_t handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataWriter.get_key_value_untyped()**.

8.152.2.18 get_key_value() [2/2]

```
String get_key_value (
              InstanceHandle_t handle )
```

<<***extension***>> *(*p. *155)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also

com.rti.ndds.example.FooDataWriter.get_key_value (p. 1114)

References **DataWriter.get_key_value_untyped()**, and **KeyedString.key**.

8.152.2.19 lookup_instance() [1/2]

```
InstanceHandle_t lookup_instance (
              KeyedString key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

com.rti.ndds.example.FooDataWriter.lookup_instance (p. 1115)

References **DataWriter.lookup_instance_untyped()**.

8.152.2.20 lookup_instance() [2/2]

```
InstanceHandle_t lookup_instance (
            String key )
```

<<***extension***>> *(*p. *155)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also

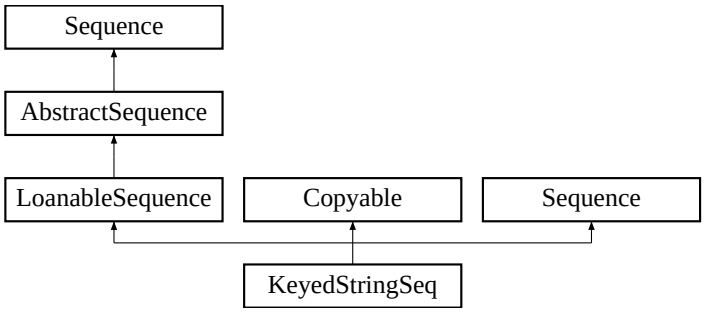
com.rti.ndds.example.FooDataWriter.lookup_instance (p. 1115)

References **KeyedString.key**, and **DataWriter.lookup_instance_untyped()**.

8.153 KeyedStringSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.type.↩ builtin.com.rti.dds.type.builtin.KeyedString > .

Inheritance diagram for KeyedStringSeq:



Public Member Functions

- **KeyedStringSeq** ()

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects with an initial maximum of zero.
- **KeyedStringSeq** (int initialMaximum)

  Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects with the given initial maximum.
- **KeyedStringSeq** (Collection elements)

  Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects.
- **KeyedString get** (int index)

  Returns the element at the specified position in this sequence.
- Object **copy_from** (Object src)

8.153.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString

8.153.2 Constructor & Destructor Documentation

8.153.2.1 KeyedStringSeq() [1/3]

```
KeyedStringSeq ( )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects with an initial maximum of zero.

8.153.2.2 KeyedStringSeq() [2/3]

```
KeyedStringSeq (
            int initialMaximum )
```

Constructs an empty sequence of com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects with the given initial maximum.

8.153.2.3 KeyedStringSeq() [3/3]

```
KeyedStringSeq (
            Collection elements )
```

Constructs a new sequence containing the given com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString objects.

Parameters

elements	the initial contents of this sequence.

Exceptions

NullPointerException	if the input collection is null

8.153.3 Member Function Documentation

8.153.3.1 get()

```
KeyedString get (
            int index )
```

Returns the element at the specified position in this sequence.

See also

java.util.List::get(int)

Reimplemented from **LoanableSequence** (p. 1252).

8.153.3.2 copy_from()

```
Object copy_from (
            Object src )
```

Copy data into `this` object from another. The result of this method is that both `this` and `src` will be the same size and contain the same data.

Parameters

src	The Object which contains the data to be copied

Returns

`this`

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not a `Sequence` OR if one of the objects contained in the `Sequence` is not of the expected type.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

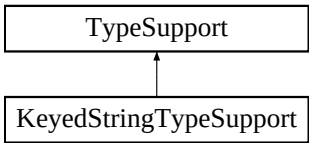
References **AbstractSequence.add()**, **KeyedStringSeq.copy_from()**, **LoanableSequence.getMaximum()**, **LoanableSequence.setMaximum()**, and **LoanableSequence.size()**.

Referenced by **KeyedStringSeq.copy_from()**.

8.154 KeyedStringTypeSupport Class Reference

<<***interface***>> *(*p. *156)* Keyed string type support.

Inheritance diagram for KeyedStringTypeSupport:



Public Member Functions

- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **KeyedString** src)

  <<***extension***>> *(*p. *155) Serializes the input sample into a CDR buffer of octets.*
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, **KeyedString** src, short representation)

  <<***extension***>> (p. *155) Serializes the input sample into a buffer of octets.*
- String **data_to_string** (**KeyedString** src, **PrintFormatProperty** property)

  <<***extension***>> *(*p. *155) Get the string representation of an input sample.*
- String **data_to_string** (**KeyedString** src)

  <<***extension***>> (p. *155) Get the string representation of an input sample.*
- void **deserialize_from_cdr_buffer** (**KeyedString** dst, byte[ ] buffer, long length)

  <<***extension***>> (p. *155) Deserializes a sample from a buffer of octets.*

Static Public Member Functions

- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data type.
- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.
- static String **get_type_name** ()

  Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString type.

8.154.1 Detailed Description

<<***interface***>> *(*p. *156)* Keyed string type support.

8.154.2 Member Function Documentation

8.154.2.1 register_type()

```
static void register_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to communicate to RTI Connext the existence of the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data type.

By default, The com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString built-in type is automatically registered when a DomainParticipant is created using the type_name returned by com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringTypeSupport.get_type_name. Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringTypeSupport with a **com.rti.dds.domain.DomainParticipant** (p. 670) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 670) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to register the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString with. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1807). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706).) The name may not be null or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.154.2.2 unregister_type()

```
static void unregister_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to unregister the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1807) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **com.rti.dds.topic.Topic** (p. 1807) is associated with the type, the operation will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to unregister the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString from. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) or **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595)

MT Safety:

SAFE.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type

8.154.2.3 get_type_name()

```
static String get_type_name ( )  [static]
```

Get the default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString type.

Can be used for calling com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type or creating **com.rti.dds.topic.Topic** (p. 1807).

Returns

default name for the com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedString type.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.154.2.4 serialize_to_cdr_buffer() [1/2]

```
long serialize_to_cdr_buffer (
             byte[] buffer,
             long length,
              KeyedString src )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a CDR buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

Referenced by **KeyedStringTypeSupport.data_to_string()**.

8.154.2.5 serialize_to_cdr_buffer() [2/2]

```
long serialize_to_cdr_buffer (
            byte[] buffer,
            long length,
             KeyedString src,
            short representation )
```

<<***extension***>> *(*p. *155)* Serializes the input sample into a buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

8.154.2.6 data_to_string() [1/2]

```
String data_to_string (
             KeyedString src,
             PrintFormatProperty property )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **DynamicData.from_cdr_buffer()**, **DynamicData.PROPERTY_DEFAULT**, **KeyedStringTypeSupport.↩ serialize_to_cdr_buffer()**, and **DynamicData.to_string()**.

Referenced by **KeyedStringTypeSupport.data_to_string()**.

8.154.2.7 data_to_string() [2/2]

```
String data_to_string (
             KeyedString src )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

Use the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387) to create the output string.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **KeyedStringTypeSupport.data_to_string()**.

8.154.2.8 deserialize_from_cdr_buffer()

```
void deserialize_from_cdr_buffer (
              KeyedString dst,
             byte[] buffer,
             long length )
```

<<***extension***>> *(*p. *155)* Deserializes a sample from a buffer of octets.

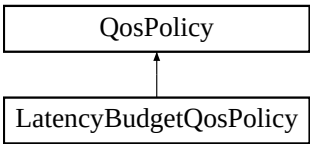
See also

com.rti.ndds.example.FooTypeSupport.deserialize_from_cdr_buffer (p. 1123)

8.155 LatencyBudgetQosPolicy Class Reference

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Inheritance diagram for LatencyBudgetQosPolicy:



Public Attributes

- final **Duration_t duration**

  Duration of the maximum acceptable delay.

8.155.1 Detailed Description

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

This policy is a *hint* to a DDS implementation; it can be used to change how it processes and sends data that has low latency requirements. The DDS specification does not mandate whether or how this policy is used.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **YES** (p. 256)

See also

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

com.rti.dds.publication.FlowController (p. 1055)

8.155.2 Usage

This policy provides a means for the application to indicate to the middleware the urgency of the data communication. By having a non-zero `duration`, RTI Connext can optimize its internal operation.

RTI Connext uses it in conjunction with com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQos↩ PolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 553) instances associated with a com.rti.dds.publication.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy.EDF_FLOW↩ _CONTROLLER_SCHED_POLICY **com.rti.dds.publication.FlowController** (p. 1055) only. Together with the time of write, **com.rti.dds.infrastructure.LatencyBudgetQosPolicy.duration** (p. 1232) determines the deadline of each individual sample. RTI Connext uses this information to prioritize the sending of asynchronously published data; see **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy** (p. 339).

8.155.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* <= *requested duration* evaluates to 'TRUE'.

8.155.4 Member Data Documentation

8.155.4.1 duration

```
final Duration_t duration
```

Duration of the maximum acceptable delay.

[default] 0 (meaning minimize the delay)

8.156 LibraryVersion_t Class Reference

The version of a single library shipped as part of an RTI Connext distribution.

Public Attributes

- final int **major**

  The major version of a single RTI Connext library.
- final int **minor**

  The minor version of a single RTI Connext library.
- final int **release**

  The release letter of a single RTI Connext library.
- final int **build**

  The build number of a single RTI Connext library.

8.156.1 Detailed Description

The version of a single library shipped as part of an RTI Connext distribution.

RTI Connext is comprised of a number of separate libraries. Although RTI Connext as a whole has a version, the individual libraries each have their own versions as well. It may be necessary to check these individual library versions when seeking technical support.

8.156.2 Member Data Documentation

8.156.2.1 major

```
final int major
```

The major version of a single RTI Connext library.

8.156.2.2 minor

```
final int minor
```

The minor version of a single RTI Connext library.

8.156.2.3 release

```
final int release
```

The release letter of a single RTI Connext library.

8.156.2.4 build

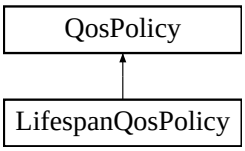
```
final int build
```

The build number of a single RTI Connext library.

8.157 LifespanQosPolicy Class Reference

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.

Inheritance diagram for LifespanQosPolicy:



Public Attributes

- final **Duration_t duration**

  Maximum duration for the data's validity.

8.157.1 Detailed Description

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 553) is considered valid.

Each data sample written by the **com.rti.dds.publication.DataWriter** (p. 553) has an associated expiration time beyond which the data should not be delivered to any application. Once the sample expires, the data will be removed from the **com.rti.dds.subscription.DataReader** (p. 450) caches as well as from the transient and persistent information caches.

The expiration time of each sample from the **com.rti.dds.publication.DataWriter** (p. 553)'s cache is computed by adding the duration specified by this QoS policy to the time when the sample is added to the **com.rti.dds.publication.↩ DataWriter** (p. 553)'s cache. This timestamp is not necessarily equal to the sample's source timestamp that can be provided by the user using the **com.rti.ndds.example.FooDataWriter.write_w_timestamp** (p. 1109) or **com.rti.ndds.↩ example.FooDataWriter.write_w_params** (p. 1110) API.

The expiration time of each sample from the **com.rti.dds.subscription.DataReader** (p. 450)'s cache is computed by adding the duration to the reception timestamp.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **YES** (p. 256)

8.157.2 Usage

The Lifespan QoS policy can be used to control how much data is stored by RTI Connext. Even if it is configured to store "all" of the data sent or received for a topic (see **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144)), the total amount of data it stores may be limited by this QoS policy.

You may also use this QoS policy to ensure that applications do not receive or act on data, commands or messages that are too old and have 'expired.'

To avoid inconsistencies, multiple writers of the same instance should have the same lifespan.

See also

com.rti.dds.subscription.SampleInfo.source_timestamp (p. 1640)

com.rti.dds.subscription.SampleInfo.reception_timestamp (p. 1643)

8.157.3 Member Data Documentation

8.157.3.1 duration

```
final Duration_t duration
```

Maximum duration for the data's validity.

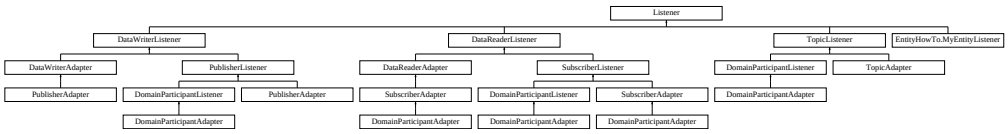
[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 nanosec, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.158 Listener Interface Reference

<<***interface***>> *(*p. *156)* Abstract base class for all **Listener** (p. 1236) interfaces.

Inheritance diagram for Listener:



8.158.1 Detailed Description

<<***interface***>> *(*p. *156)* Abstract base class for all **Listener** (p. 1236) interfaces.

Entity:

com.rti.dds.infrastructure.Entity (p. 1029)

QoS:

QoS Policies (p. 250)

Status:

Status Kinds (p. 262)

All the supported kinds of concrete **com.rti.dds.infrastructure.Listener** (p. 1236) interfaces (one per concrete **com.↩ rti.dds.infrastructure.Entity** (p. 1029) type) derive from this root and add methods whose prototype depends on the concrete **Listener** (p. 1236).

Listeners provide a way for RTI Connext to asynchronously alert the application when there are relevant status changes.

Almost every application will have to implement listener interfaces.

Each dedicated listener presents a list of operations that correspond to the relevant communication status changes to which an application may respond.

The same **com.rti.dds.infrastructure.Listener** (p. 1236) instance may be shared among multiple entities if you so desire. Consequently, the provided parameter contains a reference to the concerned **com.rti.dds.infrastructure.↩ Entity** (p. 1029).

8.158.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 262)) and the listeners' operations is as follows:

- For each communication status, there is a corresponding operation whose name is `on_<communication_status>`(), which takes a parameter of type `<communication_status>` as listed in **Status Kinds** (p. 262).
- `on_<communication_status>` is available on the relevant **com.rti.dds.infrastructure.Entity** (p. 1029) as well as those that embed it, as expressed in the following figure:

listener processing. The most *specific* relevant enabled listener is called."

- When the application attaches a listener on an entity, it must set a mask. The mask indicates to RTI Connext which operations are enabled within the listener (cf. operation **com.rti.dds.infrastructure.Entity** (p. 1029) set_listener()).
- When a plain communication status changes, RTI Connext triggers the most specific relevant listener operation that is enabled. In case the most specific relevant listener operation corresponds to an application-installed 'nil' listener the operation will be considered handled by a NO-OP operation that does not reset the communication status.

This behavior allows the application to set a default behavior (e.g., in the listener associated with the **com.rti.dds.domain.DomainParticipant** (p. 670)) and to set dedicated behaviors only where needed.

8.158.3 Access to Read Communication Status

The two statuses related to data arrival are treated slightly differently. Since they constitute the core purpose of the Data Distribution Service, there is no need to provide a default mechanism (as is done for the plain communication statuses above).

The rule is as follows. Each time the read communication status changes:

- First, RTI Connext tries to trigger the **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1756) with a parameter of the related **com.rti.dds.subscription.Subscriber** (p. 1730);
- If this does not succeed (there is no listener or the operation is not enabled), RTI Connext tries to trigger **com.rti.dds.subscription.DataReaderListener.on_data_available** (p. 500) on all the related **com.rti.dds.subscription.DataReaderListener** (p. 497) objects, with a parameter of the related **com.rti.dds.subscription.DataReader** (p. 450).

The rationale is that either the application is interested in relations among data arrivals and it must use the first option (and then get the corresponding **com.rti.dds.subscription.DataReader** (p. 450) objects by calling **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1741) on the related **com.rti.dds.subscription.Subscriber** (p. 1730) and then get the data by calling **com.rti.ndds.example.FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) on the returned **com.rti.dds.subscription.DataReader** (p. 450) objects), or it wants to treat each **com.rti.dds.subscription.DataReader** (p. 450) independently and it may choose the second option (and then get the data by calling **com.rti.ndds.example.FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) on the related **com.rti.dds.subscription.DataReader** (p. 450)).

Note that if **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1756) is called, RTI Connext will *not* try to call **com.rti.dds.subscription.DataReaderListener.on_data_available** (p. 500). However, an application can force a call to the **com.rti.dds.subscription.DataReader** (p. 450) objects that have data by calling **com.rti.dds.subscription.Subscriber.notify_datareaders** (p. 1742).

8.158.4 Operations Allowed in Listener Callbacks

See `Restricted Operations in Listener Callbacks`, in the Core Libraries User's Manual.

8.158.5 Best Practices with Listeners

Note that all the issues described below are avoided by using **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Avoid blocking or performing a lot of processing in Listener (p. 1236) callbacks

Listeners are invoked by internal threads that perform critical functions within the middleware and need to run in a timely manner. By default, Connext DDS creates a few threads to use to receive data and only a single thread to handle periodic events.

Because of this, user applications installing Listeners should never block in a **Listener** (p. 1236) callback. There are several negative consequences of blocking in a listener callback:

- The application may lose data for the DataReader the listener is installed on, because the receive thread is not removing it from the socket buffer and it gets overwritten.
- The application may receive strictly reliable data with a delay, because the receive thread is not removing it from the socket buffer and if it gets overwritten it must be re-sent.
- The application may lose or delay data for other DataReaders, because by default all DataReaders created with the same DomainParticipant share the same threads.
- The application may not be notified of periodic events on time

If the application needs to make a blocking call when data is available, or when another event occurs, the application should use **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Avoid taking application mutexes/semaphores in Listener (p. 1236) callbacks

Taking application mutexes/sempahores within a **Listener** (p. 1236) callback may lead to unexpected deadlock scenarios.

When a **Listener** (p. 1236) callback is invoked the EA (Exclusive Area) of the **Entity** (p. 1029) 'E' to which the callback applies is taken by the middleware.

If the application takes an application mutex 'M' within a critical section in which the application makes DDS calls affecting 'E', this may lead to following deadlock:

The middleware thread is within the entity EA trying to acquire the mutex 'M'. At the same time, the application thread has acquired 'M' and is blocked trying to acquire the entity EA.

Do not write data with a DataWriter within the on_data_available callback

Avoid writing data with a DataWriter within the **com.rti.dds.subscription.DataReaderListener::on_data_available()** (p. 500) callback. If the write operation blocks because e.g. the send window is full, this will lead to a deadlock.

Do not call wait_for_acknowledgements within the on_data_available callback

Do not call the **com.rti.dds.publication.DataWriter.wait_for_acknowledgments** (p. 570) within the **com.rti.dds.↩ subscription.DataReaderListener::on_data_available()** (p. 500) callback. This will lead to deadlock.

See also

Status Kinds (p. 262)

com.rti.dds.infrastructure.WaitSet (p. 1973), **com.rti.dds.infrastructure.Condition** (p. 429)

8.159 LivelinessChangedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS

Inherits Status.

Public Member Functions

- **LivelinessChangedStatus** ()

  The no-argument constructor for this status object.
- **LivelinessChangedStatus** (**LivelinessChangedStatus** src)

  A copy constructor.

Public Attributes

- int **alive_count**

  *The total count of currently alive **com.rti.dds.publication.DataWriter** (p. 553) entities that write the **com.rti.dds.topic.Topic** (p. 1807) that this **com.rti.dds.subscription.DataReader** (p. 450) reads.*
- int **not_alive_count**

  *The total count of currently not_alive **com.rti.dds.publication.DataWriter** (p. 553) entities that write the **com.rti.dds.topic.Topic** (p. 1807) that this **com.rti.dds.subscription.DataReader** (p. 450) reads.*
- int **alive_count_change**

  The change in the alive_count since the last time the listener was called or the status was read.
- int **not_alive_count_change**

  The change in the not_alive_count since the last time the listener was called or the status was read.
- final **InstanceHandle_t last_publication_handle**

  *This InstanceHandle can be used to look up which remote **com.rti.dds.publication.DataWriter** (p. 553) was the last to cause this **DataReader** (p. 450)'s status to change, using **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466).*

8.159.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS

The **com.rti.dds.subscription.DataReaderListener.on_liveliness_changed** (p. 499) callback may be invoked for the following reasons:

- The liveliness of any **com.rti.dds.publication.DataWriter** (p. 553) matching this **DataReader** (p. 450) (as defined by the **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247) setting) is lost.
- A DataWriter's liveliness is recovered after being lost.
- A new matching DataWriter has been discovered.
- A matching DataWriter has been deleted.

- A QoS Policy has changed such that a DataWriter that matched this **DataReader** (p. 450) before no longer matches (such as a change to the **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365)). In this case, RTI Connext will no longer keep track of the DataWriter's liveliness. Furthermore, consider two scenarios:
  - DataWriter was alive when it and **DataReader** (p. 450) stopped matching: **com.rti.dds.subscription.LivelinessChangedStatus.alive_count** (p. 1241) will decrease (since there's one less matching alive DataWriter) and **com.rti.dds.subscription.LivelinessChangedStatus.not_alive_count** (p. 1241) will remain the same (since the DataWriter is still alive).
  - DataWriter was not alive when it and **DataReader** (p. 450) stopped matching: **com.rti.dds.subscription.LivelinessChangedStatus.alive_count** (p. 1241) will remain the same (since the matching DataWriter was not alive) and **com.rti.dds.subscription.LivelinessChangedStatus.not_alive_count** (p. 1241) will decrease (since there's one less not-alive matching DataWriter).

    Note: There are several ways that a DataWriter and **DataReader** (p. 450) can become incompatible after the DataWriter has lost liveliness. For example, when the **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247) is set to DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, it is possible that the DataWriter has not asserted its liveliness in a timely manner, and then a QoS change occurs on the DataWriter or **DataReader** (p. 450) that makes the entities incompatible.
- A QoS Policy (such as the **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365)) has changed such that a DataWriter that was unmatched with the **DataReader** (p. 450) now matches.

8.159.2 Constructor & Destructor Documentation

8.159.2.1 LivelinessChangedStatus() [1/2]

```
LivelinessChangedStatus ( )
```

The no-argument constructor for this status object.

8.159.2.2 LivelinessChangedStatus() [2/2]

```
LivelinessChangedStatus (
            LivelinessChangedStatus src )
```

A copy constructor.

Exceptions

NullPointerException	if the given status is null.

References **LivelinessChangedStatus.alive_count**, **LivelinessChangedStatus.alive_count_change**, **LivelinessChangedStatus.last_publication_handle**, **LivelinessChangedStatus.not_alive_count**, and **LivelinessChangedStatus.not_alive_count_change**.

8.159.3 Member Data Documentation

8.159.3.1 alive_count

```
int alive_count
```

The total count of currently alive **com.rti.dds.publication.DataWriter** (p. 553) entities that write the **com.rti.dds.↩ topic.Topic** (p. 1807) that this **com.rti.dds.subscription.DataReader** (p. 450) reads.

Referenced by **LivelinessChangedStatus.LivelinessChangedStatus()**.

8.159.3.2 not_alive_count

```
int not_alive_count
```

The total count of currently not_alive **com.rti.dds.publication.DataWriter** (p. 553) entities that write the **com.rti.dds.↩ topic.Topic** (p. 1807) that this **com.rti.dds.subscription.DataReader** (p. 450) reads.

Referenced by **LivelinessChangedStatus.LivelinessChangedStatus()**.

8.159.3.3 alive_count_change

```
int alive_count_change
```

The change in the alive_count since the last time the listener was called or the status was read.

Referenced by **LivelinessChangedStatus.LivelinessChangedStatus()**.

8.159.3.4 not_alive_count_change

```
int not_alive_count_change
```

The change in the not_alive_count since the last time the listener was called or the status was read.

Note that a positive not_alive_count_change means one of the following:

- The DomainParticipant containing the matched DataWriter has lost liveliness or has been deleted.
- The matched DataWriter has lost liveliness or has been deleted.

Referenced by **LivelinessChangedStatus.LivelinessChangedStatus()**.

8.159.3.5 last_publication_handle

```
final InstanceHandle_t last_publication_handle
```

This InstanceHandle can be used to look up which remote **com.rti.dds.publication.DataWriter** (p. 553) was the last to cause this **DataReader** (p. 450)'s status to change, using **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466).

It's possible that the DataWriter has been purged from the discovery database. (See the "Discovery Overview" section of the `User's Manual`.) If so, the **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466) method will not be able to return information about the DataWriter. In this case, the only way to get information about the lost DataWriter is if you cached the information previously.

Referenced by **LivelinessChangedStatus.LivelinessChangedStatus()**.

8.160 LivelinessLostStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative number of times that a previously-alive **com.rti.dds.publication.DataWriter** (p. 553) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.*
- int **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.

8.160.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

The liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243) was not respected; thus **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer "alive/active".

8.160.2 Member Data Documentation

8.160.2.1 total_count

```
int total_count
```

Total cumulative number of times that a previously-alive **com.rti.dds.publication.DataWriter** (p. 553) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.

This count does not change when an already not alive **com.rti.dds.publication.DataWriter** (p. 553) simply remains not alive for another liveliness period.

8.160.2.2 total_count_change

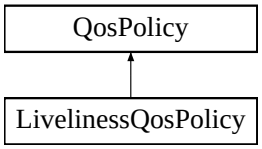
```
int total_count_change
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.161 LivelinessQosPolicy Class Reference

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".

Inheritance diagram for LivelinessQosPolicy:



Public Attributes

- **LivelinessQosPolicyKind kind**

  The kind of liveliness desired.
- final **Duration_t lease_duration**

  *The duration within which a **com.rti.dds.publication.DataWriter** (p. 553) must be asserted, or else it is assumed to be not alive.*
- int **assertions_per_lease_duration**

  *The number of assertions a **com.rti.dds.publication.DataWriter** (p. 553) will send during a its **com.rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1246).*

8.161.1 Detailed Description

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 450) entities to detect when **com.rti.dds.publication.DataWriter** (p. 553) entities become disconnected or "dead.".

The liveliness status of a **com.rti.dds.publication.DataWriter** (p. 553) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 244) policy. The application is also informed via **com.rti.dds.↩ infrastructure.Listener** (p. 1236) when an **com.rti.dds.publication.DataWriter** (p. 553) is no longer alive.

A **com.rti.dds.publication.DataWriter** (p. 553) commits to signalling its liveliness at intervals not to exceed the **com.↩ rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1246) configured on the **com.rti.dds.publication.↩ DataWriter** (p. 553). The rate at which the **com.rti.dds.publication.DataWriter** (p. 553) will signal its liveliness is defined by **com.rti.dds.infrastructure.LivelinessQosPolicy.assertions_per_lease_duration** (p. 1246).

The **com.rti.dds.subscription.DataReader** (p. 450) lease_duration specifies the maximum period at which matching DataWriters must have their liveliness asserted.

In addition, in the subscribing application Connext DDS uses an internal thread that wakes up at the period set by the DataReader's lease_duration to see if a DataWriter lease_duration has been violated.

Important: The DataReader lease_duration only configures how often a DataReader will check the liveliness of the matching DataWriters. The DataWriter lease_duration configures the period at which a DataWriter must assert its liveliness and is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

Listeners are used to notify a **com.rti.dds.subscription.DataReader** (p. 450) of loss of liveliness and **com.rti.dds.↩ publication.DataWriter** (p. 553) of violations to the liveliness contract. The on_liveliness_lost() callback is only called *once*, after the first time the lease_duration is exceeded (when the **com.rti.dds.publication.DataWriter** (p. 553) first loses liveliness).

This QoS policy can be used during system integration to ensure that applications have been coded to meet design specifications. It can also be used during runtime to detect when systems are performing outside of design specifications. Receiving applications can take appropriate actions in response to disconnected DataWriters.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.↩ LivelinessLostStatus** (p. 1242);
com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS, **com.rti.dds.subscription.↩ LivelinessChangedStatus** (p. 1239);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.161.2 Usage

This policy controls the mechanism and parameters used by RTI Connext to ensure that particular DataWriters on the network are still alive. The liveliness can also affect the ownership of a particular instance, as determined by the **OWNERSHIP** (p. 244) policy.

This policy has several settings to support both data types that are updated periodically as well as those that are changed sporadically. It also allows customisation for different application requirements in terms of the kinds of failures that will be detected by the liveliness mechanism.

The com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS liveliness setting is most appropriate for applications that only need to detect failures at the process-level, but not application-logic failures within a process. RTI Connext takes responsibility for renewing the leases at the required rates and thus, as long as the local process where a **com.rti.dds.domain.DomainParticipant** (p. 670) is running and the link connecting it to remote participants remains connected, the entities within the **com.rti.dds.domain.DomainParticipant** (p. 670) will be considered alive. This requires the lowest overhead.

The manual settings (com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_↩ PARTICIPANT_LIVELINESS_QOS, com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.↩ MANUAL_BY_TOPIC_LIVELINESS_QOS) require the application on the publishing side to periodically assert the liveliness before the lease expires to indicate the corresponding **com.rti.dds.infrastructure.Entity** (p. 1029) is still alive. The action can be explicit by calling the **com.rti.dds.publication.DataWriter.assert_liveliness** (p. 572) operation or implicit by writing some data.

The two possible manual settings control the granularity at which the application must assert liveliness.

- The setting com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_↩ PARTICIPANT_LIVELINESS_QOS requires only that one **com.rti.dds.infrastructure.Entity** (p. 1029) within a participant is asserted to be alive to deduce all other **com.rti.dds.infrastructure.Entity** (p. 1029) objects within the same **com.rti.dds.domain.DomainParticipant** (p. 670) are also alive.

- The setting com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_TOPIC↩ _LIVELINESS_QOS requires that at least one instance within the **com.rti.dds.publication.DataWriter** (p. 553) is asserted.

Changes in **LIVELINESS** (p. 239) must be detected by the Service with a time-granularity greater or equal to the **com.↩ rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1246). This ensures that the value of the **com.rti.↩ dds.subscription.LivelinessChangedStatus** (p. 1239) is updated at least once during each lease_duration and the related Listeners and **com.rti.dds.infrastructure.WaitSet** (p. 1973) s are notified within a lease_duration from the time the **LIVELINESS** (p. 239) changed.

8.161.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered kind* >= *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247) kind are considered ordered such that↩ : com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS < com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_↩ LIVELINESS_QOS < com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_↩ BY_TOPIC_LIVELINESS_QOS.

- the inequality *offered* `lease_duration` <= *requested* `lease_duration` evaluates to com.rti.dds.↩ infrastructure.true.

See also

Relationship between registration, liveliness and ownership (p. 1344)

8.161.4 Member Data Documentation

8.161.4.1 kind

```
LivelinessQosPolicyKind kind
```

The kind of liveliness desired.

[default] com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS

8.161.4.2 lease_duration

```
final  Duration_t lease_duration
```

The duration within which a **com.rti.dds.publication.DataWriter** (p. 553) must be asserted, or else it is assumed to be not alive.

For a DataWriter, the lease_duration specifies a timeout by which liveliness must be asserted for the DataWriter, or the DataWriter will be considered inactive or not alive.

For a DataReader, the lease_duration specifies the maximum period at which the DataReader will check to see if the matching DataWriters are still alive according to the DataWriters lease_duration value.

Important: The DataWriter lease_duration is the value that the DataReader uses to determine if a matching DataWriter is alive or not.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [0,1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.161.4.3 assertions_per_lease_duration

```
int assertions_per_lease_duration
```

The number of assertions a **com.rti.dds.publication.DataWriter** (p. 553) will send during a its **com.rti.dds.↩ infrastructure.LivelinessQosPolicy.lease_duration** (p. 1246).

This field only applies to a **com.rti.dds.publication.DataWriter** (p. 553) and is not considered during QoS compatibility checks.

The default value is 3. A higher value will make the liveliness mechanism more robust against packet losses, but it will also increase the network traffic.

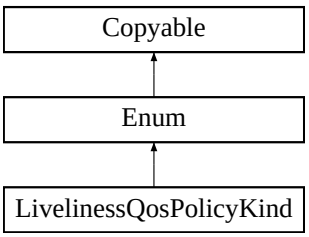
[default] 3

[range] [2, 100 million]

8.162 LivelinessQosPolicyKind Class Reference

Kinds of liveliness.

Inheritance diagram for LivelinessQosPolicyKind:



Static Public Attributes

- static final **LivelinessQosPolicyKind AUTOMATIC_LIVELINESS_QOS**

  [default] *The infrastructure will automatically signal liveliness for the* ***com.rti.dds.publication.DataWriter*** *(p. 553) (s) at least as often as required by the* ***com.rti.dds.publication.DataWriter*** *(p. 553) (S)* `lease_duration`*.*
- static final **LivelinessQosPolicyKind MANUAL_BY_PARTICIPANT_LIVELINESS_QOS**

  RTI Connext will assume that as long as at least one ***com.rti.dds.publication.DataWriter*** *(p. 553) belonging to the* ***com.rti.dds.domain.DomainParticipant*** *(p. 670) (or the* ***com.rti.dds.domain.DomainParticipant*** *(p. 670) itself) has asserted its liveliness, then the other DataWriters belonging to that same* ***com.rti.dds.domain.DomainParticipant*** *(p. 670) are also alive.*
- static final **LivelinessQosPolicyKind MANUAL_BY_TOPIC_LIVELINESS_QOS**

  RTI Connext will only assume liveliness of the ***com.rti.dds.publication.DataWriter*** *(p. 553) if the application has asserted liveliness of that* ***com.rti.dds.publication.DataWriter*** *(p. 553) itself.*

Additional Inherited Members

8.162.1 Detailed Description

Kinds of liveliness.

QoS:

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1243)

8.162.2 Member Data Documentation

8.162.2.1 AUTOMATIC_LIVELINESS_QOS

```
final LivelinessQosPolicyKind AUTOMATIC_LIVELINESS_QOS [static]
```

[default] The infrastructure will automatically signal liveliness for the **com.rti.dds.publication.DataWriter** (p. 553) (s) at least as often as required by the **com.rti.dds.publication.DataWriter** (p. 553) (S) `lease_duration`.

A **com.rti.dds.publication.DataWriter** (p. 553) with this setting does not need to take any specific action in order to be considered 'alive.' The **com.rti.dds.publication.DataWriter** (p. 553) is only 'not alive' when the participant to which it belongs terminates (gracefully or not), or when there is a network problem that prevents the current participant from contacting that remote participant.

8.162.2.2 MANUAL_BY_PARTICIPANT_LIVELINESS_QOS

```
final LivelinessQosPolicyKind MANUAL_BY_PARTICIPANT_LIVELINESS_QOS [static]
```

RTI Connext will assume that as long as at least one **com.rti.dds.publication.DataWriter** (p. 553) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 670) (or the **com.rti.dds.domain.DomainParticipant** (p. 670) itself) has asserted its liveliness, then the other DataWriters belonging to that same **com.rti.dds.domain.DomainParticipant** (p. 670) are also alive.

The user application takes responsibility to signal liveliness to RTI Connext either by calling **com.rti.dds.domain.↩ DomainParticipant.assert_liveliness** (p. 727), or by calling **com.rti.dds.publication.DataWriter.assert_liveliness** (p. 572), or **com.rti.ndds.example.FooDataWriter.write** (p. 1105) on any **com.rti.dds.publication.DataWriter** (p. 553) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 670).

8.162.2.3 MANUAL_BY_TOPIC_LIVELINESS_QOS

```
final LivelinessQosPolicyKind MANUAL_BY_TOPIC_LIVELINESS_QOS [static]
```

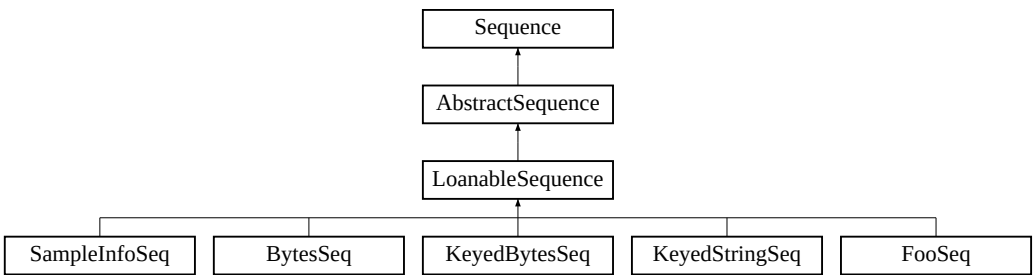
RTI Connext will only assume liveliness of the **com.rti.dds.publication.DataWriter** (p. 553) if the application has asserted liveliness of that **com.rti.dds.publication.DataWriter** (p. 553) itself.

The user application takes responsibility to signal liveliness to RTI Connext using the **com.rti.dds.publication.Data↩ Writer.assert_liveliness** (p. 572) method, or by writing some data.

8.163 LoanableSequence Class Reference

A sequence capable of storing its elements directly or taking out a loan on them from an internal middleware store.

Inheritance diagram for LoanableSequence:



Public Member Functions

- **LoanableSequence** (Class elementType)

  Construct a new sequence for elements of the given type.
- **LoanableSequence** (Class elementType, int maximum)

  Construct a new sequence for elements of the given type.
- **LoanableSequence** (Class elementType, Collection elements)

  Construct a new sequence for elements of the given type.
- final boolean **hasOwnership** ()

  Return the value of the owned flag.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- void **setMaximum** (int new_max)

  Resize this sequence to a new desired maximum.
- Object **set** (int index, Object element)

  Replaces the element at the specified position in this sequence with the specified element.
- Object **get** (int index)

  Returns the element at the specified position in this sequence.
- int **size** ()

  Returns the length of the sequence.

8.163.1 Detailed Description

A sequence capable of storing its elements directly or taking out a loan on them from an internal middleware store.

See also

com.rti.dds.subscription.DataReader::read_untyped (p. 474)(java.util.List, **com.rti.dds.subscription.SampleInfoSeq** (p. 1647), int, int, int, int)

com.rti.dds.subscription.DataReader::take_untyped (p. 475)(java.util.List, **com.rti.dds.subscription.SampleInfoSeq** (p. 1647), int, int, int, int)

8.163.2 Constructor & Destructor Documentation

8.163.2.1 LoanableSequence() [1/3]

```
LoanableSequence (
            Class elementType )
```

Construct a new sequence for elements of the given type.

8.163.2.2 LoanableSequence() [2/3]

```
LoanableSequence (
            Class elementType,
            int maximum )
```

Construct a new sequence for elements of the given type.

8.163.2.3 LoanableSequence() [3/3]

```
LoanableSequence (
            Class elementType,
            Collection elements )
```

Construct a new sequence for elements of the given type.

8.163.3 Member Function Documentation

8.163.3.1 hasOwnership()

```
final boolean hasOwnership ( )
```

Return the value of the owned flag.

Returns

com.rti.dds.infrastructure.true if sequence owns the underlying buffer, or com.rti.dds.infrastructure.false if it has an outstanding loan.

Referenced by **LoanableSequence.get()**, **LoanableSequence.getMaximum()**, and **LoanableSequence.size()**.

8.163.3.2 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 329), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

References **LoanableSequence.hasOwnership()**.

Referenced by **BytesSeq.copy_from()**, **KeyedBytesSeq.copy_from()**, **KeyedStringSeq.copy_from()**, and **Foo↩Seq.copy_from()**.

8.163.3.3 setMaximum()

```
void setMaximum (
             int new_max )
```

Resize this sequence to a new desired maximum.

This operation does nothing if the new desired maximum matches the current maximum.

Note: If you add an element with **`add()`** (p. 329), the sequence's size is increased implicitly.

Postcondition

length == MINIMUM(original length, new_max)

Parameters

new_max	Must be >= 0.

Reimplemented from **AbstractSequence** (p. 328).

Referenced by **BytesSeq.copy_from()**, **KeyedBytesSeq.copy_from()**, **KeyedStringSeq.copy_from()**, and **Foo↩Seq.copy_from()**.

8.163.3.4 set()

```
Object set (
             int index,
             Object element )
```

Replaces the element at the specified position in this sequence with the specified element.

See also

java.util.List::set(int, java.lang.Object)

8.163.3.5 get()

```
Object get (
             int index )
```

Returns the element at the specified position in this sequence.

See also

java.util.List::get(int)

Reimplemented in **SampleInfoSeq** (p. 1648), **BytesSeq** (p. 404), **KeyedBytesSeq** (p. 1197), and **KeyedStringSeq** (p. 1225).

References **LoanableSequence.hasOwnership()**.

8.163.3.6 size()

```
int size ( )
```

Returns the length of the sequence.

See also

java.util.List::get(int)

References **LoanableSequence.hasOwnership()**.

Referenced by **BytesSeq.copy_from()**, **KeyedBytesSeq.copy_from()**, **KeyedStringSeq.copy_from()**, and **Foo↩ Seq.copy_from()**.

8.164 Locator_t Class Reference

<<***extension***>> *(*p. *155)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

Inherits Struct.

Public Member Functions

- **Locator_t** ()
  Constructor.

Public Attributes

- int **kind**
  The kind of locator.
- int **port**
  the port number
- final byte[ ] **address** = new byte[**ADDRESS_LENGTH_MAX**]
  *A **com.rti.dds.infrastructure.Locator_t.ADDRESS_LENGTH_MAX** (*p. *1257) octet field to hold the IP address.*

Static Public Attributes

- static final int **KIND_INVALID**

  Locator of this kind is invalid.
- static final int **PORT_INVALID**

  An invalid port.
- static final byte[ ] **ADDRESS_INVALID**

  An invalid address.
- static final **Locator_t INVALID**

  An invalid locator.
- static final int **KIND_UDPv4**

  A locator for a UDPv4 address.
- static final int **KIND_UDPv4_WAN**

  A locator for a UDPv4 asymmetric transport address.
- static final int **KIND_SHMEM**

  A locator for an address acessed via shared memory.
- static final int **KIND_SHMEM_510**

  A locator for an address acessed via shared memory for RTI Connext 5.1.0 and earlier.
- static final int **KIND_UDPv6**

  A locator for a UDPv6 address.
- static final int **KIND_UDPv6_510**

  A locator for a UDPv6 address for RTI Connext 5.1.0 and earlier.
- static final int **KIND_RESERVED**

  Locator of this kind is reserved.
- static final int **ADDRESS_LENGTH_MAX** = 16

  Declares length of address field in locator.

8.164.1 Detailed Description

<<***extension***>> *(*p. *155)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

8.164.2 Constructor & Destructor Documentation

8.164.2.1 Locator_t()

```
Locator_t ( )
```

Constructor.

8.164.3 Member Data Documentation

8.164.3.1 KIND_INVALID

```
final int KIND_INVALID  [static]
```

Locator of this kind is invalid.

8.164.3.2 PORT_INVALID

```
final int PORT_INVALID  [static]
```

An invalid port.

8.164.3.3 ADDRESS_INVALID

```
final byte [] ADDRESS_INVALID  [static]
```

Initial value:

```
= {0,0,0,0,
                                                    0,0,0,0,
                                                    0,0,0,0,
                                                    0,0,0,0}
```

An invalid address.

8.164.3.4 INVALID

```
final  Locator_t INVALID  [static]
```

Initial value:

```
= new Locator_t(
        KIND_INVALID, PORT_INVALID, ADDRESS_INVALID)
```

An invalid locator.

8.164.3.5 KIND_UDPv4

```
final int KIND_UDPv4  [static]
```

A locator for a UDPv4 address.

8.164.3.6 KIND_UDPv4_WAN

```
final int KIND_UDPv4_WAN  [static]
```

A locator for a UDPv4 asymmetric transport address.

8.164.3.7 KIND_SHMEM

```
final int KIND_SHMEM  [static]
```

A locator for an address acessed via shared memory.

8.164.3.8 KIND_SHMEM_510

```
final int KIND_SHMEM_510  [static]
```

A locator for an address acessed via shared memory for RTI Connext 5.1.0 and earlier.

8.164.3.9 KIND_UDPv6

```
final int KIND_UDPv6  [static]
```

A locator for a UDPv6 address.

8.164.3.10 KIND_UDPv6_510

```
final int KIND_UDPv6_510  [static]
```

A locator for a UDPv6 address for RTI Connext 5.1.0 and earlier.

8.164.3.11 KIND_RESERVED

```
final int KIND_RESERVED  [static]
```

Locator of this kind is reserved.

8.164.3.12 ADDRESS_LENGTH_MAX

```
final int ADDRESS_LENGTH_MAX = 16  [static]
```

Declares length of address field in locator.

8.164.3.13 kind

```
int kind
```

The kind of locator.

If the **Locator_t** (p. 1253) kind is **com.rti.dds.infrastructure.Locator_t.KIND_UDPv4** (p. 1255), the address contains an IPv4 address. In this case, the leading 12 octets of the **com.rti.dds.infrastructure.Locator_t.address** (p. 1257) must be zero. The last 4 octets of **com.rti.dds.infrastructure.Locator_t.address** (p. 1257) are used to store the IPv4 address.

If the **Locator_t** (p. 1253) kind is **com.rti.dds.infrastructure.Locator_t.KIND_UDPv6** (p. 1256), the address contains an IPv6 address. IPv6 addresses typically use a shorthand hexadecimal notation that maps one-to-one to the 16 octets in the **com.rti.dds.infrastructure.Locator_t.address** (p. 1257) field.

8.164.3.14 port

```
int port
```

the port number

8.164.3.15 address

```
final byte [] address = new byte[ ADDRESS_LENGTH_MAX]
```

A **com.rti.dds.infrastructure.Locator_t.ADDRESS_LENGTH_MAX** (p. 1257) octet field to hold the IP address.

8.165 LocatorFilter_t Class Reference

Specifies the configuration of an individual channel within a MultiChannel DataWriter.

Inherits Struct.

Public Member Functions

- **LocatorFilter_t** ()

  Constructor.
- **LocatorFilter_t** (**LocatorFilter_t** src)

  Constructor.
- **LocatorFilter_t** (**LocatorSeq locators**, String **filter_expression**)

  Constructor.

Public Attributes

- **LocatorSeq locators**

  Sequence containing from one to 16 ***com.rti.dds.infrastructure.Locator_t*** *(*p. *1253), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.*
- String **filter_expression**

  A logical expression used to determine the data that will be published in the channel.

8.165.1 Detailed Description

Specifies the configuration of an individual channel within a MultiChannel DataWriter.

QoS:

com.rti.dds.infrastructure.LocatorFilterQosPolicy (p. 1260)

8.165.2 Constructor & Destructor Documentation

8.165.2.1 LocatorFilter_t() [1/3]

```
LocatorFilter_t ( )
```

Constructor.

8.165.2.2 LocatorFilter_t() [2/3]

```
LocatorFilter_t (
            LocatorFilter_t src )
```

Constructor.

Parameters

src	<<***in***>> *(*p. *156)* Locator used to initialized the new locator.

References **LocatorFilter_t.filter_expression**, and **LocatorFilter_t.locators**.

8.165.2.3 LocatorFilter_t() [3/3]

```
LocatorFilter_t (
            LocatorSeq locators,
            String filter_expression )
```

Constructor.

Parameters

locators	<<***in***>> *(*p. *156)* Locators.
filter_expression	<<***in***>> *(*p. *156)* Filter expression.

References **LocatorFilter_t.filter_expression**, and **LocatorFilter_t.locators**.

8.165.3 Member Data Documentation

8.165.3.1 locators

```
LocatorSeq locators
```

Initial value:

```
=
       new LocatorSeq()
```

Sequence containing from one to 16 **com.rti.dds.infrastructure.Locator_t** (p. 1253), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.

[default] Empty sequence.

Referenced by **LocatorFilter_t.LocatorFilter_t()**.

8.165.3.2 filter_expression

```
String filter_expression
```

A logical expression used to determine the data that will be published in the channel.

If the expression evaluates to TRUE, a sample will be published on the channel.

An empty string always evaluates the expression to TRUE.

A NULL value is not allowed.

The syntax of the expression will depend on the value of **com.rti.dds.infrastructure.LocatorFilterQosPolicy.filter_↩ name** (p. 1261)

See also

Queries and Filters Syntax (p. 104)

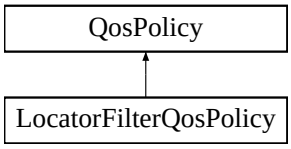
[default] NULL (invalid value)

Referenced by **LocatorFilter_t.LocatorFilter_t()**.

8.166 LocatorFilterQosPolicy Class Reference

The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.↩ builtin.PublicationBuiltinTopicData** (p. 1452).

Inheritance diagram for LocatorFilterQosPolicy:



Public Attributes

- final **LocatorFilterSeq locator_filters**

  *A sequence of **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258). Each **com.rti.dds.infrastructure.LocatorFilter↩ _t** (p. 1258) reports the configuration of a single channel of a MultiChannel DataWriter.*
- String **filter_name**

  Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

8.166.1 Detailed Description

The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452).

Entity:

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.166.2 Member Data Documentation

8.166.2.1 locator_filters

```
final LocatorFilterSeq locator_filters
```

A sequence of **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258). Each **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258) reports the configuration of a single channel of a MultiChannel DataWriter.

A sequence length of zero indicates the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) is not in use.

[default] Empty sequence.

8.166.2.2 filter_name

```
String filter_name
```

Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported: **com.rti.dds.domain.DomainParticipant.SQLFILTER_NAME** (p. 50) and **com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50).

[default] com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME (p. 50)

8.167 LocatorFilterSeq Class Reference

Declares IDL sequence< **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258) >.

Inherits ArraySequence.

8.167.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258) `>`.

A sequence of **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1258) used to report the channels' properties. If the length of the sequence is zero, the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) is not in use.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.LocatorFilter_t (p. 1258)

8.168 LocatorSeq Class Reference

Declares IDL `sequence <` **com.rti.dds.infrastructure.Locator_t** (p. 1253) `>`

Inherits ArraySequence.

8.168.1 Detailed Description

Declares IDL `sequence <` **com.rti.dds.infrastructure.Locator_t** (p. 1253) `>`

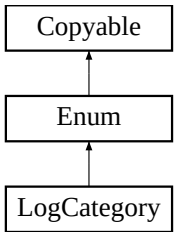
See also

com.rti.dds.infrastructure.Locator_t (p. 1253)

8.169 LogCategory Class Reference

Categories of logged messages.

Inheritance diagram for LogCategory:



Static Public Attributes

- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_PLATFORM**

  Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connext is running are in this category.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION**

  Log messages pertaining to data serialization and deserialization and network traffic are in this category.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_DATABASE**

  Log messages pertaining to the internal database in which RTI Connext objects are stored are in this category.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_ENTITIES**

  Log messages pertaining to local and remote entities, and to a subset of the discovery process, are in this category. (To see all discovery-related messages, use the DISCOVERY category.)
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_API**

  Log messages pertaining to the API layer of RTI Connext (such as method argument validation) are in this category.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_DISCOVERY**

  Log messages pertaining to discovery are in this category.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_SECURITY**

  Log messages pertaining to security are in this category. These messages include any messages logged by RTI Connext components even if they are not related to the Security Plugins.
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_USER**

  Log messages that are generated by the user using the following log APIs: ***com.rti.ndds.config.Logger.emergency*** *(p. 1272),* ***com.rti.ndds.config.Logger.alert*** *(p. 1272),* ***com.rti.ndds.config.Logger.critical*** *(p. 1273),* ***com.rti.ndds.config.Logger.error*** *(p. 1273),* ***com.rti.ndds.config.Logger.warning*** *(p. 1273),* ***com.rti.ndds.config.Logger.notice*** *(p. 1273),* ***com.rti.ndds.config.Logger.informational*** *(p. 1273),* ***com.rti.ndds.config.Logger.debug*** *(p. 1274).*
- static final **LogCategory NDDS_CONFIG_LOG_CATEGORY_ALL**

  Log messages pertaining to all categories in RTI Connext.

Additional Inherited Members

8.169.1 Detailed Description

Categories of logged messages.

The **com.rti.ndds.config.Logger.get_verbosity_by_category** (p. 1269) and **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1269) can be used to specify different verbosities for different categories of messages.

8.169.2 Member Data Documentation

8.169.2.1 NDDS_CONFIG_LOG_CATEGORY_PLATFORM

```
final LogCategory NDDS_CONFIG_LOG_CATEGORY_PLATFORM [static]
```

Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connext is running are in this category.

8.169.2.2 NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION

final **LogCategory** NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION [static]

Log messages pertaining to data serialization and deserialization and network traffic are in this category.

8.169.2.3 NDDS_CONFIG_LOG_CATEGORY_DATABASE

final **LogCategory** NDDS_CONFIG_LOG_CATEGORY_DATABASE [static]

Log messages pertaining to the internal database in which RTI Connext objects are stored are in this category.

8.169.2.4 NDDS_CONFIG_LOG_CATEGORY_ENTITIES

final **LogCategory** NDDS_CONFIG_LOG_CATEGORY_ENTITIES [static]

Log messages pertaining to local and remote entities, and to a subset of the discovery process, are in this category. (To see all discovery-related messages, use the DISCOVERY category.)

8.169.2.5 NDDS_CONFIG_LOG_CATEGORY_API

final **LogCategory** NDDS_CONFIG_LOG_CATEGORY_API [static]

Log messages pertaining to the API layer of RTI Connext (such as method argument validation) are in this category.

8.169.2.6 NDDS_CONFIG_LOG_CATEGORY_DISCOVERY

final **LogCategory** NDDS_CONFIG_LOG_CATEGORY_DISCOVERY [static]

Log messages pertaining to discovery are in this category.

8.169.2.7 NDDS_CONFIG_LOG_CATEGORY_SECURITY

```
final LogCategory NDDS_CONFIG_LOG_CATEGORY_SECURITY [static]
```

Log messages pertaining to security are in this category. These messages include any messages logged by RTI Connext components even if they are not related to the Security Plugins.

8.169.2.8 NDDS_CONFIG_LOG_CATEGORY_USER

```
final LogCategory NDDS_CONFIG_LOG_CATEGORY_USER [static]
```

Log messages that are generated by the user using the following log APIs: **com.rti.ndds.config.Logger.emergency** (p. 1272), **com.rti.ndds.config.Logger.alert** (p. 1272), **com.rti.ndds.config.Logger.critical** (p. 1273), **com.rti.↩ ndds.config.Logger.error** (p. 1273), **com.rti.ndds.config.Logger.warning** (p. 1273), **com.rti.ndds.config.Logger.↩ notice** (p. 1273), **com.rti.ndds.config.Logger.informational** (p. 1273), **com.rti.ndds.config.Logger.debug** (p. 1274).

8.169.2.9 NDDS_CONFIG_LOG_CATEGORY_ALL

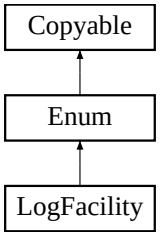
```
final LogCategory NDDS_CONFIG_LOG_CATEGORY_ALL [static]
```

Log messages pertaining to all categories in RTI Connext.

8.170 LogFacility Class Reference

A number that identifies the source of a log message.

Inheritance diagram for LogFacility:



Static Public Attributes

- static final **LogFacility NDDS_CONFIG_LOG_FACILITY_USER**

  *A log message produced by the APIs that log user messages in **com.rti.ndds.config.Logger** (p. 1267) (e.g, **com.rti.ndds.config.Logger.emergency** (p. 1272)).*
- static final **LogFacility NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT**

  A security-related message logged by the RTI Security Plugins Logging Plugin.
- static final **LogFacility NDDS_CONFIG_LOG_FACILITY_SERVICE**

  A log message produced by an Infrastructure Service, such as Routing Service.
- static final **LogFacility NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE**

  A log message produced by RTI Connext Core Libraries.

Additional Inherited Members

8.170.1 Detailed Description

A number that identifies the source of a log message.

In the Syslog Protocol, the Facility is a numerical code that represents the machine process that created a Syslog event. RTI Connext uses the facility to represent the source of a given log message.

8.170.2 Member Data Documentation

8.170.2.1 NDDS_CONFIG_LOG_FACILITY_USER

```
final LogFacility NDDS_CONFIG_LOG_FACILITY_USER [static]
```

A log message produced by the APIs that log user messages in **com.rti.ndds.config.Logger** (p. 1267) (e.g, **com.rti.ndds.config.Logger.emergency** (p. 1272)).

Numerical code: 1

8.170.2.2 NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT

```
final LogFacility NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT [static]
```

A security-related message logged by the RTI Security Plugins Logging Plugin.

Numerical code: 10

8.170.2.3 NDDS_CONFIG_LOG_FACILITY_SERVICE

```
final  LogFacility NDDS_CONFIG_LOG_FACILITY_SERVICE  [static]
```

A log message produced by an Infrastructure Service, such as Routing Service.

Infrastructure Services operate using the Core Libraries. The messages that the Core Libraries create are categorized under the MIDDLEWARE facility. Log messages that are directly generated by the Infrastructure Service itself are marked with the SERVICE facility.

Numerical code: 22

8.170.2.4 NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE

```
final  LogFacility NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE  [static]
```

A log message produced by RTI Connext Core Libraries.

Numerical code: 23

8.171 Logger Class Reference

<<***interface***>> *(*p. *156)* The singleton type used to configure RTI Connext logging.

Public Member Functions

- **LogVerbosity get_verbosity** ()

  Get the verbosity at which RTI Connext is currently logging diagnostic information.
- **LogVerbosity get_verbosity_by_category** (**LogCategory** category)

  Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.
- void **set_verbosity** (**LogVerbosity** verbosity)

  Set the verbosity at which RTI Connext will log diagnostic information.
- void **set_verbosity_by_category** (**LogCategory** category, **LogVerbosity** verbosity)

  Set the verbosity at which RTI Connext will log diagnostic information in the given category.
- File **get_output_file** ()

  Get the file to which the logged output is redirected.
- void **set_output_file** (File out) throws IOException

  Set the file to which the logged output is redirected.
- void **set_output_file_set** (String file_prefix, String file_suffix, int max_capacity, int max_files) throws IOException

  Configure a set of files to redirect the logged output.
- **LoggerDevice get_output_device** ()

  Return the user device registered with the logger.
- void **set_output_device** (**LoggerDevice** device) throws IOException

  Register a ***com.rti.ndds.config.LoggerDevice*** *(*p. *1274).*
- **LogPrintFormat get_print_format** ()

*Get the current message format for the log level **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1279).*

- boolean **set_print_format_by_log_level** (**LogPrintFormat** print_format, **LogLevel** log_level)

  *Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (p. 288) is printed, the user can select the information that will be part of the **Activity Context** (p. 288) by using the API **com.rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).*

- **LogPrintFormat get_print_format_by_log_level** (**LogLevel** log_level)

  Get the current message format, by log level, that RTI Connext is using to log diagnostic information.

- boolean **set_print_format** (**LogPrintFormat** print_format)

  *Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1279). When the **Activity Context** (p. 288) is printed, the user can select the information that will be part of the **Activity Context** (p. 288) by using the API **com.rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).*

- void **emergency** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **alert** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **critical** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **error** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **warning** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **notice** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **informational** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

- void **debug** (String msg)

  *Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).*

Static Public Member Functions

- static **Logger get_instance** ()

  Get the singleton instance of this type.

8.171.1 Detailed Description

<<***interface***>> *(p. 156)* The singleton type used to configure RTI Connext logging.

8.171.2 Member Function Documentation

8.171.2.1 get_instance()

```
static Logger get_instance ( ) [static]
```

Get the singleton instance of this type.

8.171.2.2 get_verbosity()

```
LogVerbosity get_verbosity ( )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information.

The default verbosity if **com.rti.ndds.config.Logger.set_verbosity** (p. 1269) is never called is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1286).

If **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1269) has been used to set different verbosities for different categories of messages, this method will return the maximum verbosity of all categories.

8.171.2.3 get_verbosity_by_category()

```
LogVerbosity get_verbosity_by_category (
            LogCategory category )
```

Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.

The default verbosity if **com.rti.ndds.config.Logger.set_verbosity** (p. 1269) and **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1269) are never called is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1286).

References **Enum.ordinal()**.

8.171.2.4 set_verbosity()

```
void set_verbosity (
            LogVerbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information.

Note: Logging at high verbosities will be detrimental to your application's performance. Your default setting should typically remain at **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1286) or below. (The default verbosity if you never set it is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1286).)

References **Enum.ordinal()**.

8.171.2.5 set_verbosity_by_category()

```
void set_verbosity_by_category (
              LogCategory category,
              LogVerbosity verbosity )
```

Set the verbosity at which RTI Connext will log diagnostic information in the given category.

References **Enum.ordinal()**.

8.171.2.6 get_output_file()

```
File get_output_file ( )
```

Get the file to which the logged output is redirected.

If no output file has been registered through **com.rti.ndds.config.Logger.set_output_file** (p. 1270), this method will return NULL. In this case, logged output will on most platforms go to standard out as if through printf.

8.171.2.7 set_output_file()

```
void set_output_file (
             File out ) throws IOException
```

Set the file to which the logged output is redirected.

The file passed may be NULL, in which case further logged output will be redirected to the platform-specific default output location (standard out on most platforms).

For better performance when log messages are generated frequently, the log messages are not flushed into a file immediately after they are generated. In other words, while writing a log message, RTI Connext only calls the function fwrite() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fwrite.html); it does not call the function fflush() (see https://pubs.opengroup.org/onlinepubs/009695399/functions/fflush.html). If your application requires a different flushing behavior, you may use **com.rti.ndds.config.Logger.set_output_device** (p. 1271) to configure a custom logging device.

8.171.2.8 set_output_file_set()

```
void set_output_file_set (
             String file_prefix,
             String file_suffix,
             int max_capacity,
             int max_files ) throws IOException
```

Configure a set of files to redirect the logged output.

The logged output will be redirected to a set of files whose names are configured with a prefix and a suffix. The maximum number of bytes configures how many bytes to write into a file before opening the next file. After reaching the maximum number of files, the first one is overwritten.

For example, if the prefix is 'Foo', the suffix is '.txt', the max number of bytes is 1GB, and the max number of files is 3, the logger will create (at most) these files: Foo1.txt, Foo2.txt, and Foo3.txt. It will write to Foo1.txt, and after writing 1GB, it will move on to Foo2.txt, then to Foo3.txt, then to Foo1.txt again, and so on.

To stop logging to these files and redirect the output to the platform-specific location, pass NULL, NULL, 0, 0.

See **com.rti.ndds.config.Logger.set_output_file** (p. 1270) for the flushing behavior.

8.171.2.9 get_output_device()

```
LoggerDevice get_output_device ( )
```

Return the user device registered with the logger.

Returns

Registered user device or NULL if no user device is registered.

8.171.2.10 set_output_device()

```
void set_output_device (
            LoggerDevice device ) throws IOException
```

Register a **com.rti.ndds.config.LoggerDevice** (p. 1274).

Register the specified logging device with the logger.

There can be at most only one device registered with the logger at any given time.

When a device is installed, the logger will stop sending the log messages to the standard output and to the file set with **com.rti.ndds.config.Logger.set_output_file** (p. 1270).

To remove an existing device, use this method with NULL as the device parameter. After a device is removed the logger will continue sending log messages to the standard output and to the output file.

To replace an existing device with a new device, use this method providing the new device as the device parameter.

When a device is unregistered (by setting it to NULL), **com.rti.ndds.config.LoggerDevice** (p. 1274) calls the method **com.rti.ndds.config.LoggerDevice.close** (p. 1275).

Parameters

device	<<***in***>> *(p. 156)* Logging device.

8.171.2.11 get_print_format()

```
LogPrintFormat get_print_format ( )
```

Get the current message format for the log level **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_↩ ERROR** (p. 1279).

Use **com.rti.ndds.config.Logger.get_print_format_by_log_level** (p. 1272) to retrieve the format for other log levels.

If **com.rti.ndds.config.Logger.set_print_format** (p. 1272) is never called, the default format is **com.rti.ndds.config.↩ LogPrintFormat.NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1283).

8.171.2.12 set_print_format_by_log_level()

```
boolean set_print_format_by_log_level (
              LogPrintFormat print_format,
              LogLevel log_level )
```

Set the message format, by log level, that RTI Connext will use to log diagnostic information. When the **Activity Context** (p. 288) is printed, the user can select the information that will be part of the **Activity Context** (p. 288) by using the API **com.rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).

References **Enum.ordinal()**.

8.171.2.13 get_print_format_by_log_level()

```
 LogPrintFormat get_print_format_by_log_level (
              LogLevel log_level )
```

Get the current message format, by log level, that RTI Connext is using to log diagnostic information.

If **com.rti.ndds.config.Logger.set_print_format** (p. 1272) is never called, the default format is **com.rti.ndds.config.↩ LogPrintFormat.NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1283).

References **Enum.ordinal()**.

8.171.2.14 set_print_format()

```
boolean set_print_format (
              LogPrintFormat print_format )
```

Set the message format that RTI Connext will use to log diagnostic information for all the log levels, except for **com.rti.↩ ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1279). When the **Activity Context** (p. 288) is printed, the user can select the information that will be part of the **Activity Context** (p. 288) by using the API **com.↩ rti.ndds.config.ActivityContext.set_attribute_mask** (p. 289).

References **Enum.ordinal()**.

8.171.2.15 emergency()

```
void emergency (
             String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.16 alert()

```
void alert (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.17 critical()

```
void critical (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.18 error()

```
void error (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.19 warning()

```
void warning (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p. 1777) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.20 notice()

```
void notice (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.21 informational()

```
void informational (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.171.2.22 debug()

```
void debug (
            String msg )
```

Logs message with **com.rti.ndds.config.SyslogLevel.NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p. 1778) and **com.rti.ndds.config.LogCategory.NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1265).

8.172 LoggerDevice Interface Reference

<<***interface***>> *(p. 156)* Logging device interface. Use for user-defined logging devices.

Public Member Functions

- void **write** (**LogMessage** message)

  Write a log message to a specified logging device.
- void **close** ()

  Close the logging device.

8.172.1 Detailed Description

<<***interface***>> *(p. 156)* Logging device interface. Use for user-defined logging devices.

Interface for handling log messages.

By default, the logger sends the log messages generated by RTI Connext to the standard output.

You can use the method **com.rti.ndds.config.Logger.set_output_file** (p. 1270) to redirect the log messages to a file.

To further customize the management of generated log messages, the logger offers the method **com.rti.ndds.config.↩ Logger.set_output_device** (p. 1271) that allows you to install a user-defined logging device.

The logging device installed by the user must implement this interface.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **com.rti.dds.domain.Domain↩ Participant.get_current_time** (p. 732), from any of the logging device operations.

8.172.2 Member Function Documentation

8.172.2.1 write()

```
void write (
            LogMessage message )
```

Write a log message to a specified logging device.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **com.rti.dds.domain.Domain↩ Participant.get_current_time** (p. 732), from any of the logging device operations.

Parameters

message	<<***in***>> *(p. 156)* Message to log.

8.172.2.2 close()

```
void close ( )
```

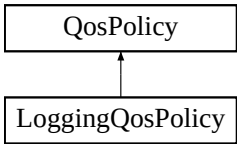
Close the logging device.

Note: It is not safe to make any calls to the RTI Connext core library, including calls to **com.rti.dds.domain.Domain↩ Participant.get_current_time** (p. 732), from any of the logging device operations.

8.173 LoggingQosPolicy Class Reference

Configures the RTI Connext logging facility.

Inheritance diagram for LoggingQosPolicy:



Public Attributes

- **LogVerbosity verbosity**

  The verbosities at which RTI Connext diagnostic information is logged.
- **LogCategory category**

  Categories of logged messages.
- **LogPrintFormat print_format**

  The format used to output RTI Connext diagnostic information.
- String **output_file**

  Specifies the file to which log messages will be redirected to.
- String **output_file_suffix**

  Sets the file suffix when logging to a set of files.
- int **max_bytes_per_file**

  Specifies the maximum number of bytes a single file can contain.
- int **max_files**

  Specifies the maximum number of files to create before overwritting the previous ones.

8.173.1 Detailed Description

Configures the RTI Connext logging facility.

All the properties associated with RTI Connext logging can be configured using this QoS policy. This allows you to configure logging using XML QoS Profiles. See the "Troubleshooting" chapter in the `User's Manual` for details.

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 761)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **Changeable** (p. 256)

8.173.2 Member Data Documentation

8.173.2.1 verbosity

LogVerbosity verbosity

The verbosities at which RTI Connext diagnostic information is logged.

[default] com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR (p. 1286)

8.173.2.2 category

```
LogCategory category
```

Categories of logged messages.

[default] Logging will be enabled for all the categories.

8.173.2.3 print_format

```
LogPrintFormat print_format
```

The format used to output RTI Connext diagnostic information.

[default] com.rti.ndds.config.LogPrintFormat.NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT (p. 1283).

8.173.2.4 output_file

```
String output_file
```

Specifies the file to which log messages will be redirected to.

If the value of ouput_file is set to NULL, log messages will sent to standard output.

If **com.rti.dds.infrastructure.LoggingQosPolicy.max_bytes_per_file** (p. 1277) is not **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), this is used as the file name preffix for a set of numbered files.

[default] NULL

See also

com.rti.ndds.config.Logger.set_output_file_name

com.rti.ndds.config.Logger.set_output_file_set (p. 1270)

8.173.2.5 output_file_suffix

```
String output_file_suffix
```

Sets the file suffix when logging to a set of files.

Note

This field only applies when idref_LoggingQosPolicy_max_bytes_per_file is different than **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

It specifies the suffix to use for the set of files used to redirect the logging output. The preffix is **com.rti.dds.↩ infrastructure.LoggingQosPolicy.output_file** (p. 1277).

[default] NULL

[default] No suffix

See also

com.rti.ndds.config.Logger.set_output_file_set (p. 1270)

8.173.2.6 max_bytes_per_file

```
int max_bytes_per_file
```

Specifies the maximum number of bytes a single file can contain.

When this field is different than **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), it enables logging to separate files as they reach this size.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259) (a single file is used)

See also

com.rti.ndds.config.Logger.set_output_file_set (p. 1270)

8.173.2.7 max_files

```
int max_files
```

Specifies the maximum number of files to create before overwritting the previous ones.

Note

This field only applies when idref_LoggingQosPolicy_max_bytes_per_file is differenet than **com.rti.dds.↩ infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

When this field is different than **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), and the number of files reaches this number, future logging messages overwrite the previously created files.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259) (files aren't overwritten)

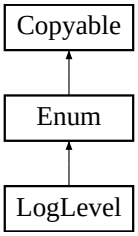
See also

com.rti.ndds.config.Logger.set_output_file_set (p. 1270)

8.174 LogLevel Class Reference

Level category assigned to RTI Connext log messages returned to an output device.

Inheritance diagram for LogLevel:



Static Public Attributes

- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR**

  The message describes a fatal error.
- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_ERROR**

  The message describes an error.
- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_WARNING**

  The message describes a warning.
- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL**

  The message contains information about the lifecycles of local RTI Connext objects will be logged.
- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE**

  The message contains information about the lifecycles of remote RTI Connext objects will be logged.
- static final **LogLevel NDDS_CONFIG_LOG_LEVEL_DEBUG**

  The message contains debug information that might be relevant to your application.

Additional Inherited Members

8.174.1 Detailed Description

Level category assigned to RTI Connext log messages returned to an output device.

8.174.2 Member Data Documentation

8.174.2.1 NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR [static]
```

The message describes a fatal error.

A fatal error indicates an unrecoverable situation in the functioning of RTI Connext. Error messages with this log level usually indicate a violation of an internal invariant or a segfault, and may include the function call stack where the fatal error happened.

8.174.2.2 NDDS_CONFIG_LOG_LEVEL_ERROR

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_ERROR [static]
```

The message describes an error.

An error indicates a non-fatal problem in the functioning of RTI Connext. Errors are usually recoverable and will not stop application execution, although they may prevent some features from working properly. The most common cause of non-fatal errors is incorrect configuration and incorrect arguments.

8.174.2.3 NDDS_CONFIG_LOG_LEVEL_WARNING

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_WARNING [static]
```

The message describes a warning.

A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.

8.174.2.4 NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL [static]
```

The message contains information about the lifecycles of local RTI Connext objects will be logged.

8.174.2.5 NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE [static]
```

The message contains information about the lifecycles of remote RTI Connext objects will be logged.

8.174.2.6 NDDS_CONFIG_LOG_LEVEL_DEBUG

```
final LogLevel NDDS_CONFIG_LOG_LEVEL_DEBUG [static]
```

The message contains debug information that might be relevant to your application.

8.175 LogMessage Class Reference

Log message.

Inherits Struct.

Public Attributes

- String **text**

  Message text.
- **LogLevel level**

  Message level.
- boolean **is_security_message**

  Indicates if the message is a security-related message.
- int **message_id**

  A numeric code that identifies a specific log message.
- **Duration_t timestamp**

  The time when the log message was printed.
- **LogFacility facility**

  The Facility associated with the log message. See ***com.rti.ndds.config.LogFacility*** *(p. 1265).*

8.175.1 Detailed Description

Log message.

8.175.2 Member Data Documentation

8.175.2.1 text

```
String text
```

Message text.

8.175.2.2 level

```
LogLevel level
```

Message level.

8.175.2.3 is_security_message

```
boolean is_security_message
```

Indicates if the message is a security-related message.

A "security-related message" is a log message that meets any of the following:

- A security event logged with the RTI Security Plugins Logging Plugin.
- Any regular message coming from the RTI Security Plugins.
- RTI TLS Support log messages related to OpenSSL (e.g., SSL handshake failures or certificate validation failures).

8.175.2.4 message_id

```
int message_id
```

A numeric code that identifies a specific log message.

Two log messages that have the same message_id will have a similar structure. E.g. "ERROR: Failed to get Data↩ WriterQos" and "ERROR: Failed to get TopicName". In this case, the message_id is associated to the get failure.

8.175.2.5 timestamp

```
Duration_t timestamp
```

The time when the log message was printed.

8.175.2.6 facility

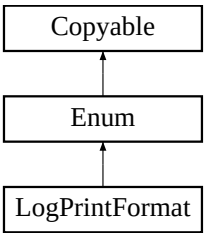
```
LogFacility facility
```

The Facility associated with the log message. See **com.rti.ndds.config.LogFacility** (p. 1265).

8.176 LogPrintFormat Class Reference

The format used to output RTI Connext diagnostic information.

Inheritance diagram for LogPrintFormat:



Static Public Attributes

- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT**

  (default) Print message, method name, log level, ***Activity Context*** *(*p. 288*) (what was happening when the event occurred), and logging category.*
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED**

  Print message, method name, log level, ***Activity Context*** *(*p. 288*), logging category, and timestamp.*
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE**

  Print message with all available context information (includes thread identifier, message location).
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED**

  Print message with all available context information, and timestamp.
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG**

  Print a set of fields (including message number and backtrace information) that may be useful for internal debugging.
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL**

  Print only message number and message location.
- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL**

  Print all available fields.

Additional Inherited Members

8.176.1 Detailed Description

The format used to output RTI Connext diagnostic information.

8.176.2 Member Data Documentation

8.176.2.1 NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT [static]

(default) Print message, method name, log level, **Activity Context** (p. 288) (what was happening when the event occurred), and logging category.

8.176.2.2 NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED [static]

Print message, method name, log level, **Activity Context** (p. 288), logging category, and timestamp.

8.176.2.3 NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE [static]

Print message with all available context information (includes thread identifier, message location).

8.176.2.4 NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED [static]

Print message with all available context information, and timestamp.

8.176.2.5 NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG [static]

Print a set of fields (including message number and backtrace information) that may be useful for internal debugging.

8.176.2.6 NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL

final **LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL [static]

Print only message number and message location.

8.176.2.7 NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL

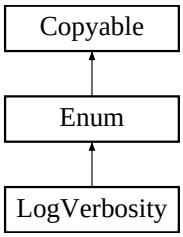
```
final  LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL  [static]
```

Print all available fields.

8.177 LogVerbosity Class Reference

The verbosities at which RTI Connext diagnostic information is logged.

Inheritance diagram for LogVerbosity:



Static Public Attributes

- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_SILENT**

  No further output will be logged.
- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_ERROR**

  Only error and fatal error messages will be logged.
- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_WARNING**

  Both error and warning messages will be logged.
- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL**

  Errors, warnings, and verbose information about the lifecycles of local RTI Connext objects will be logged.
- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE**

  Errors, warnings, and verbose information about the lifecycles of remote RTI Connext objects will be logged.
- static final **LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL**

  Errors, warnings, verbose information about the lifecycles of local and remote RTI Connext objects, and periodic information about RTI Connext threads will be logged.

Additional Inherited Members

8.177.1 Detailed Description

The verbosities at which RTI Connext diagnostic information is logged.

8.177.2 Member Data Documentation

8.177.2.1 NDDS_CONFIG_LOG_VERBOSITY_SILENT

```
final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_SILENT [static]
```

No further output will be logged.

8.177.2.2 NDDS_CONFIG_LOG_VERBOSITY_ERROR

```
final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_ERROR [static]
```

Only error and fatal error messages will be logged.

An error indicates something wrong in the functioning of RTI Connext. The most common cause of errors is incorrect configuration.

8.177.2.3 NDDS_CONFIG_LOG_VERBOSITY_WARNING

```
final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_WARNING [static]
```

Both error and warning messages will be logged.

A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.

8.177.2.4 NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL

```
final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL [static]
```

Errors, warnings, and verbose information about the lifecycles of local RTI Connext objects will be logged.

8.177.2.5 NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE

```
final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE [static]
```

Errors, warnings, and verbose information about the lifecycles of remote RTI Connext objects will be logged.

8.177.2.6 NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL

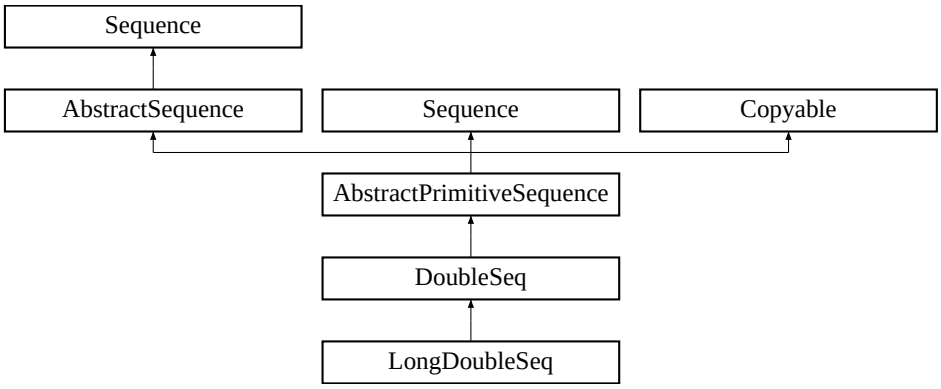
```
final  LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL  [static]
```

Errors, warnings, verbose information about the lifecycles of local and remote RTI Connext objects, and periodic information about RTI Connext threads will be logged.

8.178 LongDoubleSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ LongDouble >

Inheritance diagram for LongDoubleSeq:



Public Member Functions

- **LongDoubleSeq** ()

  Constructs an empty sequence of long doubles with an initial maximum of zero.
- **LongDoubleSeq** (int initialMaximum)

  Constructs an empty sequence of long doubles with the given initial maximum.
- **LongDoubleSeq** (double[ ] doubles)

  Constructs a new sequence containing the given long doubles.

8.178.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ LongDouble >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.LongDouble

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.178.2 Constructor & Destructor Documentation

8.178.2.1 LongDoubleSeq() [1/3]

```
LongDoubleSeq ( )
```

Constructs an empty sequence of long doubles with an initial maximum of zero.

8.178.2.2 LongDoubleSeq() [2/3]

```
LongDoubleSeq (
            int initialMaximum )
```

Constructs an empty sequence of long doubles with the given initial maximum.

8.178.2.3 LongDoubleSeq() [3/3]

```
LongDoubleSeq (
            double[] doubles )
```

Constructs a new sequence containing the given long doubles.

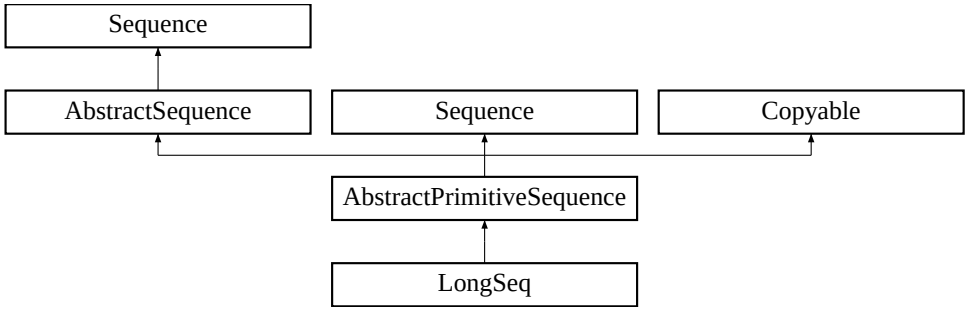
Parameters

doubles	the initial contents of this sequence

8.179 LongSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < long >

Inheritance diagram for LongSeq:



Public Member Functions

- **LongSeq** ()

  Constructs an empty sequence of long integers with an initial maximum of zero.
- **LongSeq** (int initialMaximum)

  Constructs an empty sequence of long integers with the given initial maximum.
- **LongSeq** (long[ ] longs)

  Constructs a new sequence containing the given longs.
- boolean **addAllLong** (long[ ] elements, int offset, int length)

  Append `length` *elements from the given array to this sequence, starting at index* `offset` *in that array.*
- boolean **addAllLong** (long[ ] elements)
- void **addLong** (long element)

  Append the element to the end of the sequence.
- void **addLong** (int index, long element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- long **getLong** (int index)

  Returns the long at the given index.
- long **setLong** (int index, long element)

  Set the new long at the given index and return the old long.
- void **setLong** (int dstIndex, long[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.
- long[ ] **toArrayLong** (long[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  A wrapper for ***getLong(int)*** *(*p. *1292)* *that return a java.lang.Long.*
- Object **set** (int index, Object element)

  A wrapper for ***setLong()*** *(*p. *1292)**.*
- void **add** (int index, Object element)

  A wrapper for addLong(int, int).

8.179.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < long >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

long

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.179.2 Constructor & Destructor Documentation

8.179.2.1 LongSeq() [1/3]

```
LongSeq ( )
```

Constructs an empty sequence of long integers with an initial maximum of zero.

8.179.2.2 LongSeq() [2/3]

```
LongSeq (
            int initialMaximum )
```

Constructs an empty sequence of long integers with the given initial maximum.

8.179.2.3 LongSeq() [3/3]

```
LongSeq (
            long[] longs )
```

Constructs a new sequence containing the given longs.

Parameters

longs	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **LongSeq.addAllLong()**.

8.179.3 Member Function Documentation

8.179.3.1 addAllLong() [1/2]

```
boolean addAllLong (
            long[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **LongSeq.addAllLong()**, and **LongSeq.LongSeq()**.

8.179.3.2 addAllLong() [2/2]

```
boolean addAllLong (
            long[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **LongSeq.addAllLong()**.

8.179.3.3 addLong() [1/2]

```
void addLong (
            long element )
```

Append the element to the end of the sequence.

Referenced by **LongSeq.add()**.

8.179.3.4 addLong() [2/2]

```
void addLong (
             int index,
             long element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.179.3.5 getLong()

```
long getLong (
             int index )
```

Returns the long at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **LongSeq.get()**, **DynamicData.get_ulong_seq()**, and **DynamicData.set_uint_seq()**.

8.179.3.6 setLong() [1/2]

```
long setLong (
             int index,
             long element )
```

Set the new long at the given index and return the old long.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **LongSeq.set()**.

8.179.3.7 setLong() [2/2]

```
void setLong (
            int dstIndex,
            long[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.179.3.8 toArrayLong()

```
long[] toArrayLong (
            long[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.179.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 1295), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_long_seq()**, **DynamicData.get_uint_seq()**, **DynamicData.set_long_seq()**, and **DynamicData.set_uint_seq()**.

8.179.3.10 get()

```
Object get (
             int index )
```

A wrapper for **getLong(int)** (p. 1292) that return a java.lang.Long.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **LongSeq.getLong()**.

8.179.3.11 set()

```
Object set (
             int index,
             Object element )
```

A wrapper for **setLong()** (p. 1292).

Exceptions

ClassCastException	if the element is not of type Long.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **LongSeq.setLong()**.

Referenced by **DynamicData.get_uint_seq()**, and **DynamicData.set_ulong_seq()**.

8.179.3.12 add()

```
void add (
            int index,
            Object element )
```

A wrapper for addLong(int, int).

Exceptions

ClassCastException	if the element is not of type Long.

See also

java.util.List::add(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **LongSeq.addLong()**.

8.180 MonitoringDedicatedParticipantSettings Class Reference

Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.

Inherits Struct.

Public Member Functions

- **MonitoringDedicatedParticipantSettings** ()

  Constructor.
- **MonitoringDedicatedParticipantSettings** (**MonitoringDedicatedParticipantSettings** src)

  Copy constructor.

Public Attributes

- boolean **enable**

  *Enables the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.*
- int **domain_id**

  *The domain ID used in the creation of RTI Monitoring Library 2.0 **com.rti.dds.domain.DomainParticipant** (p. 670).*
- String **participant_qos_profile_name**

  *The fully qualified name of the profile used to configure the **com.rti.dds.domain.DomainParticipant** (p. 670) that will be used to distribute telemetry data.*
- final **StringSeq collector_initial_peers**

  *Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **com.rti.dds.infrastructure.MonitoringDistributionSettings.dedicated_participant** (p. 1300).*

8.180.1 Detailed Description

Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.

8.180.2 Constructor & Destructor Documentation

8.180.2.1 MonitoringDedicatedParticipantSettings() [1/2]

```
MonitoringDedicatedParticipantSettings ( )
```

Constructor.

8.180.2.2 MonitoringDedicatedParticipantSettings() [2/2]

```
MonitoringDedicatedParticipantSettings (
            MonitoringDedicatedParticipantSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(p. 156)* Source object.

References **MonitoringDedicatedParticipantSettings.collector_initial_peers**, **MonitoringDedicatedParticipant↩ Settings.domain_id**, **MonitoringDedicatedParticipantSettings.enable**, and **MonitoringDedicatedParticipant↩ Settings.participant_qos_profile_name**.

8.180.3 Member Data Documentation

8.180.3.1 enable

```
boolean enable
```

Enables the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.

Setting this value to com.rti.dds.infrastructure.false is not currently supported.

[default] com.rti.dds.infrastructure.true

Referenced by **MonitoringDedicatedParticipantSettings.MonitoringDedicatedParticipantSettings()**.

8.180.3.2 domain_id

```
int domain_id
```

The domain ID used in the creation of RTI Monitoring Library 2.0 **com.rti.dds.domain.DomainParticipant** (p. 670).

[default] 2

Referenced by **MonitoringDedicatedParticipantSettings.MonitoringDedicatedParticipantSettings()**.

8.180.3.3 participant_qos_profile_name

```
String participant_qos_profile_name
```

The fully qualified name of the profile used to configure the **com.rti.dds.domain.DomainParticipant** (p. 670) that will be used to distribute telemetry data.

If null (the default value) then RTI Monitoring Library 2.0 uses **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_GENERIC_MONITORING2** (p. 189).

[default] null

Referenced by **MonitoringDedicatedParticipantSettings.MonitoringDedicatedParticipantSettings()**.

8.180.3.4 collector_initial_peers

```
final StringSeq collector_initial_peers
```

Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the **com.rti.dds.infrastructure.MonitoringDistributionSettings.dedicated_participant** (p. 1300).

These initial peers should correspond with the RTI Observability Collector Service with which RTI Monitoring Library 2.0 has to communicate. The collector_initial_peers works the same as initial_peers for other DomainParticipants, except that it allows you to easily specify the initial peer(s) for the RTI Monitoring Library 2.0 **com.rti.dds.domain.DomainParticipant** (p. 670), which usually has different initial peer(s) than those used by your application.

If no collector_initial_peers are specified, or if it is explicitly set to an empty list, the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 668) list of **com.rti.dds.infrastructure.MonitoringDedicatedParticipantSettings.participant_qos_profile_name** (p. 1297) will be used as the initial peers of **com.rti.dds.infrastructure.MonitoringDistributionSettings.dedicated_participant** (p. 1300).

[default] An empty sequence.

See also

com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 668) for further information about initial peers.

Referenced by **MonitoringDedicatedParticipantSettings.MonitoringDedicatedParticipantSettings()**.

8.181 MonitoringDistributionSettings Class Reference

Configures the distribution of telemetry data.

Inherits Struct.

Public Member Functions

- **MonitoringDistributionSettings** ()

  Constructor.
- **MonitoringDistributionSettings** (**MonitoringDistributionSettings** src)

  Copy constructor.

Public Attributes

- String **publisher_qos_profile_name**

  The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
- **MonitoringDedicatedParticipantSettings dedicated_participant**

  *Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.*
- **MonitoringPeriodicDistributionSettings periodic_settings**

  Configures the distribution of periodic metrics.
- **MonitoringEventDistributionSettings event_settings**

  Configures the distribution of event metrics.
- **MonitoringLoggingDistributionSettings logging_settings**

  Configures the distribution of logging messages.

8.181.1 Detailed Description

Configures the distribution of telemetry data.

8.181.2 Constructor & Destructor Documentation

8.181.2.1 MonitoringDistributionSettings() [1/2]

```
MonitoringDistributionSettings ( )
```

Constructor.

8.181.2.2 MonitoringDistributionSettings() [2/2]

```
MonitoringDistributionSettings (
            MonitoringDistributionSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(p. 156)* Source object.

References **MonitoringDistributionSettings.dedicated_participant**, **MonitoringDistributionSettings.event_↩ settings**, **MonitoringDistributionSettings.logging_settings**, **MonitoringDistributionSettings.periodic_settings**, and **MonitoringDistributionSettings.publisher_qos_profile_name**.

8.181.3 Member Data Documentation

8.181.3.1 publisher_qos_profile_name

```
String publisher_qos_profile_name
```

The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

There is one Publisher for each telemetry data **com.rti.dds.topic.Topic** (p. 1807): com.rti.ndds.utility.MONITORING↩ _PERIODIC_TOPIC_NAME, com.rti.ndds.utility.MONITORING_EVENT_TOPIC_NAME, and com.rti.ndds.utility.↩ MONITORING_LOGGING_TOPIC_NAME.

If null (the default value) then RTI Monitoring Library 2.0 uses **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_GENERIC_MONITORING2** (p. 189).

[default] null

Referenced by **MonitoringDistributionSettings.MonitoringDistributionSettings()**.

8.181.3.2 dedicated_participant

MonitoringDedicatedParticipantSettings dedicated_participant

Configures the use of a dedicated **com.rti.dds.domain.DomainParticipant** (p. 670) to distribute the RTI Connext application telemetry data.

Referenced by **MonitoringDistributionSettings.MonitoringDistributionSettings()**.

8.181.3.3 periodic_settings

MonitoringPeriodicDistributionSettings periodic_settings

Configures the distribution of periodic metrics.

Referenced by **MonitoringDistributionSettings.MonitoringDistributionSettings()**.

8.181.3.4 event_settings

MonitoringEventDistributionSettings event_settings

Configures the distribution of event metrics.

Referenced by **MonitoringDistributionSettings.MonitoringDistributionSettings()**.

8.181.3.5 logging_settings

MonitoringLoggingDistributionSettings logging_settings

Configures the distribution of logging messages.

Referenced by **MonitoringDistributionSettings.MonitoringDistributionSettings()**.

8.182 MonitoringEventDistributionSettings Class Reference

Configures the distribution of event metrics.

Inherits Struct.

Public Member Functions

- **MonitoringEventDistributionSettings** ()
  Constructor.
- **MonitoringEventDistributionSettings** (**MonitoringEventDistributionSettings** src)
  Copy constructor.

Public Attributes

- int **concurrency_level**
  Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
- String **datawriter_qos_profile_name**
  *The fully qualified name of the profile used to configure the **com.rti.dds.publication.DataWriter** (p. 553) that distributes event metrics.*
- **ThreadSettings_t thread** = new **ThreadSettings_t**()
  The settings of the event metric thread.
- **Duration_t publication_period** = new **Duration_t**(1,0)
  Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

8.182.1 Detailed Description

Configures the distribution of event metrics.

Event metrics are provided to RTI Monitoring Library 2.0 when they change.

For example, if the liveliness of a **com.rti.dds.publication.DataWriter** (p. 553) is lost, a matching **com.rti.dds.subscription.DataReader** (p. 450) will push the new value of **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1239) to RTI Monitoring Library 2.0 so that the liveliness status change can be distributed.

There are three kinds of event metrics:

- **Configuration metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to QoS policies.
- **Status metrics:** Provided to RTI Monitoring Library 2.0 by pushing changes to the event statuses such as **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1239).
- **Resource metrics:** Provided to RTI Monitoring Library 2.0 when a resource (such as **com.rti.dds.publication.DataWriter** (p. 553)) is created or deleted.

The event metrics that will be distributed for an observable resource can be configured with **com.rti.dds.infrastructure.MonitoringTelemetryData.metrics** (p. 1317).

8.182.2 Constructor & Destructor Documentation

8.182.2.1 MonitoringEventDistributionSettings() [1/2]

```
MonitoringEventDistributionSettings ( )
```

Constructor.

8.182.2.2 MonitoringEventDistributionSettings() [2/2]

```
MonitoringEventDistributionSettings (
            MonitoringEventDistributionSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Source object.

8.182.3 Member Data Documentation

8.182.3.1 concurrency_level

```
int concurrency_level
```

Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the event metrics pushed to RTI Monitoring Library 2.0 will be stored in a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for event metrics, each queue protected by its own mutex. Each resource (e.g, a **com.rti.dds.subscription.DataReader** (p. 450)) will be associated with one of the queues when the resource is registered with RTI Monitoring Library 2.0. Therefore, all the event metrics for a single resource always go to the same queue.

The event metrics for two resources associated with different event queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The event metrics added to the event queues are processed by a single thread configured using **com.rti.dds.↩ infrastructure.MonitoringEventDistributionSettings.thread** (p. 1303).

[default] 5

[range] [1, 100]

8.182.3.2 datawriter_qos_profile_name

```
String datawriter_qos_profile_name
```

The fully qualified name of the profile used to configure the **com.rti.dds.publication.DataWriter** (p. 553) that distributes event metrics.

The **com.rti.dds.publication.DataWriter** (p. 553) Topic is com.rti.ndds.utility.MONITORING_EVENT_TOPIC_NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_GENERIC_MONITORING2** (p. 189).

[default] null

8.182.3.3 thread

```
ThreadSettings_t thread = new  ThreadSettings_t()
```

The settings of the event metric thread.

The event metric thread periodically publishes the event metrics pushed into RTI Monitoring Library 2.0 event metric queues after they change their values.

The thread runs at the period configured using **com.rti.dds.infrastructure.MonitoringEventDistributionSettings.↩ publication_period** (p. 1303).

[default] DDS_THREAD_SETTINGS_DEFAULT

8.182.3.4 publication_period

```
Duration_t publication_period = new  Duration_t(1,0)
```

Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

With a period of 0 seconds, changes to event metrics will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

8.183 MonitoringLoggingDistributionSettings Class Reference

Configures the distribution of log messages.

Inherits Struct.

Public Member Functions

- **MonitoringLoggingDistributionSettings** ()

  Constructor.
- **MonitoringLoggingDistributionSettings** (**MonitoringLoggingDistributionSettings** src)

  Copy constructor.

Public Attributes

- int **concurrency_level**

  Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
- int **max_historical_logs**

  The number of log messages that RTI Monitoring Library 2.0 will keep as history.
- String **datawriter_qos_profile_name**

  *The fully qualified name of the profile used to configure the **com.rti.dds.publication.DataWriter** (p. 553) that distributes log messages.*
- **ThreadSettings_t thread** = new **ThreadSettings_t**()

  The settings of the logging thread.
- **Duration_t publication_period** = new **Duration_t**(1,0)

  Period at which the logging thread publishes log messages.

8.183.1 Detailed Description

Configures the distribution of log messages.

Log messages are pushed into RTI Monitoring Library 2.0 and published by the logging thread.

The logging thread only publishes a log message with a Syslog level smaller than or equal to the forwarding level of the **com.rti.ndds.config.LogFacility** (p. 1265) associated with the log message.

The default value of the forwarding level for all facilities is **com.rti.ndds.config.SyslogVerbosity.NDDS_CONFIG↩ _SYSLOG_VERBOSITY_WARNING** (p. 1780). This value can be changed with the **com.rti.dds.infrastructure.↩ MonitoringTelemetryData.logs** (p. 1317) QoS Policy or by sending a command to RTI Monitoring Library 2.0.

In this release, commands can only be sent from the RTI Observability Dashboards.

RTI Monitoring Library 2.0 can be configured to keep a history of log messages for later distribution when a log snapshot is requested by a RTI Observability Collector Service.

The log messages maintained in the history are the last 'n' messages published by the logging thread (where 'n' is the value of **com.rti.dds.infrastructure.MonitoringLoggingDistributionSettings.max_historical_logs** (p. 1305)).

8.183.2 Constructor & Destructor Documentation

8.183.2.1 MonitoringLoggingDistributionSettings() [1/2]

```
MonitoringLoggingDistributionSettings ( )
```

Constructor.

8.183.2.2 MonitoringLoggingDistributionSettings() [2/2]

```
MonitoringLoggingDistributionSettings (
            MonitoringLoggingDistributionSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Source object.

8.183.3 Member Data Documentation

8.183.3.1 concurrency_level

```
int concurrency_level
```

Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.

With a concurrency_level of one, all the log messages pushed to RTI Monitoring Library 2.0 will be stored into a single queue protected by a single mutex.

With a concurrency_level of 'n', RTI Monitoring Library 2.0 will create 'n' queues for log messages, each queue protected by its own mutex.

The log messages generated by a single thread will always be pushed to the same queue. The log messages for two threads associated with different log queues can be pushed in parallel. This is why a higher concurrency_level provides more concurrency.

The log messages added to the log queues are processed and published by a single thread configured using **com.↩ rti.dds.infrastructure.MonitoringLoggingDistributionSettings.thread** (p. 1306).

[default] 5

[range] [1, 100]

8.183.3.2 max_historical_logs

```
int max_historical_logs
```

The number of log messages that RTI Monitoring Library 2.0 will keep as history.

RTI Monitoring Library 2.0 will keep as history the last `max_historical_logs` published messages.

A value of 0 means that RTI Monitoring Library 2.0 should not keep any history.

[default] 128

8.183.3.3 datawriter_qos_profile_name

```
String datawriter_qos_profile_name
```

The fully qualified name of the profile used to configure the **com.rti.dds.publication.DataWriter** (p. 553) that distributes log messages.

The **com.rti.dds.publication.DataWriter** (p. 553) Topic is com.rti.ndds.utility.MONITORING_LOGGING_TOPIC_↩ NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **com.rti.dds.infrastructure.BuiltinQosProfiles.**↩ **PROFILE_GENERIC_MONITORING2** (p. 189).

[default] null

8.183.3.4 thread

```
ThreadSettings_t thread = new  ThreadSettings_t()
```

The settings of the logging thread.

The logging thread periodically publishes the log messages pushed into RTI Monitoring Library 2.0 log message queues after they are generated.

The thread runs at the period configured using **com.rti.dds.infrastructure.MonitoringLoggingDistribution**↩ **Settings.publication_period** (p. 1306).

[default] DDS_THREAD_SETTINGS_DEFAULT

8.183.3.5 publication_period

```
Duration_t publication_period = new  Duration_t(1,0)
```

Period at which the logging thread publishes log messages.

With a period of 0 seconds, log messages will be published immediately after they are pushed into RTI Monitoring Library 2.0.

[default] 1 second

8.184 MonitoringLoggingForwardingSettings Class Reference

Configures the forwarding levels of log messages for the different **com.rti.ndds.config.LogFacility** (p. 1265).

Inherits Struct.

Public Attributes

- **SyslogVerbosity middleware_forwarding_level**

  *Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 1267) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **SyslogVerbosity security_event_forwarding_level**

  *Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **SyslogVerbosity service_forwarding_level**

  *Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*
- **SyslogVerbosity user_forwarding_level**

  *Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_USER** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.*

8.184.1 Detailed Description

Configures the forwarding levels of log messages for the different **com.rti.ndds.config.LogFacility** (p. 1265).

8.184.2 Member Data Documentation

8.184.2.1 middleware_forwarding_level

```
SyslogVerbosity middleware_forwarding_level
```

Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 1267) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] com.rti.ndds.config.SyslogVerbosity.NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 1780)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1286), may affect performance due to the large amount of messages produced.

8.184.2.2 security_event_forwarding_level

```
SyslogVerbosity security_event_forwarding_level
```

Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] com.rti.ndds.config.SyslogVerbosity.NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 1780)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **com.rti.ndds.↩ config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1286), may affect performance due to the large amount of messages produced.

8.184.2.3 service_forwarding_level

```
SyslogVerbosity service_forwarding_level
```

Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] com.rti.ndds.config.SyslogVerbosity.NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 1780)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **com.rti.ndds.↩ config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1286), may affect performance due to the large amount of messages produced.

8.184.2.4 user_forwarding_level

```
SyslogVerbosity user_forwarding_level
```

Log messages with **com.rti.ndds.config.LogFacility.NDDS_CONFIG_LOG_FACILITY_USER** (p. 1266) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

[default] com.rti.ndds.config.SyslogVerbosity.NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING (p. 1780)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **com.rti.ndds.↩ config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1286), may affect performance due to the large amount of messages produced.

8.185 MonitoringMetricSelection Class Reference

This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

Inherits Struct.

Public Member Functions

- **MonitoringMetricSelection** ()

  Constructor.
- **MonitoringMetricSelection** (**MonitoringMetricSelection** src)

  Copy constructor.

Public Attributes

- String **resource_selection**

  An expression pattern that selects a subset of resources by matching the pattern to the resource names. ***com.rti.dds.infrastructure.MonitoringMetricSelection.enabled_metrics_selection*** *(p. 1310) and* ***com.rti.dds.infrastructure.MonitoringMetricSelection.disabled_metrics_selection*** *(p. 1311) are applied to this subset of resources.*
- **StringSeq enabled_metrics_selection**

  A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by ***com.rti.dds.infrastructure.MonitoringMetricSelection.resource_selection*** *(p. 1310).*
- **StringSeq disabled_metrics_selection**

  A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by ***com.rti.dds.infrastructure.MonitoringMetricSelection.resource_selection*** *(p. 1310).*

8.185.1 Detailed Description

This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

8.185.2 Constructor & Destructor Documentation

8.185.2.1 MonitoringMetricSelection() [1/2]

```
MonitoringMetricSelection ( )
```

Constructor.

8.185.2.2 MonitoringMetricSelection() [2/2]

```
MonitoringMetricSelection (
            MonitoringMetricSelection src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Source object.

References **StringSeq.copy_from()**, **MonitoringMetricSelection.disabled_metrics_selection**, **Monitoring↩ MetricSelection.enabled_metrics_selection**, and **MonitoringMetricSelection.resource_selection**.

8.185.3 Member Data Documentation

8.185.3.1 resource_selection

```
String resource_selection
```

An expression pattern that selects a subset of resources by matching the pattern to the resource names. **com.rti.dds.↩ infrastructure.MonitoringMetricSelection.enabled_metrics_selection** (p. 1310) and **com.rti.dds.infrastructure.↩ MonitoringMetricSelection.disabled_metrics_selection** (p. 1311) are applied to this subset of resources.

Examples of resource expression patterns:

- /applications/myApp/domain_participants/myParticipant
- /applications/∗/domain_participants/∗/subscribers/∗
- /applications/myApp/domain_participants/GUID(1234.5678.4321.8765)
- //myEntity

The first expression refers to a DomainParticipant named "myParticipant" that belongs to an application named "myApp". The second expression applies to all the Subscribers in the system (resource name wildcards follow the POSIX fnmatch syntax). The third expression refers to a DomainParticipant with a specific resource GUID in the "myApp" application. The last expression uses the XPath "//" operator. It matches observable resources named "myEntity" no matter where they are located in the resource hieararchy.

See the Telemetry Data / Resources chapter of the RTI Connext Observability Framework documentation for further information on the observable resource names and expression patterns.

Referenced by **MonitoringMetricSelection.MonitoringMetricSelection()**.

8.185.3.2 enabled_metrics_selection

StringSeq enabled_metrics_selection

A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by **com.rti.dds.infrastructure.MonitoringMetricSelection.resource_selection** (p. 1310).

This sequence is evaluated first, followed by **com.rti.dds.infrastructure.MonitoringMetricSelection.disabled_↩ metrics_selection** (p. 1311). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_∗
- dds_application_∗

The first pattern refers to a specific DataWriter metric (**com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834)). The second pattern refers to all the DataReader **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Referenced by **MonitoringMetricSelection.MonitoringMetricSelection()**.

8.185.3.3 disabled_metrics_selection

StringSeq disabled_metrics_selection

A sequence of POSIX fnmatch patterns that mach the names of the metrics that should not be collected and distributed for the observable resources selected by **com.rti.dds.infrastructure.MonitoringMetricSelection.resource_selection** (p. 1310).

This sequence is evaluated after **com.rti.dds.infrastructure.MonitoringMetricSelection.enabled_metrics_selection** (p. 1310). Therefore, if the same metric is enabled and disabled at the same time, the disablement will be the final result.

The patterns in the sequence are evaluated in order.

Examples of valid patterns:

- dds_data_writer_qos_durability_writer_depth
- dds_data_reader_qos_reliability_∗
- dds_application_∗

The first pattern refers to a specific DataWriter metric (**com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834)). The second pattern refers to all the DataReader **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1526) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connext Observability Framework documentation for further information on the metric names and metric patterns.

Referenced by **MonitoringMetricSelection.MonitoringMetricSelection()**.

8.186 MonitoringMetricSelectionSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) >

Inherits ArraySequence.

8.186.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.MonitoringMetricSelection (p. 1308)

8.187 MonitoringPeriodicDistributionSettings Class Reference

Configures the distribution of periodic metrics.

Inherits Struct.

Public Member Functions

- **MonitoringPeriodicDistributionSettings** ()
  Constructor.
- **MonitoringPeriodicDistributionSettings** (**MonitoringPeriodicDistributionSettings** src)
  Copy constructor.

Public Attributes

- String **datawriter_qos_profile_name**
  The fully qualified name of the profile used to configure the ***com.rti.dds.publication.DataWriter*** *(*p. *553) that distributes periodic metrics.*
- **ThreadSettings_t thread** = new **ThreadSettings_t**()
  The settings of the periodic metric thread.
- **Duration_t polling_period** = new **Duration_t**(5,0)
  Period at which the periodic metric thread polls and publishes the periodic metrics.

8.187.1 Detailed Description

Configures the distribution of periodic metrics.

Periodic metrics change often, and they are polled and published periodically by a thread created by RTI Monitoring Library 2.0.

RTI Monitoring Library 2.0 obtains periodic metrics by polling the current value of periodic statuses such as **com.rti.dds.publication.DataWriterProtocolStatus** (p. 601).

The periodic metrics that will be distributed for an observable resource can be configured with **com.rti.dds.infrastructure.MonitoringTelemetryData.metrics** (p. 1317).

8.187.2 Constructor & Destructor Documentation

8.187.2.1 MonitoringPeriodicDistributionSettings() [1/2]

```
MonitoringPeriodicDistributionSettings ( )
```

Constructor.

8.187.2.2 MonitoringPeriodicDistributionSettings() [2/2]

```
MonitoringPeriodicDistributionSettings (
            MonitoringPeriodicDistributionSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(p. 156)* Source object.

8.187.3 Member Data Documentation

8.187.3.1 datawriter_qos_profile_name

```
String datawriter_qos_profile_name
```

The fully qualified name of the profile used to configure the **com.rti.dds.publication.DataWriter** (p. 553) that distributes periodic metrics.

The **com.rti.dds.publication.DataWriter** (p. 553) Topic is com.rti.ndds.utility.MONITORING_PERIODIC_TOPIC_↩ NAME.

If null (the default value), then RTI Monitoring Library 2.0 uses **com.rti.dds.infrastructure.BuiltinQosProfiles.↩ PROFILE_GENERIC_MONITORING2** (p. 189).

[default] null

8.187.3.2 thread

```
ThreadSettings_t thread = new  ThreadSettings_t()
```

The settings of the periodic metric thread.

The periodic metric thread periodically polls and publishes periodic event metrics.

The thread runs at the period configured using **com.rti.dds.infrastructure.MonitoringPeriodicDistribution↩ Settings.polling_period** (p. 1314).

[default] DDS_THREAD_SETTINGS_DEFAULT

8.187.3.3 polling_period

```
Duration_t polling_period = new  Duration_t(5,0)
```

Period at which the periodic metric thread polls and publishes the periodic metrics.

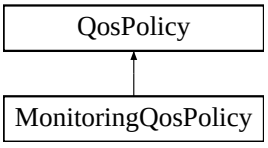
[default] 5 seconds

[range] $>$ 0 seconds

8.188 MonitoringQosPolicy Class Reference

Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

Inheritance diagram for MonitoringQosPolicy:



Public Attributes

- boolean **enable**

  Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
- String **application_name**

  The name of the resource that represents this RTI Connext application.
- **MonitoringDistributionSettings distribution_settings**

  Configures the distribution of telemetry data.
- **MonitoringTelemetryData telemetry_data**

  Configures the telemetry data that will be distributed.

8.188.1 Detailed Description

Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connext telemetry data.

8.188.2 Member Data Documentation

8.188.2.1 enable

```
boolean enable
```

Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.

Note: Enabling and disabling RTI Monitoring Library 2.0 while DDS Entities are being created or deleted is not a safe operation. The entities created while RTI Monitoring Library 2.0 is being enabled may not be monitored. In that case, children entities from that entity (invisible to the library) will not be monitored either.

[default] com.rti.dds.infrastructure.false

8.188.2.2 application_name

```
String application_name
```

The name of the resource that represents this RTI Connext application.

When this member is set to a value other than null , the resource identifier representing this application will be:

/applications/<application_name>

This is the resource identifier that will be used to send commands to this application from the RTI Observability Dashboards.

The application_name should be unique across the RTI Connext system; however, RTI Monitoring Library 2.0 does not currently enforce uniqueness.

When this member is set to null , RTI Monitoring Library 2.0 will automatically assign a resource identifier with this format:

/applications/<host_name:process_id:uuid>

[default] null

8.188.2.3 distribution_settings

```
MonitoringDistributionSettings distribution_settings
```

Configures the distribution of telemetry data.

8.188.2.4 telemetry_data

```
MonitoringTelemetryData telemetry_data
```

Configures the telemetry data that will be distributed.

8.189 MonitoringTelemetryData Class Reference

Configures the telemetry data that will be distributed.

Inherits Struct.

Public Member Functions

- **MonitoringTelemetryData** ()
  Constructor.
- **MonitoringTelemetryData** (**MonitoringTelemetryData** src)
  Copy constructor.

Public Attributes

- **MonitoringMetricSelectionSeq metrics**
  *Sequence of **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.*
- **MonitoringLoggingForwardingSettings logs**
  ***com.rti.dds.infrastructure.MonitoringLoggingForwardingSettings** (p. 1307) containing the **com.rti.ndds.config.↩ SyslogVerbosity** (p. 1778) levels that will be forwarded for the different **com.rti.ndds.config.LogFacility** (p. 1265).*

8.189.1 Detailed Description

Configures the telemetry data that will be distributed.

8.189.2 Constructor & Destructor Documentation

8.189.2.1 MonitoringTelemetryData() [1/2]

```
MonitoringTelemetryData ( )
```

Constructor.

8.189.2.2 MonitoringTelemetryData() [2/2]

```
MonitoringTelemetryData (
            MonitoringTelemetryData src )
```

Copy constructor.

Parameters

src	<<***in***>> *(*p. *156)* Source object.

References **MonitoringTelemetryData.logs**, and **MonitoringTelemetryData.metrics**.

8.189.3 Member Data Documentation

8.189.3.1 metrics

```
MonitoringMetricSelectionSeq metrics
```

Sequence of **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

The different **com.rti.dds.infrastructure.MonitoringMetricSelection** (p. 1308) in the sequence are evaluated in order.

[default] An empty sequence, meaning that no metrics will be collected and distributed for any observable resource.

See also

com.rti.dds.infrastructure.MonitoringEventDistributionSettings (p. 1301) and **com.rti.dds.infrastructure.↩ MonitoringPeriodicDistributionSettings** (p. 1312) for further information on how RTI Monitoring Library 2.↩ 0 distributes **metrics** (p. 1317).

Referenced by **MonitoringTelemetryData.MonitoringTelemetryData()**.

8.189.3.2 logs

```
MonitoringLoggingForwardingSettings logs
```

com.rti.dds.infrastructure.MonitoringLoggingForwardingSettings (p. 1307) containing the **com.rti.ndds.config.↩ SyslogVerbosity** (p. 1778) levels that will be forwarded for the different **com.rti.ndds.config.LogFacility** (p. 1265).

See also

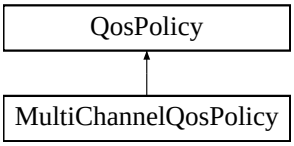
com.rti.dds.infrastructure.MonitoringLoggingDistributionSettings (p. 1303) for further information on how RTI Monitoring Library 2.0 distributes log messages.

Referenced by **MonitoringTelemetryData.MonitoringTelemetryData()**.

8.190 MultiChannelQosPolicy Class Reference

Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Inheritance diagram for MultiChannelQosPolicy:



Public Attributes

- final **ChannelSettingsSeq channels**

  *A sequence of **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.*
- String **filter_name**

  Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

8.190.1 Detailed Description

Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

This QoS policy is used to partition the data published by a **com.rti.dds.publication.DataWriter** (p. 553) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.190.2 Usage

By using this QoS, a **com.rti.dds.publication.DataWriter** (p. 553) can be configured to send data to different multicast groups based on the content of the data. Using syntax similar to those used in Content-Based Filters, you can associate different multicast addresses with filter expressions that operate on the values of the fields within the data. When your application's code calls **com.rti.ndds.example.FooDataWriter.write** (p. 1105), data is sent to any multicast address for which the data passes the filter.

Multi-channel DataWriters can be used to trade off network bandwidth with the unnecessary processing of unwanted data for situations where there are multiple DataReaders that are interested in different subsets of data that come from the same data stream (Topic). For example, in Financial applications, the data stream may be quotes for different stocks at an exchange. Applications usually only want to receive data (quotes) for only a subset of the stocks being traded. In tracking applications, a data stream may carry information on hundreds or thousands of objects being tracked, but again, applications may only be interested in a subset.

The problem is that the most efficient way to deliver data to multiple applications is to use multicast, so that a data value is only sent once on the network for any number of subscribers to the data. However, using multicast, an application will receive *all* of the data sent and not just the data in which it is interested, thus extra CPU time is wasted to throw away unwanted data. With this QoS, you can analyze the data-usage patterns of your applications and optimize network vs. CPU usage by partitioning the data into multiple multicast streams. While network bandwidth is still being conserved by sending data only once using multicast, most applications will only need to listen to a subset of the multicast addresses and receive a reduced amount of unwanted data.

Your system can gain more of the benefits of using multiple multicast groups if your network uses Layer 2 Ethernet switches. Layer 2 switches can be configured to only route multicast packets to those ports that have added membership to specific multicast groups. Using those switches will ensure that only the multicast packets used by applications on a node are routed to the node; all others are filtered-out by the switch.

8.190.3 Member Data Documentation

8.190.3.1 channels

```
final ChannelSettingsSeq channels
```

A sequence of **com.rti.dds.infrastructure.ChannelSettings_t** (p. 411) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.

A sequence length of zero indicates the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318) is not in use.

The sequence length cannot be greater than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQos↩ Policy.channel_seq_max_length** (p. 819).

[default] Empty sequence.

8.190.3.2 filter_name

```
String filter_name
```

Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported:

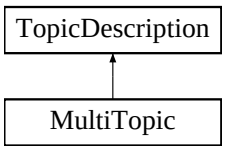
- **com.rti.dds.domain.DomainParticipant.SQLFILTER_NAME** (p. 50)
- **com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50)

[default] com.rti.dds.domain.DomainParticipant.STRINGMATCHFILTER_NAME (p. 50)

8.191 MultiTopic Interface Reference

[Not supported (optional)] <<***interface***>> *(p. 156)* A specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows subscriptions that combine/filter/rearrange data coming from several topics.

Inheritance diagram for MultiTopic:



Public Member Functions

- String **get_subscription_expression** ()

  *Get the expression for this **com.rti.dds.topic.MultiTopic** (p. 1320).*
- void **get_expression_parameters** (**StringSeq** parameters)

  Get the expression parameters.
- void **set_expression_parameters** (**StringSeq** parameters)

  Set the `expression_parameters`.

8.191.1 Detailed Description

[Not supported (optional)] *<<interface>>* *(p. 156)* A specialization of **com.rti.dds.topic.TopicDescription** (p. 1820) that allows subscriptions that combine/filter/rearrange data coming from several topics.

com.rti.dds.topic.MultiTopic (p. 1320) allows a more sophisticated subscription that can select and combine data received from multiple topics into a single resulting type (specified by the inherited `type_name`). The data will then be filtered (selection) and possibly re-arranged (aggregation/projection) according to a `subscription_expression` with parameters `expression_parameters`.

- The `subscription_expression` is a string that identifies the selection and re-arrangement of data from the associated topics. It is similar to an SQL statement where the SELECT part provides the fields to be kept, the FROM part provides the names of the topics that are searched for those fields, and the WHERE clause gives the content filter. The Topics combined may have different types but they are restricted in that the type of the fields used for the NATURAL JOIN operation must be the same.

- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.e. "%n" tokens) in the `subscription_expression`. The number of supplied parameters must fit with the requested values in the `subscription_expression` (i.e. the number of n tokens).

- **com.rti.dds.subscription.DataReader** (p. 450) entities associated with a **com.rti.dds.topic.MultiTopic** (p. 1320) are alerted of data modifications by the usual **com.rti.dds.infrastructure.Listener** (p. 1236) or **com.rti.dds.infrastructure.WaitSet** (p. 1973) / **com.rti.dds.infrastructure.Condition** (p. 429) mechanisms whenever modifications occur to the data associated with any of the topics relevant to the **com.rti.dds.topic.MultiTopic** (p. 1320).

Note that the source for data may not be restricted to a single topic.

com.rti.dds.subscription.DataReader (p. 450) entities associated with a **com.rti.dds.topic.MultiTopic** (p. 1320) may access instances that are "constructed" at the **com.rti.dds.subscription.DataReader** (p. 450) side from the instances written by multiple **com.rti.dds.publication.DataWriter** (p. 553) entities. The **com.rti.dds.topic.MultiTopic** (p. 1320) access instance will begin to exist as soon as all the constituting **com.rti.dds.topic.Topic** (p. 1807) instances are in existence. The `view_state` and `instance_state` is computed from the corresponding states of the constituting instances:

- The `view_state` of the **com.rti.dds.topic.MultiTopic** (p. 1320) instance is com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE if at least one of the constituting instances has `view_state` = com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE. Otherwise, it will be com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE.

- The `instance_state` of the **com.rti.dds.topic.MultiTopic** (p. 1320) instance is **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161) if the `instance_state` of all the constituting **com.rti.dds.topic.Topic** (p. 1807) instances is **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161). It is **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) if at least one of the constituting **com.rti.dds.topic.Topic** (p. 1807) instances is **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161). Otherwise, it is **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).

Queries and Filters Syntax (p. 104) describes the syntax of `subscription_expression` and `expression_parameters`.

8.191.2 Member Function Documentation

8.191.2.1 get_subscription_expression()

```
String get_subscription_expression ( )
```

Get the expression for this **com.rti.dds.topic.MultiTopic** (p. 1320).

The expressions syntax is described in the DDS specification. It is specified when the **com.rti.dds.topic.MultiTopic** (p. 1320) is created.

Returns

`subscription_expression` of the **com.rti.dds.topic.MultiTopic** (p. 1320).

8.191.2.2 get_expression_parameters()

```
void get_expression_parameters (
              StringSeq parameters )
```

Get the expression parameters.

The expressions syntax is described in the DDS specification.

The `parameters` is either specified on the last successful call to **com.rti.dds.topic.MultiTopic.set_expression_↩ parameters** (p. 1322), or if **com.rti.dds.topic.MultiTopic.set_expression_parameters** (p. 1322) was never called, the `parameters` specified when the **com.rti.dds.topic.MultiTopic** (p. 1320) was created.

Parameters

parameters	<<***inout***>> *(p. 156)* Fill in this sequence with the expression parameters. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.191.2.3 set_expression_parameters()

```
void set_expression_parameters (
              StringSeq parameters )
```

Set the `expression_parameters`.

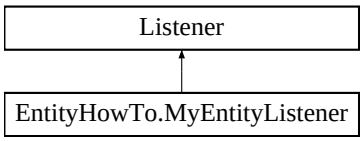
Changes the `expression_parameters` associated with the **com.rti.dds.topic.MultiTopic** (p. 1320).

Parameters

parameters	<<***in***>> *(p. 156)* the filter expression parameters

8.192 EntityHowTo.MyEntityListener Class Reference

Inheritance diagram for EntityHowTo.MyEntityListener:



8.192.1 Detailed Description

{

8.193 NetworkCapture Class Reference

Network Capture APIs.

Static Public Member Functions

- static native boolean **enable** ()
  Enable Network Capture.
- static native boolean **disable** ()
  Disable Network Capture.
- static boolean **set_default_params** (**NetworkCaptureParams** params)
  Set the default Network Capture parameters.
- static native boolean **start** (String filename)
  Start capturing traffic for all DomainParticipants, with the default parameters.
- static boolean **start** (**DomainParticipant** participant, String filename)
  Start capturing traffic for a DomainParticipant, with the default parameters.
- static boolean **start** (String filename, **NetworkCaptureParams** params)
  Start capturing traffic for all DomainParticipants, with the provided parameters.
- static boolean **start** (**DomainParticipant** participant, String filename, **NetworkCaptureParams** params)

Start capturing traffic for a DomainParticipant, with the provided parameters.

- static native boolean **stop** ()

  Stop capturing traffic for all participants.
- static boolean **stop** (**DomainParticipant** participant)

  Stop capturing traffic for a DomainParticipant.
- static native boolean **pause** ()

  Pause capturing traffic for all DomainParticipants.
- static boolean **pause** (**DomainParticipant** participant)

  Pause capturing traffic for a DomainParticipant.
- static native boolean **resume** ()

  Resume capturing traffic for all DomainParticipants.
- static boolean **resume** (**DomainParticipant** participant)

  Resume capturing traffic for a DomainParticipant.

8.193.1 Detailed Description

Network Capture APIs.

8.193.2 Member Function Documentation

8.193.2.1 enable()

```
static native boolean enable ( )  [static]
```

Enable Network Capture.

This method must be called before any other Network Capture method. It must also be called before creating the participants for which we want to capture traffic.

Use this method only for debugging purposes, since it may introduce a significant performance impact.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Caputre related method, including this one.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simulatenously calling **com.↩ rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipant↩ Factory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), **com.↩ rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

See also

com.rti.ndds.utility.NetworkCapture.disable (p. 1324)

8.193.2.2 disable()

```
static native boolean disable ( )  [static]
```

Disable Network Capture.

This method must be the last Network Capture method to be called. It must also be called after deleting the participants for which we captured traffic. Disabling Network Capture without stopping it first is not ok!

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

MT Safety:

UNSAFE. It is not safe to call this method while another thread may be simultaneously calling another Network Capture related method, including this one.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simulatenously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

See also

com.rti.ndds.utility.NetworkCapture.enable (p. 1324)

8.193.2.3 set_default_params()

```
static boolean set_default_params (
              NetworkCaptureParams params )  [static]
```

Set the default Network Capture parameters.

The default parameters are used when Network Capture is started without parameters, i.e., **com.rti.ndds.utility.NetworkCapture.start** (p. 1326).

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

params	<<***in***>> *(*p. *156)*. Configuration parameters that we want to set as defaults.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.start (p. 1326)

com.rti.ndds.utility.NetworkCapture.start(DomainParticipant, String) (p. 1326)

8.193.2.4 start() [1/4]

```
static native boolean start (
            String filename )  [static]
```

Start capturing traffic for all DomainParticipants, with the default parameters.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

filename	<<***in***>> *(*p. *156)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.stop (p. 1329)

com.rti.ndds.utility.NetworkCapture.start(DomainParticipant, String) (p. 1326)

8.193.2.5 start() [2/4]

```
static boolean start (
              DomainParticipant participant,
             String filename )  [static]
```

Start capturing traffic for a DomainParticipant, with the default parameters.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

participant	<<***in***>> *(p. 156).* DomainParticipant for which we want to capture traffic.
filename	<<***in***>> *(p. 156).* The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.stop(DomainParticipant) (p. 1329)

com.rti.ndds.utility.NetworkCapture.start(DDS:DomainParticipant^, System::String^, NetworkCaptureParams)

com.rti.ndds.utility.NetworkCapture.enable (p. 1324)

8.193.2.6 start() [3/4]

```
static boolean start (
             String filename,
              NetworkCaptureParams params )  [static]
```

Start capturing traffic for all DomainParticipants, with the provided parameters.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Perfoms the same function as **com.rti.ndds.utility.NetworkCapture.start** (p. 1326) except that it uses the provided parameters, instead of the default ones.

Parameters

filename	<<***in***>> *(*p. *156)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, the `"_GUID-"` string followed by the decimal representation of bytes 8-11 of the DomainParticipant's GUID, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> *(*p. *156)*. Configuration parameters for the capture.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.stop (p. 1329)

com.rti.ndds.utility.NetworkCapture.start(DDS:DomainParticipant^, System::String^, NetworkCaptureParams)

8.193.2.7 start() [4/4]

```
static boolean start (
              DomainParticipant participant,
             String filename,
              NetworkCaptureParams params )  [static]
```

Start capturing traffic for a DomainParticipant, with the provided parameters.

Precondition

This method requires enabling first Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

participant	<<***in***>> *(*p. *156)*. DomainParticipant for which we want to capture traffic.
filename	<<***in***>> *(*p. *156)*. The name of the output capture file will be based on this input parameter.

In particular, the name for the capture file is the concatenation of the `filename` input parameter, and the file extension (`".pcap"`).

Parameters

params	<<***in***>> *(p. 156).* Parameters for configuring the capture.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.stop(DomainParticipant) (p. 1329)

com.rti.ndds.utility.NetworkCapture.start(DomainParticipant, String) (p. 1326)

8.193.2.8 stop() [1/2]

```
static native boolean stop ( )  [static]
```

Stop capturing traffic for all participants.

Precondition

This method requires enabling first Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

This method can (and must) be called after **com.rti.ndds.utility.NetworkCapture.start** (p. 1326), not **com.rti.ndds.↩ utility.NetworkCapture.start(DomainParticipant, String)** (p. 1326). That is, if we start capturing traffic globally (for all DomainParticipants), we must stop capturing traffic also globally. It is not possible to start capturing traffic for a participant but stop it globally.

It is possible to start capturing globally and then stop capturing for a participant, as long as we eventually stop capturing traffic globally.

We must stop capturing for a participant before deleting it.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.start (p. 1326)

com.rti.ndds.utility.NetworkCapture.stop(DomainParticipant) (p. 1329)

8.193.2.9 stop() [2/2]

```
static boolean stop (
              DomainParticipant participant )  [static]
```

Stop capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

participant	<<***in***>> *(p. 156).* DomainParticipant for which we want to stop capturing traffic.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.start(DomainParticipant, String) (p. 1326)

com.rti.ndds.utility.NetworkCapture.stop (p. 1329)

8.193.2.10 pause() [1/2]

```
static native boolean pause ( )  [static]
```

Pause capturing traffic for all DomainParticipants.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.resume (p. 1331)

com.rti.ndds.utility.NetworkCapture.pause(DomainParticipant) (p. 1330)

8.193.2.11 pause() [2/2]

```
static boolean pause (
             DomainParticipant participant )  [static]
```

Pause capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

participant	<<***in***>> *(p. 156).* DomainParticipant for which we want to pause capturing traffic.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.resume(DomainParticipant) (p. 1331)

com.rti.ndds.utility.NetworkCapture.pause (p. 1330)

8.193.2.12 resume() [1/2]

```
static native boolean resume ( )  [static]
```

Resume capturing traffic for all DomainParticipants.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.pause (p. 1330)

com.rti.ndds.utility.NetworkCapture.resume(DomainParticipant) (p. 1331)

8.193.2.13 resume() [2/2]

```
static boolean resume (
              DomainParticipant participant )  [static]
```

Resume capturing traffic for a DomainParticipant.

Precondition

This method requires first enabling Network Capture. See **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324).

Parameters

participant	<<***in***>> *(p. 156).* DomainParticipant for which we want to resume capturing traffic.

Returns

com.rti.dds.infrastructure.true if success. Otherwise, com.rti.dds.infrastructure.false

See also

com.rti.ndds.utility.NetworkCapture.pause(DomainParticipant) (p. 1330)

com.rti.ndds.utility.NetworkCapture.resume (p. 1331)

8.194 NetworkCaptureContentKind Class Reference

Bitmap used to specify a content type, i.e., a part of the RTPS frame.

Static Public Attributes

- static final int **USER_SERIALIZED_DATA**

  The serialized data coming from a user.
- static final int **ENCRYPTED_DATA**

  The encrypted user data.
- static final int **MASK_DEFAULT**

  *Default mask for **com.rti.ndds.utility.NetworkCaptureContentKind** (p. 1332): do not remove any content.*
- static final int **MASK_NONE**

  The RTPS frames in the capture file will be saved as they are.
- static final int **MASK_ALL**

  The RTPS frames in the capture file will not include user data (either plain or encrypted).

8.194.1 Detailed Description

Bitmap used to specify a content type, i.e., a part of the RTPS frame.

Several values can be combined. Read **com.rti.ndds.utility.NetworkCaptureContentKind** (p. 1332) for typical combinations.

8.194.2 Member Data Documentation

8.194.2.1 USER_SERIALIZED_DATA

```
final int USER_SERIALIZED_DATA  [static]
```

The serialized data coming from a user.

8.194.2.2 ENCRYPTED_DATA

```
final int ENCRYPTED_DATA  [static]
```

The encrypted user data.

8.194.2.3 MASK_DEFAULT

```
final int MASK_DEFAULT  [static]
```

Default mask for **com.rti.ndds.utility.NetworkCaptureContentKind** (p. 1332): do not remove any content.

It is equivalent to **com.rti.ndds.utility.NetworkCaptureContentKind.MASK_NONE** (p. 1333).

[default] Do not remove any content.

8.194.2.4 MASK_NONE

```
final int MASK_NONE  [static]
```

The RTPS frames in the capture file will be saved as they are.

8.194.2.5 MASK_ALL

```
final int MASK_ALL  [static]
```

The RTPS frames in the capture file will not include user data (either plain or encrypted).

Its value is the result of setting the bits for removing user data and removing encrypted data: (**com.rti.ndds.↩ utility.NetworkCaptureContentKind.USER_SERIALIZED_DATA** (p. 1332)) | **com.rti.ndds.utility.NetworkCapture↩ ContentKind.ENCRYPTED_DATA** (p. 1333))

8.195 NetworkCaptureParams Class Reference

Input parameters for starting network capture.

Inherits Struct.

Public Attributes

- **StringSeq transports** = new **StringSeq**()
  List of transports to capture.
- int **dropped_content** = **NetworkCaptureContentKind.MASK_DEFAULT**
  Exclude contents from the capture file.
- int **traffic** = **NetworkCaptureTrafficKind.MASK_DEFAULT**
  Traffic direction to capture.
- boolean **parse_encrypted_content** = false
  If secure traffic should be decrypted or not.
- **ThreadSettings_t checkpoint_thread_settings** = new **ThreadSettings_t**()
  The properties of the checkpoint thread.
- int **frame_queue_size** = 2097152
  Size of the frame queue (Bytes).

8.195.1 Detailed Description

Input parameters for starting network capture.

8.195.2 Member Data Documentation

8.195.2.1 transports

```
StringSeq transports = new  StringSeq()
```

List of transports to capture.

Network Capture will only save RTPS frames if the associated transport protocol is part of this sequence.

[default] Empty sequence initializer. This means that by default all transports will be captured.

8.195.2.2 dropped_content

```
int dropped_content = NetworkCaptureContentKind.MASK_DEFAULT
```

Exclude contents from the capture file.

It accepts values from **com.rti.ndds.utility.NetworkCaptureContentKind** (p. 1332) .

We can choose to exclude user data or encrypted content from the capture file.

[default] No content is excluded - **com.rti.ndds.utility.NetworkCaptureContentKind.MASK_DEFAULT** (p. 1333).

8.195.2.3 traffic

```
int traffic = NetworkCaptureTrafficKind.MASK_DEFAULT
```

Traffic direction to capture.

It accepts values from **com.rti.ndds.utility.NetworkCaptureTrafficKind** (p. 1336) .

[default] Capture both inbound and outbound traffic - **com.rti.ndds.utility.NetworkCaptureTrafficKind.MASK_↩ DEFAULT** (p. 1337).

8.195.2.4 parse_encrypted_content

```
boolean parse_encrypted_content = false
```

If secure traffic should be decrypted or not.

Network Capture supports decryption of RTPS messages and submessages. It does not support decryption of the user data payload (<data_protection_kind> tag in the GovernanceDocument).

[default] com.rti.dds.infrastructure.false

8.195.2.5 checkpoint_thread_settings

```
ThreadSettings_t checkpoint_thread_settings = new ThreadSettings_t()
```

The properties of the checkpoint thread.

The checkpoint thread is a per-participant thread responsible for reading frames from a queue and saving them to disk (as soon as they are produced).

The members that can be configured are:

- **com.rti.dds.infrastructure.ThreadSettings_t.mask** (p. 1793). Default value of (**com.rti.dds.infrastructure.↩ ThreadSettingsKind.THREAD_SETTINGS_PRIORITY_ENFORCE** (p. 1797) | **com.rti.dds.infrastructure.↩ ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1797)).
- **com.rti.dds.infrastructure.ThreadSettings_t.priority** (p. 1793). Platform-dependent - Consult `Platform Notes` for additional details.
- **com.rti.dds.infrastructure.ThreadSettings_t.stack_size** (p. 1793). Platform-dependent - Consult `Platform Notes` for additional details.

8.195.2.6 frame_queue_size

```
int frame_queue_size = 2097152
```

Size of the frame queue (Bytes).

Network Capture enqueues frames before saving them to disk, which takes place in a separate thread.

Network Capture does not block if the queue becomes full. Attempting to enqueue a frame with a full frame queue will fail (frame won't be captured) with a log message.

The size of the queue is dependent on the network traffic (amount of frames that we want to capture) and system resources (how fast we can capture frames).

[default] 2097152 (2MB).

8.196 NetworkCaptureTrafficKind Class Reference

Bitmap used to specify whether we want to capture inbound or outbound traffic.

Static Public Attributes

- static final int **TRAFFIC_OUT**

  Do not capture outbound traffic.
- static final int **TRAFFIC_IN**

  Do not capture inbound traffic.
- static final int **MASK_DEFAULT**

  *Default mask for **com.rti.ndds.utility.NetworkCaptureTrafficKind** (p. 1336).*
- static final int **MASK_NONE**

  Do not capture any traffic.
- static final int **MASK_ALL**

  Capture all traffic (both inbound and outbound).

8.196.1 Detailed Description

Bitmap used to specify whether we want to capture inbound or outbound traffic.

Several values can be combined. Read **com.rti.ndds.utility.NetworkCaptureTrafficKind** (p. 1336) for typical combinations.

8.196.2 Member Data Documentation

8.196.2.1 TRAFFIC_OUT

```
final int TRAFFIC_OUT  [static]
```

Do not capture outbound traffic.

8.196.2.2 TRAFFIC_IN

```
final int TRAFFIC_IN  [static]
```

Do not capture inbound traffic.

8.196.2.3 MASK_DEFAULT

```
final int MASK_DEFAULT  [static]
```

Default mask for **com.rti.ndds.utility.NetworkCaptureTrafficKind** (p. 1336).

It is equivalent to **com.rti.ndds.utility.NetworkCaptureTrafficKind.MASK_ALL** (p. 1337).

[default] Capture all traffic: inbound and outbound.

8.196.2.4 MASK_NONE

```
final int MASK_NONE  [static]
```

Do not capture any traffic.

8.196.2.5 MASK_ALL

```
final int MASK_ALL  [static]
```

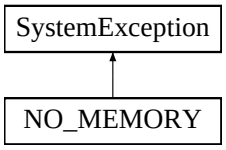
Capture all traffic (both inbound and outbound).

The value is equal to setting both the input and output bits of the mask: (**com.rti.ndds.utility.NetworkCaptureTraffic↩ Kind.TRAFFIC_OUT** (p. 1336) | **com.rti.ndds.utility.NetworkCaptureTrafficKind.TRAFFIC_IN** (p. 1337)).

8.197 NO_MEMORY Class Reference

Exception thrown when there is not enough memory for a dynamic memory allocation.

Inheritance diagram for NO_MEMORY:



8.197.1 Detailed Description

Exception thrown when there is not enough memory for a dynamic memory allocation.

8.198 ObjectHolder Class Reference

<<***extension***>> *(*p. *155)* Holder of object instance

Public Attributes

- Object **value** = null
  Instance of Object embedded in ***ObjectHolder*** *(*p. *1338).*

8.198.1 Detailed Description

<<***extension***>> *(*p. *155)* Holder of object instance

Holder of object instance. Can be used as an output parameter in a method for non-primitive types.

8.198.2 Member Data Documentation

8.198.2.1 value

```
Object value = null
```

Instance of Object embedded in **ObjectHolder** (p. 1338).

8.199 OfferedDeadlineMissedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of the number of times the **com.rti.dds.publication.DataWriter** (p. 553) failed to write within its offered deadline.*
- int **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- final **InstanceHandle_t last_instance_handle**

  *Handle to the last instance in the **com.rti.dds.publication.DataWriter** (p. 553) for which an offered deadline was missed.*

8.199.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

The deadline that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632) was not respected for a specific instance.

8.199.2 Member Data Documentation

8.199.2.1 total_count

```
int total_count
```

Total cumulative count of the number of times the **com.rti.dds.publication.DataWriter** (p. 553) failed to write within its offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

8.199.2.2 total_count_change

```
int total_count_change
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.199.2.3 last_instance_handle

```
final InstanceHandle_t last_instance_handle
```

Handle to the last instance in the **com.rti.dds.publication.DataWriter** (p. 553) for which an offered deadline was missed.

8.200 OfferedIncompatibleQosStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative number of times the concerned **com.rti.dds.publication.DataWriter** (p. 553) discovered a **com.rti.dds.subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807), common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).*
- int **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- **QosPolicyId_t last_policy_id**

  *The **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- final **QosPolicyCountSeq policies**

  *A list containing for each policy the total number of times that the concerned **com.rti.dds.publication.DataWriter** (p. 553) discovered a **com.rti.dds.subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807) and common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).*

8.200.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

The qos policy value was incompatible with what was requested.

8.200.2 Member Data Documentation

8.200.2.1 total_count

```
int total_count
```

Total cumulative number of times the concerned **com.rti.dds.publication.DataWriter** (p. 553) discovered a **com.↩ rti.dds.subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807), common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553).

8.200.2.2 total_count_change

```
int total_count_change
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.200.2.3 last_policy_id

```
QosPolicyId_t last_policy_id
```

The **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.200.2.4 policies

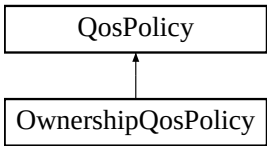
```
final QosPolicyCountSeq policies
```

A list containing for each policy the total number of times that the concerned **com.rti.dds.publication.DataWriter** (p. 553) discovered a **com.rti.dds.subscription.DataReader** (p. 450) for the same **com.rti.dds.topic.Topic** (p. 1807) and common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.↩ DataWriter** (p. 553).

8.201 OwnershipQosPolicy Class Reference

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Inheritance diagram for OwnershipQosPolicy:



Public Attributes

- **OwnershipQosPolicyKind kind**
  The kind of ownership.

8.201.1 Detailed Description

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 553) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

See also

OWNERSHIP_STRENGTH (p. 245)

8.201.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 245), this QoS policy specifies if **com.rti.dds.subscription.DataReader** (p. 450) entities can receive updates to the same instance (identified by its key) from multiple **com.rti.dds.publication.↩ DataWriter** (p. 553) entities at the same time.

There are two kinds of ownership, selected by the setting of the `kind:` SHARED and EXCLUSIVE.

8.201.2.1 SHARED ownership

com.rti.dds.infrastructure.OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS (p. 1347) indicates that RTI Connext does not enforce unique ownership for each instance. In this case, multiple writers can update the same data type instance. The subscriber to the **com.rti.dds.topic.Topic** (p. 1807) will be able to access modifications from all **com.rti.dds.publication.DataWriter** (p. 553) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 267) or **HISTORY** (p. 237) policy). In any case, there is no "filtering" of modifications made based on the identity of the **com.rti.dds.publication.DataWriter** (p. 553) that causes the modification.

8.201.2.2 EXCLUSIVE ownership

com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS (p. 1348) indicates that each instance of a data type can only be modified by one **com.rti.dds.publication.DataWriter** (p. 553). In other words, at any point in time, a single **com.rti.dds.publication.DataWriter** (p. 553) owns each instance and is the only one whose modifications will be visible to the **com.rti.dds.subscription.DataReader** (p. 450) objects. The owner is determined by selecting the **com.rti.dds.publication.DataWriter** (p. 553) with the highest value of the **com.rti.dds.↩ infrastructure.OwnershipStrengthQosPolicy.value** (p. 1349) that is currently alive, as defined by the **LIVELINESS** (p. 239) policy, and has not violated its **DEADLINE** (p. 217) contract with regards to the data instance.

Ownership can therefore change as a result of:

- a **com.rti.dds.publication.DataWriter** (p. 553) in the system with a higher value of the strength that modifies the instance,
- a change in the strength value of the **com.rti.dds.publication.DataWriter** (p. 553) that owns the instance, and
- a change in the liveliness of the **com.rti.dds.publication.DataWriter** (p. 553) that owns the instance.
- a deadline with regards to the instance that is missed by the **com.rti.dds.publication.DataWriter** (p. 553) that owns the instance.

The behavior of the system is as if the determination was made independently by each **com.rti.dds.subscription.↩ DataReader** (p. 450). Each **com.rti.dds.subscription.DataReader** (p. 450) may detect the change of ownership at a different time. It is not a requirement that at a particular point in time all the **com.rti.dds.subscription.DataReader** (p. 450) objects for that **com.rti.dds.topic.Topic** (p. 1807) have a consistent picture of who owns each instance.

It is also not a requirement that the **com.rti.dds.publication.DataWriter** (p. 553) objects are aware of whether they own a particular instance. There is no error or notification given to a **com.rti.dds.publication.DataWriter** (p. 553) that modifies an instance it does not currently own.

The requirements are chosen to (a) preserve the decoupling of publishers and subscriber, and (b) allow the policy to be implemented efficiently.

It is possible that multiple **com.rti.dds.publication.DataWriter** (p. 553) objects with the same strength modify the same instance. If this occurs RTI Connext will pick one of the **com.rti.dds.publication.DataWriter** (p. 553) objects as the owner. It is not specified how the owner is selected. However, the algorithm used to select the owner guarantees that all **com.rti.dds.subscription.DataReader** (p. 450) objects will make the same choice of the particular **com.rti.dds.↩ publication.DataWriter** (p. 553) that is the owner. It also guarantees that the owner remains the same until there is a change in strength, liveliness, the owner misses a deadline on the instance, or a new **com.rti.dds.publication.Data↩ Writer** (p. 553) with higher same strength, or a new **com.rti.dds.publication.DataWriter** (p. 553) with same strength that should be deemed the owner according to the policy of the Service, modifies the instance.

Exclusive ownership is on an instance-by-instance basis. That is, a subscriber can receive values written by a lower strength **com.rti.dds.publication.DataWriter** (p. 553) as long as they affect instances whose values have not been set by the higher-strength **com.rti.dds.publication.DataWriter** (p. 553).

8.201.3 Compatibility

The value of the **com.rti.dds.infrastructure.OwnershipQosPolicyKind** (p. 1347) offered must exactly match the one requested or else they are considered incompatible.

8.201.4 Relationship between registration, liveliness and ownership

The need for registering/unregistering instances stems from two use cases:

- Ownership resolution on redundant systems
- Detection of loss in topological connectivity

These two use cases also illustrate the semantic differences between the **com.rti.ndds.example.FooDataWriter.↩ unregister_instance** (p. 1100) and **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111).

8.201.4.1 Ownership Resolution on Redundant Systems

It is expected that users may use DDS to set up redundant systems where multiple **com.rti.dds.publication.DataWriter** (p. 553) entities are "capable" of writing the same instance. In this situation, the **com.rti.dds.publication.DataWriter** (p. 553) entities are configured such that:

- Either both are writing the instance "constantly"
- Or else they use some mechanism to classify each other as "primary" and "secondary", such that the primary is the only one writing, and the secondary monitors the primary and only writes when it detects that the primary "writer" is no longer writing.

Both cases above use the **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1348) and arbitrate themselves by means of the **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1348). Regardless of the scheme, the desired behavior from the **com.rti.dds.subscription.DataReader** (p. 450) point of view is that **com.rti.dds.subscription.DataReader** (p. 450) normally receives data from the primary unless the "primary" writer stops writing, in which case the **com.rti.dds.subscription.DataReader** (p. 450) starts to receive data from the secondary **com.rti.dds.publication.DataWriter** (p. 553).

This approach requires some mechanism to detect that a **com.rti.dds.publication.DataWriter** (p. 553) (the primary) is no longer "writing" the data as it should. There are several reasons why this may happen and all must be detected (but not necessarily distinguished):

crash The writing process is no longer running (e.g. the whole application has crashed)

connectivity loss Connectivity to the writing application has been lost (e.g. network disconnection)

application fault The application logic that was writing the data is faulty and has stopped calling **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

Arbitrating from a **com.rti.dds.publication.DataWriter** (p. 553) to one of a higher strength is simple and the decision can be taken autonomously by the **com.rti.dds.subscription.DataReader** (p. 450). Switching ownership from a higher strength **com.rti.dds.publication.DataWriter** (p. 553) to one of a lower strength **com.rti.dds.publication.DataWriter** (p. 553) requires that the **com.rti.dds.subscription.DataReader** (p. 450) can make a determination that the stronger **com.rti.dds.publication.DataWriter** (p. 553) is "no longer writing the instance".

8.201.4.1.1 Case where the data is periodically updated This determination is reasonably simple when the data is being written periodically at some rate. The **com.rti.dds.publication.DataWriter** (p. 553) simply states its offered **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632) (maximum interval between updates) and the **com.rti.dds.subscription.DataReader** (p. 450) automatically monitors that the **com.rti.dds.publication.DataWriter** (p. 553) indeed updates the instance at least once per **com.rti.dds.infrastructure.DeadlineQosPolicy.period** (p. 634). If the deadline is missed, the **com.rti.dds.subscription.DataReader** (p. 450) considers the **com.rti.dds.publication.DataWriter** (p. 553) "not alive" and automatically gives ownership to the next highest-strength **com.rti.dds.publication.DataWriter** (p. 553) that *is* alive.

8.201.4.1.2 Case where data is not periodically updated The case where the **com.rti.dds.publication.DataWriter** (p. 553) is not writing data periodically is also a very important use-case. Since the instance is not being updated at any fixed period, the "deadline" mechanism cannot be used to determine ownership. The liveliness solves this situation. Ownership is maintained while the **com.rti.dds.publication.DataWriter** (p. 553) is "alive" and for the **com.rti.dds.publication.DataWriter** (p. 553) to be alive it must fulfill its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243) contract. The different means to renew liveliness (automatic, manual) combined by the implied renewal each time data is written handle the three conditions above [crash], [connectivity loss], and [application fault]. Note that to handle [application fault], LIVELINESS must be com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS. The **com.rti.dds.publication.DataWriter** (p. 553) can retain ownership by periodically writing data or else calling assert_liveliness if it has no data to write. Alternatively if only protection against [crash] or [connectivity loss] is desired, it is sufficient that some task on the **com.rti.dds.publication.DataWriter** (p. 553) process periodically writes data or calls **com.rti.dds.domain.DomainParticipant.assert_liveliness** (p. 727). However, this scenario requires that the **com.rti.dds.subscription.DataReader** (p. 450) knows what instances are being "written" by the **com.rti.dds.publication.DataWriter** (p. 553). That is the only way that the **com.rti.dds.subscription.DataReader** (p. 450) deduces the ownership of specific instances from the fact that the **com.rti.dds.publication.DataWriter** (p. 553) is still "alive". Hence the need for the **com.rti.dds.publication.DataWriter** (p. 553) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **com.rti.dds.publication.DataWriter** (p. 553) writes the instance, "unregistration," in general, cannot. Similar reasoning will lead to the fact that unregistration will also require a message to be sent to the **com.rti.dds.subscription.DataReader** (p. 450).

8.201.4.2 Detection of Loss in Topological Connectivity

There are applications that are designed in such a way that their correct operation requires some minimal topological connectivity, that is, the writer needs to have a minimum number of readers or alternatively the reader must have a minimum number of writers.

A common scenario is that the application does not start doing its logic until it knows that some specific writers have the minimum configured readers (e.g the alarm monitor is up).

A *more* common scenario is that the application logic will wait until some writers appear that can provide some needed source of information (e.g. the raw sensor data that must be processed).

Furthermore, once the application is running it is a requirement that this minimal connectivity (from the source of the data) is monitored and the application informed if it is ever lost. For the case where data is being written periodically, the **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632) and the on_deadline_missed listener provides the notification. The case where data is not periodically updated requires the use of the **com.rti.dds.infrastructure.LivelinessQos↩ Policy** (p. 1243) in combination with register_instance/unregister_instance to detect whether the "connectivity" has been lost, and the notification is provided by means of **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE** (p. 1161).

In terms of the required mechanisms, the scenario is very similar to the case of maintaining ownership. In both cases, the reader needs to know whether a writer is still "managing the current value of an instance" even though it is not continually writing it and this knowledge requires the writer to keep its liveliness plus some means to know which instances the writer is currently "managing" (i.e. the registered instances).

8.201.4.3 Semantic Difference between unregister_instance and dispose

com.rti.ndds.example.FooDataWriter.dispose (p. 1111) is semantically different from **com.rti.ndds.example.Foo↩ DataWriter.unregister_instance** (p. 1100). **com.rti.ndds.example.FooDataWriter.dispose** (p. 1111) indicates that the data instance no longer exists (e.g. a track that has disappeared, a simulation entity that has been destroyed, a record entry that has been deleted, etc.) whereas **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **com.rti.dds.publication.DataWriter** (p. 553) is equivalent to unregistering all the instances it was writing, but is *not* the same as "disposing" all the instances.

For a **com.rti.dds.topic.Topic** (p. 1807) with **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_↩ OWNERSHIP_QOS** (p. 1348), if the current owner of an instance *disposes* it, the readers accessing the instance will see the instance_state as being "DISPOSED" and not see the values being written by the weaker writer (even after the stronger one has disposed the instance). This is because the **com.rti.dds.publication.DataWriter** (p. 553) that owns the instance is saying that the instance no longer exists (e.g. the master of the database is saying that a record has been deleted) and thus the readers should see it as such.

For a **com.rti.dds.topic.Topic** (p. 1807) with **com.rti.dds.infrastructure.OwnershipQosPolicyKind.EXCLUSIVE_↩ OWNERSHIP_QOS** (p. 1348), if the current owner of an instance *unregisters* it, then it will relinquish ownership of the instance and thus the readers may see the value updated by another writer (which will then become the owner). This is because the owner said that it no longer will be providing values for the instance and thus another writer can take ownership and provide those values.

8.201.5 Member Data Documentation

8.201.5.1 kind

```
OwnershipQosPolicyKind kind
```

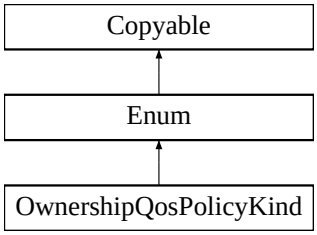
The kind of ownership.

[default] com.rti.dds.infrastructure.OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS (p. 1347)

8.202 OwnershipQosPolicyKind Class Reference

Kinds of ownership.

Inheritance diagram for OwnershipQosPolicyKind:



Static Public Attributes

- static final **OwnershipQosPolicyKind SHARED_OWNERSHIP_QOS**

  [default] *Indicates shared ownership for each instance.*
- static final **OwnershipQosPolicyKind EXCLUSIVE_OWNERSHIP_QOS**

  Indicates each instance can only be owned by one ***com.rti.dds.publication.DataWriter*** *(p. 553), but the owner of an instance can change dynamically.*

Additional Inherited Members

8.202.1 Detailed Description

Kinds of ownership.

QoS:

com.rti.dds.infrastructure.OwnershipQosPolicy (p. 1342)

8.202.2 Member Data Documentation

8.202.2.1 SHARED_OWNERSHIP_QOS

```
final OwnershipQosPolicyKind SHARED_OWNERSHIP_QOS [static]
```

[default] Indicates shared ownership for each instance.

Multiple writers are allowed to update the same instance and all the updates are made available to the readers. In other words there is no concept of an owner for the instances.

This is the **default** behavior if the **OWNERSHIP** (p. 244) policy is not specified or supported.

8.202.2.2 EXCLUSIVE_OWNERSHIP_QOS

```
final OwnershipQosPolicyKind EXCLUSIVE_OWNERSHIP_QOS [static]
```

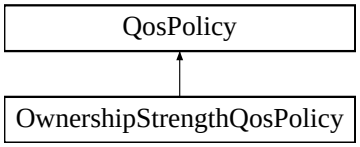
Indicates each instance can only be owned by one **com.rti.dds.publication.DataWriter** (p. 553), but the owner of an instance can change dynamically.

The selection of the owner is controlled by the setting of the **OWNERSHIP_STRENGTH** (p. 245) policy. The owner is always set to be the highest-strength **com.rti.dds.publication.DataWriter** (p. 553) object among the ones currently active (as determined by the **LIVELINESS** (p. 239)).

8.203 OwnershipStrengthQosPolicy Class Reference

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).

Inheritance diagram for OwnershipStrengthQosPolicy:



Public Attributes

- int **value**

  The strength value used to arbitrate among multiple writers.

8.203.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 553) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1807) + key).

This policy only applies if the **OWNERSHIP** (p. 244) policy is of kind **com.rti.dds.infrastructure.OwnershipQos↩ PolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1348).

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **YES** (p. 256)

The value of the **OWNERSHIP_STRENGTH** (p. 245) is used to determine the ownership of a data instance (identified by the key). The arbitration is performed by the **com.rti.dds.subscription.DataReader** (p. 450).

See also

EXCLUSIVE ownership (p. 1343)

8.203.2 Member Data Documentation

8.203.2.1 value

```
int value
```

The strength value used to arbitrate among multiple writers.

[default] 0

[range] [0, 1 million]

8.204 ParticipantBuiltinTopicData Class Reference

Entry created when a **DomainParticipant** (p. 670) object is discovered.

Inherits AbstractBuiltinTopicData.

Public Attributes

- final **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- final **UserDataQosPolicy user_data**

  *Policy of the corresponding **DomainParticipant** (p. 670).*
- final **PropertyQosPolicy property**

  *<<**extension**>> (p. 155) Name value pair properties to be stored with **DomainParticipant** (p. 670)*
- final **ProtocolVersion_t rtps_protocol_version**

  *<<**extension**>> (p. 155) Version number of the RTPS wire protocol used.*
- final **VendorId_t rtps_vendor_id**

  *<<**extension**>> (p. 155) ID of vendor implementing the RTPS wire protocol.*
- int **dds_builtin_endpoints**

  *<<**extension**>> (p. 155) Bitmap of builtin endpoints supported by the participant.*
- final **LocatorSeq default_unicast_locators**

  *<<**extension**>> (p. 155) Unicast locators used when individual entities do not specify unicast locators.*
- final **ProductVersion_t product_version**

  *<<**extension**>> (p. 155) This is a vendor specific parameter. It gives the current version for rti-dds.*
- final **EntityNameQosPolicy participant_name**

  *<<**extension**>> (p. 155) The participant name and role name.*
- final **PartitionQosPolicy partition**

  *<<**extension**>> (p. 155) PartitionQosPolicy of the participant.*
- final **ParticipantTrustProtectionInfo trust_protection_info**

  *<<**extension**>> (p. 155) Trust Plugins protection information associated with the discovered **DomainParticipant** (p. 670).*
- final **ParticipantTrustAlgorithmInfo trust_algorithm_info**

  *<<**extension**>> (p. 155) Trust Plugins algorithms associated with the discovered **DomainParticipant** (p. 670).*
- int **domain_id**

  *<<**extension**>> (p. 155) Domain ID associated with the discovered participant.*
- **TransportInfoSeq transport_info**

  *<<**extension**>> (p. 155) A sequence of **com.rti.dds.infrastructure.TransportInfo_t** (p. 1845) containing information about each of the installed transports of the discovered participant.*
- boolean **partial_configuration**

  *<<**extension**>> (p. 155) Indicates whether a **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) only contains bootstrapping information.*

8.204.1 Detailed Description

Entry created when a **DomainParticipant** (p. 670) object is discovered.

Data associated with the built-in topic **com.rti.dds.domain.builtin.ParticipantBuiltinTopicDataTypeSupport.↩ PARTICIPANT_TOPIC_NAME** (p. 162). It contains QoS policies and additional information that apply to the remote **com.rti.dds.domain.DomainParticipant** (p. 670).

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 162)

builtin.ParticipantBuiltinTopicDataDataReader (p. 1354)

8.204.2 Member Data Documentation

8.204.2.1 key

```
final  BuiltinTopicKey_t key
```

DCPS key to distinguish entries.

8.204.2.2 user_data

```
final  UserDataQosPolicy user_data
```

Policy of the corresponding **DomainParticipant** (p. 670).

8.204.2.3 property

```
final  PropertyQosPolicy property
```

<<***extension***>> *(p. 155)* Name value pair properties to be stored with **DomainParticipant** (p. 670)

8.204.2.4 rtps_protocol_version

```
final  ProtocolVersion_t rtps_protocol_version
```

<<***extension***>> *(p. 155)* Version number of the RTPS wire protocol used.

8.204.2.5 rtps_vendor_id

```
final  VendorId_t rtps_vendor_id
```

<<***extension***>> *(p. 155)* ID of vendor implementing the RTPS wire protocol.

8.204.2.6 dds_builtin_endpoints

```
int dds_builtin_endpoints
```

<<***extension***>> *(p. 155)* Bitmap of builtin endpoints supported by the participant.

Each bit indicates a builtin endpoint that may be available on the participant for use in discovery.

8.204.2.7 default_unicast_locators

```
final LocatorSeq default_unicast_locators
```

<<***extension***>> *(p. 155)* Unicast locators used when individual entities do not specify unicast locators.

8.204.2.8 product_version

```
final ProductVersion_t product_version
```

<<***extension***>> *(p. 155)* This is a vendor specific parameter. It gives the current version for rti-dds.

8.204.2.9 participant_name

```
final EntityNameQosPolicy participant_name
```

<<***extension***>> *(p. 155)* The participant name and role name.

This parameter contains the name and the role name of the discovered participant.

8.204.2.10 partition

```
final PartitionQosPolicy partition
```

<<***extension***>> *(p. 155)* PartitionQosPolicy of the participant.

8.204.2.11 trust_protection_info

```
final ParticipantTrustProtectionInfo trust_protection_info
```

<<***extension***>> *(*p. *155)* Trust Plugins protection information associated with the discovered **DomainParticipant** (p. 670).

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged **DomainParticipant** (p. 670) data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two DomainParticipants will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the **DomainParticipant** (p. 670) is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.204.2.12 trust_algorithm_info

```
final ParticipantTrustAlgorithmInfo trust_algorithm_info
```

<<***extension***>> *(*p. *155)* Trust Plugins algorithms associated with the discovered **DomainParticipant** (p. 670).

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged **DomainParticipant** (p. 670) data and metadata. trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two DomainParticipants will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the **DomainParticipant** (p. 670) is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.204.2.13 domain_id

```
int domain_id
```

<<***extension***>> *(*p. *155)* Domain ID associated with the discovered participant.

8.204.2.14 transport_info

```
TransportInfoSeq transport_info
```

<<***extension***>> *(*p. *155)* A sequence of **com.rti.dds.infrastructure.TransportInfo_t** (p. 1845) containing information about each of the installed transports of the discovered participant.

This parameter contains a sequence of **com.rti.dds.infrastructure.TransportInfo_t** (p. 1845) containing the class_id and message_size_max for all installed transports of the discovered participant. The maximum number of **com.↩ rti.dds.infrastructure.TransportInfo_t** (p. 1845) that will be stored in this sequence is controlled by the Domain Participant's resource limit **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.transport_info_↩ list_max_length** (p. 821).

8.204.2.15 partial_configuration

```
boolean partial_configuration
```

<<***extension***>> *(p. 155)* Indicates whether a **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) only contains bootstrapping information.

If this is com.rti.dds.infrastructure.true, the **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) only contains bootstrapping information. If it is com.rti.dds.infrastructure.false, it contains both bootstrapping and configuration information. The following fields are valid when this is set to com.rti.dds.infrastructure.true:

- **builtin.ParticipantBuiltinTopicData.key** (p. 1351)
- **builtin.ParticipantBuiltinTopicData.property** (p. 1351) (only dds.domain_participant.domain_tag is valid if set)
- **builtin.ParticipantBuiltinTopicData.rtps_protocol_version** (p. 1351)
- **builtin.ParticipantBuiltinTopicData.rtps_vendor_id** (p. 1351)
- **builtin.ParticipantBuiltinTopicData.product_version** (p. 1352)
- **builtin.ParticipantBuiltinTopicData.domain_id** (p. 1353)
- **builtin.ParticipantBuiltinTopicData.transport_info** (p. 1353)
- **builtin.ParticipantBuiltinTopicData.partition** (p. 1352)
- **builtin.ParticipantBuiltinTopicData.trust_protection_info** (p. 1352)
- **builtin.ParticipantBuiltinTopicData.trust_algorithm_info** (p. 1353)

All other fields are invalid.

This field will only be set to com.rti.dds.infrastructure.true if a participant using **com.rti.dds.infrastructure.Discovery↩ ConfigBuiltinPluginKind.SPDP2** (p. 645) receives a bootstrap message and **com.rti.dds.infrastructure.Discovery↩ ConfigQosPolicy.ignore_default_domain_announcements** (p. 658) is set to com.rti.dds.infrastructure.false (non-default).

If a participant is using **com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind.SPDP** (p. 644), this field will always be set to com.rti.dds.infrastructure.false.

8.205 ParticipantBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) > .

Inherits DataReaderImpl.

8.205.1 Detailed Description

Instantiates `DataReader` < **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) > .

com.rti.dds.subscription.DataReader (p. 450) of topic **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData↩ TypeSupport.PARTICIPANT_TOPIC_NAME** (p. 162) used for accessing **com.rti.dds.domain.builtin.Participant↩ BuiltinTopicData** (p. 1349) of the remote **com.rti.dds.domain.DomainParticipant** (p. 670).

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooDataReader** (p. 1067)

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

com.rti.dds.domain.builtin.ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 162)

8.206 ParticipantBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.domain.↩ builtin.ParticipantBuiltinTopicData** (p. 1349) > .

Inherits AbstractBuiltinTopicDataSeq.

8.206.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.domain.↩ builtin.ParticipantBuiltinTopicData** (p. 1349) > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

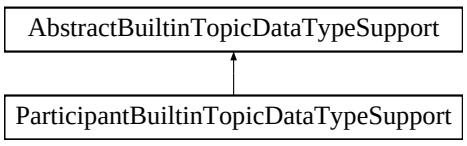
See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

8.207 ParticipantBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) > .

Inheritance diagram for ParticipantBuiltinTopicDataTypeSupport:



Static Public Attributes

- static final String **PARTICIPANT_TOPIC_NAME** = DDS_PARTICIPANT_TOPIC_NAME()

  Participant topic name.

Additional Inherited Members

8.207.1 Detailed Description

Instantiates `TypeSupport` < **com.rti.dds.domain.builtin.ParticipantBuiltinTopicData** (p. 1349) > .

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooTypeSupport** (p. 1118)

See also

com.rti.dds.domain.builtin.ParticipantBuiltinTopicData (p. 1349)

8.208 ParticipantTrustAlgorithmInfo Class Reference

Trust Plugins algorithm information associated with the discovered DomainParticipant.

Inherits Struct.

Public Member Functions

- **ParticipantTrustAlgorithmInfo** ()

  Create an instance with the default Trust Algorithm Info associated with the discovered DomainParticipant.
- **ParticipantTrustAlgorithmInfo** (**ParticipantTrustSignatureAlgorithmInfo signature**, **ParticipantTrust↩ KeyEstablishmentAlgorithmInfo key_establishment**, **ParticipantTrustInterceptorAlgorithmInfo interceptor**)

  Create an instance with the Trust Algorithm Info passed as input.

Public Attributes

- **ParticipantTrustSignatureAlgorithmInfo signature**

  Information regarding algorithms for validation of data and metadata exchanged by the DomainParticipant.
- **ParticipantTrustKeyEstablishmentAlgorithmInfo key_establishment**

  Information regarding algorithms for transformation of data and metadata exchanged between two DomainParticipants.
- **ParticipantTrustInterceptorAlgorithmInfo interceptor**

  Information regarding algorithms for interception of data and metadata exchanged by the DomainParticipant.

8.208.1 Detailed Description

Trust Plugins algorithm information associated with the discovered DomainParticipant.

8.208.2 Constructor & Destructor Documentation

8.208.2.1 ParticipantTrustAlgorithmInfo() [1/2]

```
ParticipantTrustAlgorithmInfo ( )
```

Create an instance with the default Trust Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.208.2.2 ParticipantTrustAlgorithmInfo() [2/2]

```
ParticipantTrustAlgorithmInfo (
            ParticipantTrustSignatureAlgorithmInfo signature,
            ParticipantTrustKeyEstablishmentAlgorithmInfo key_establishment,
            ParticipantTrustInterceptorAlgorithmInfo interceptor )
```

Create an instance with the Trust Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

References **ParticipantTrustAlgorithmInfo.interceptor**, **ParticipantTrustAlgorithmInfo.key_establishment**, and **ParticipantTrustAlgorithmInfo.signature**.

8.208.3 Member Data Documentation

8.208.3.1 signature

```
ParticipantTrustSignatureAlgorithmInfo signature
```

Information regarding algorithms for validation of data and metadata exchanged by the DomainParticipant.

Referenced by **ParticipantTrustAlgorithmInfo.ParticipantTrustAlgorithmInfo()**.

8.208.3.2 key_establishment

ParticipantTrustKeyEstablishmentAlgorithmInfo key_establishment

Information regarding algorithms for transformation of data and metadata exchanged between two DomainParticipants.

Referenced by **ParticipantTrustAlgorithmInfo.ParticipantTrustAlgorithmInfo()**.

8.208.3.3 interceptor

ParticipantTrustInterceptorAlgorithmInfo interceptor

Information regarding algorithms for interception of data and metadata exchanged by the DomainParticipant.

Referenced by **ParticipantTrustAlgorithmInfo.ParticipantTrustAlgorithmInfo()**.

8.209 ParticipantTrustInterceptorAlgorithmInfo Class Reference

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

Inherits Struct.

Public Member Functions

- **ParticipantTrustInterceptorAlgorithmInfo** ()

  Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered DomainParticipant.
- **ParticipantTrustInterceptorAlgorithmInfo** (int **supported_mask**, int **builtin_endpoints_required_mask**, int **builtin_kx_endpoints_required_mask**, int **user_endpoints_default_required_mask**)

  Create an instance with the Trust Interceptor Algorithm Info passed as input.

Public Attributes

- int **supported_mask**

  Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.
- int **builtin_endpoints_required_mask**

  Trust Plugins algorithms used for interception of metadata exchanged by the discovery, service request, and liveliness builtin endpoints.
- int **builtin_kx_endpoints_required_mask**

  Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.
- int **user_endpoints_default_required_mask**

  Default trust plugin algorithms used for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when builtin_trust.EndpointTrustInterceptorAlgorithmInfo is not serialized.

8.209.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

8.209.2 Constructor & Destructor Documentation

8.209.2.1 ParticipantTrustInterceptorAlgorithmInfo() [1/2]

```
ParticipantTrustInterceptorAlgorithmInfo ( )
```

Create an instance with the default Trust Interceptor Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.209.2.2 ParticipantTrustInterceptorAlgorithmInfo() [2/2]

```
ParticipantTrustInterceptorAlgorithmInfo (
            int supported_mask,
            int builtin_endpoints_required_mask,
            int builtin_kx_endpoints_required_mask,
            int user_endpoints_default_required_mask )
```

Create an instance with the Trust Interceptor Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

References **ParticipantTrustInterceptorAlgorithmInfo.builtin_endpoints_required_mask**, **ParticipantTrust↩ InterceptorAlgorithmInfo.builtin_kx_endpoints_required_mask**, **ParticipantTrustInterceptorAlgorithmInfo.↩ supported_mask**, and **ParticipantTrustInterceptorAlgorithmInfo.user_endpoints_default_required_mask**.

8.209.3 Member Data Documentation

8.209.3.1 supported_mask

```
int supported_mask
```

Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.

Referenced by **ParticipantTrustInterceptorAlgorithmInfo.ParticipantTrustInterceptorAlgorithmInfo()**.

8.209.3.2 builtin_endpoints_required_mask

```
int builtin_endpoints_required_mask
```

Trust Plugins algorithms used for interception of metadata exchanged by the discovery, service request, and liveliness builtin endpoints.

Referenced by **ParticipantTrustInterceptorAlgorithmInfo.ParticipantTrustInterceptorAlgorithmInfo()**.

8.209.3.3 builtin_kx_endpoints_required_mask

```
int builtin_kx_endpoints_required_mask
```

Trust Plugins algorithms used for interception of metadata exchanged by the key exchange builtin endpoints.

Referenced by **ParticipantTrustInterceptorAlgorithmInfo.ParticipantTrustInterceptorAlgorithmInfo()**.

8.209.3.4 user_endpoints_default_required_mask

```
int user_endpoints_default_required_mask
```

Default trust plugin algorithms used for interception of data exchanged by the user endpoints. This is the algorithm that remote DomainParticipants should assume when builtin_trust.EndpointTrustInterceptorAlgorithmInfo is not serialized.

Referenced by **ParticipantTrustInterceptorAlgorithmInfo.ParticipantTrustInterceptorAlgorithmInfo()**.

8.210 ParticipantTrustKeyEstablishmentAlgorithmInfo Class Reference

Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

Inherits Struct.

Public Member Functions

- **ParticipantTrustKeyEstablishmentAlgorithmInfo** ()

  Create an instance with the default Trust Key Establishment Algorithm Info associated with the discovered Domain↩ Participant.
- **ParticipantTrustKeyEstablishmentAlgorithmInfo** (**TrustAlgorithmRequirements shared_secret**)

  Create an instance with the Trust Key Establishment Algorithm Info passed as input.

Public Attributes

- **TrustAlgorithmRequirements shared_secret**

  Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

8.210.1 Detailed Description

Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

8.210.2 Constructor & Destructor Documentation

8.210.2.1 ParticipantTrustKeyEstablishmentAlgorithmInfo() [1/2]

```
ParticipantTrustKeyEstablishmentAlgorithmInfo ( )
```

Create an instance with the default Trust Key Establishment Algorithm Info associated with the discovered Domain↩ Participant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.210.2.2 ParticipantTrustKeyEstablishmentAlgorithmInfo() [2/2]

```
ParticipantTrustKeyEstablishmentAlgorithmInfo (
            TrustAlgorithmRequirements shared_secret )
```

Create an instance with the Trust Key Establishment Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

References **ParticipantTrustKeyEstablishmentAlgorithmInfo.shared_secret**.

8.210.3 Member Data Documentation

8.210.3.1 shared_secret

```
TrustAlgorithmRequirements shared_secret
```

Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

Referenced by **ParticipantTrustKeyEstablishmentAlgorithmInfo.ParticipantTrustKeyEstablishmentAlgorithm↩ Info()**.

8.211 ParticipantTrustProtectionInfo Class Reference

Trust Plugins Protection information associated with the discovered DomainParticipant.

Inherits Struct.

Public Member Functions

- **ParticipantTrustProtectionInfo** ()

  Create an instance with the default Trust Plugins Protection Info associated with the DomainParticipant.
- **ParticipantTrustProtectionInfo** (int **bitmask**, int **plugin_bitmask**)

  Create an instance with the Trust Plugins Protection Info passed as input.

Public Attributes

- int **bitmask**

  Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.
- int **plugin_bitmask**

  Internal plugin information that is opaque to DDS.

8.211.1 Detailed Description

Trust Plugins Protection information associated with the discovered DomainParticipant.

8.211.2 Constructor & Destructor Documentation

8.211.2.1 ParticipantTrustProtectionInfo() [1/2]

```
ParticipantTrustProtectionInfo ( )
```

Create an instance with the default Trust Plugins Protection Info associated with the DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.211.2.2 ParticipantTrustProtectionInfo() [2/2]

```
ParticipantTrustProtectionInfo (
            int bitmask,
            int plugin_bitmask )
```

Create an instance with the Trust Plugins Protection Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

References **ParticipantTrustProtectionInfo.bitmask**, and **ParticipantTrustProtectionInfo.plugin_bitmask**.

8.211.3 Member Data Documentation

8.211.3.1 bitmask

```
int bitmask
```

Information about how RTPS wire serialization, discovery, and liveliness interact with the associated Trust Plugins Protection configuration.

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

Referenced by **ParticipantTrustProtectionInfo.ParticipantTrustProtectionInfo()**.

8.211.3.2 plugin_bitmask

```
int plugin_bitmask
```

Internal plugin information that is opaque to DDS.

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

Referenced by **ParticipantTrustProtectionInfo.ParticipantTrustProtectionInfo()**.

8.212 ParticipantTrustSignatureAlgorithmInfo Class Reference

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

Inherits Struct.

Public Member Functions

- **ParticipantTrustSignatureAlgorithmInfo** ()

  Create an instance with the default Trust Signature Algorithm Info associated with the discovered DomainParticipant.
- **ParticipantTrustSignatureAlgorithmInfo** (**TrustAlgorithmRequirements trust_chain**, **TrustAlgorithm↩ Requirements message_auth**)

  Create an instance with the Trust Signature Algorithm Info passed as input.

Public Attributes

- **TrustAlgorithmRequirements trust_chain**

  Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.
- **TrustAlgorithmRequirements message_auth**

  Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

8.212.1 Detailed Description

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

8.212.2 Constructor & Destructor Documentation

8.212.2.1 ParticipantTrustSignatureAlgorithmInfo() [1/2]

```
ParticipantTrustSignatureAlgorithmInfo ( )
```

Create an instance with the default Trust Signature Algorithm Info associated with the discovered DomainParticipant.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

8.212.2.2 ParticipantTrustSignatureAlgorithmInfo() [2/2]

```
ParticipantTrustSignatureAlgorithmInfo (
            TrustAlgorithmRequirements trust_chain,
            TrustAlgorithmRequirements message_auth )
```

Create an instance with the Trust Signature Algorithm Info passed as input.

The meaning of this field may vary depending on what Trust Plugins the DomainParticipant is using.

References **ParticipantTrustSignatureAlgorithmInfo.message_auth**, and **ParticipantTrustSignatureAlgorithm↩ Info.trust_chain**.

8.212.3 Member Data Documentation

8.212.3.1 trust_chain

TrustAlgorithmRequirements trust_chain

Trust Plugins signature algorithm requirements of the DomainParticipant in the context of creating a chain of trust.

Referenced by **ParticipantTrustSignatureAlgorithmInfo.ParticipantTrustSignatureAlgorithmInfo()**.

8.212.3.2 message_auth

TrustAlgorithmRequirements message_auth

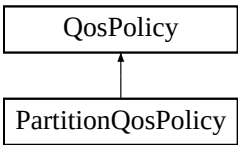
Trust Plugins signature algorithm requirements of the DomainParticipant in the context of checking that messages are authentic.

Referenced by **ParticipantTrustSignatureAlgorithmInfo.ParticipantTrustSignatureAlgorithmInfo()**.

8.213 PartitionQosPolicy Class Reference

Set of strings that introduces logical partitions in **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.↩ publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entities.

Inheritance diagram for PartitionQosPolicy:



Public Attributes

- final **StringSeq name**

  A list of partition names.

8.213.1 Detailed Description

Set of strings that introduces logical partitions in **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.↩ publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entities.

This QoS policy is used to set string identifiers that are used for partitioning entities that would otherwise be connected to and exchange data with each other:

- A **com.rti.dds.publication.DataWriter** (p. 553) within a **com.rti.dds.publication.Publisher** (p. 1466) only communicates with a **com.rti.dds.subscription.DataReader** (p. 450) in a **com.rti.dds.subscription.Subscriber** (p. 1730) if (in addition to matching the **com.rti.dds.topic.Topic** (p. 1807) and having compatible QoS) the **com.↩ rti.dds.publication.Publisher** (p. 1466) and **com.rti.dds.subscription.Subscriber** (p. 1730) have a common partition name string.
- **com.rti.dds.domain.DomainParticipant** (p. 670) entities (with the same domain ID and domain tag) are visible to each other only if they have at least one partition name string in common.

Entity:

com.rti.dds.publication.Publisher (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730), **com.rti.dds.↩ domain.DomainParticipant** (p. 670)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **YES** (p. 256)

8.213.2 Usage

The Partition QoS policy provides another way to control which entities will match-and thus communicate with-which other entities. It can be used to prevent entities that would have otherwise matched from talking to each other. Much in the same way that only applications within the same DDS domain will communicate with each other, only entities that belong to the same partition can talk to each other.

The Partition QoS policy allows you to add one or more strings, "partitions", to an entity:

- A DataWriter and DataReader for the same topic are only considered matched if their Publishers and Subscribers have partitions in common (intersecting partitions).
- DomainParticipants (with the same domain ID and domain tag) are visible to each other only if they have at least one partition in common.

Since the set of partitions for an entity can be dynamically changed, the Partition QoS policy is useful for creating temporary separation groups among entities that would otherwise be connected to and exchange data with each other.

DomainParticipant partitions and Publisher/Subscriber partitions are independent of each other. You can use both features independently or in combination to provide the right level of isolation.

Failure to match partitions is not considered an incompatible QoS and does not trigger any listeners or conditions. A change in this policy *can* potentially modify the "match" of existing DataReader and DataWriter entities. It may establish new "matches" that did not exist before, or break existing matches.

Partition strings are usually directly matched via string comparisons. However, partition strings can also contain wildcard symbols so that partitions can be matched via pattern matching. As long as the partitions or wildcard patterns of an entity intersect with the partitions or wildcard patterns of otherwise matching entities, the entities match; otherwise they do not.

These partition name patterns are regular expressions as defined by the POSIX fnmatch API (1003.2-1992 section B.6). A **com.rti.dds.domain.DomainParticipant** (p. 670), **com.rti.dds.publication.Publisher** (p. 1466), or **com.rti.dds.subscription.Subscriber** (p. 1730) entity may include regular expressions in partition names, but no two names that both contain wildcards will ever be considered to match. This means that although regular expressions may be used on the entities, RTI Connext will not try to match two regular expressions.

Each entity must belong to at least one logical partition. A regular expression is not considered to be a logical partition. If an entity has not specified a logical partition, it is assumed to be in the default partition. The default partition is defined to be an empty string (""). Put another way:

- An empty sequence of strings in this QoS policy is considered equivalent to a sequence containing only a single string, the empty string.
- A string sequence that contains only regular expressions and no literal strings, it is treated as if it had an additional element, the empty string.

Partitions are different from creating **com.rti.dds.infrastructure.Entity** (p. 1029) objects in different domains in several ways.

- First, entities belonging to different domains are completely isolated from each other; there is no traffic, meta-traffic or any other way for an application or RTI Connext itself to see entities in a domain it does not belong to.
- Second, a **com.rti.dds.infrastructure.Entity** (p. 1029) can only belong to one domain whereas a **com.rti.dds.infrastructure.Entity** (p. 1029) can be in multiple partitions.
- Finally, as far as RTI Connext is concerned, each unique data instance is identified by the tuple **(DomainID, domain tag, com.rti.dds.topic.Topic (**p. **1807), key)**. Therefore two **com.rti.dds.infrastructure.Entity** (p. 1029) objects in different domains cannot refer to the same data instance. On the other hand, the same data instance can be made available (published) or requested (subscribed) on one or more partitions.

For more information, see the "PARTITION QosPolicy" section of the `Core Libraries User's Manual`.

8.213.3 Member Data Documentation

8.213.3.1 name

```
final  StringSeq name
```

A list of partition names.

Several restrictions apply to the partition names in this sequence. A violation of one of the following rules will result in a **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) when setting a **com.rti.dds.↩ publication.Publisher** (p. 1466)'s or **com.rti.dds.subscription.Subscriber** (p. 1730)'s QoS.

- A partition name string cannot be NULL, nor can it contain the reserved comma character (',').
- The maximum number of partition name strings allowable in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365) is specified on a domain basis in **com.rti.dds.infrastructure.DomainParticipantResourceLimits↩ QosPolicy.max_partitions** (p. 816). The length of this sequence may not be greater than that value.
- The maximum cumulative length of all partition name strings in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1365) is specified on a domain basis in **com.rti.dds.infrastructure.DomainParticipantResourceLimits↩ QosPolicy.max_partition_cumulative_characters** (p. 816).

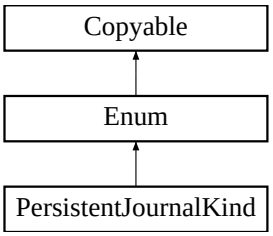
[default] Empty sequence (zero-length sequence). Since no logical partition is specified, RTI Connext will assume the entity to be in default partition (empty string partition "").

[range] List of partition name with above restrictions

8.214 PersistentJournalKind Class Reference

Sets the journal mode of the persistent storage.

Inheritance diagram for PersistentJournalKind:



Static Public Attributes

- static final **PersistentJournalKind TRUNCATE_PERSISTENT_JOURNAL**

  Commits transactions by truncating the rollback journal to zero-length instead of deleting it.
- static final **PersistentJournalKind PERSIST_PERSISTENT_JOURNAL**

  Prevents the rollback journal from being deleted at the end of each transaction. Instead, the header of the journal is overwritten with zeros.
- static final **PersistentJournalKind MEMORY_PERSISTENT_JOURNAL**

  Stores the rollback journal in volatile RAM. This saves disk I/O.
- static final **PersistentJournalKind WAL_PERSISTENT_JOURNAL**

  Uses a write-ahead log instead of a rollback journal to implement transactions.
- static final **PersistentJournalKind OFF_PERSISTENT_JOURNAL**

  Completely disables the rollback journal. If the application crashes in the middle of a transaction when the OFF journaling mode is set, the persistent storage will very likely be corrupted.

Additional Inherited Members

8.214.1 Detailed Description

Sets the journal mode of the persistent storage.

The rollback journal is used in SQLite to store the state of the persistent storage before a transaction is committed.

QoS:

com.rti.dds.infrastructure.DurabilityQosPolicy (p. 830)

8.214.2 Member Data Documentation

8.214.2.1 TRUNCATE_PERSISTENT_JOURNAL

```
final PersistentJournalKind TRUNCATE_PERSISTENT_JOURNAL [static]
```

Initial value:

```
=
            new PersistentJournalKind("TRUNCATE_PERSISTENT_JOURNAL", 1)
```

Commits transactions by truncating the rollback journal to zero-length instead of deleting it.

8.214.2.2 PERSIST_PERSISTENT_JOURNAL

```
final PersistentJournalKind PERSIST_PERSISTENT_JOURNAL [static]
```

Initial value:

```
=
            new PersistentJournalKind("PERSIST_PERSISTENT_JOURNAL", 2)
```

Prevents the rollback journal from being deleted at the end of each transaction. Instead, the header of the journal is overwritten with zeros.

8.214.2.3 MEMORY_PERSISTENT_JOURNAL

```
final PersistentJournalKind MEMORY_PERSISTENT_JOURNAL [static]
```

Initial value:

```
=
            new PersistentJournalKind("MEMORY_PERSISTENT_JOURNAL", 3)
```

Stores the rollback journal in volatile RAM. This saves disk I/O.

8.214.2.4 WAL_PERSISTENT_JOURNAL

```
final PersistentJournalKind WAL_PERSISTENT_JOURNAL [static]
```

Initial value:

```
=
            new PersistentJournalKind("WAL_PERSISTENT_JOURNAL", 4)
```

Uses a write-ahead log instead of a rollback journal to implement transactions.

8.214.2.5 OFF_PERSISTENT_JOURNAL

```
final PersistentJournalKind OFF_PERSISTENT_JOURNAL [static]
```

Initial value:

```
=
            new PersistentJournalKind("OFF_PERSISTENT_JOURNAL", 5)
```

Completely disables the rollback journal. If the application crashes in the middle of a transaction when the OFF journaling mode is set, the persistent storage will very likely be corrupted.

8.215 PersistentStorageSettings Class Reference

Configures durable writer history and durable reader state.

Inherits Struct.

Public Member Functions

- **PersistentStorageSettings** ()

  Constructor.
- **PersistentStorageSettings** (**PersistentStorageSettings** src)

  Copy constructor.

Public Attributes

- boolean **enable** = false

  *Enables durable writer history in a **com.rti.dds.publication.DataWriter** (p. 553) and durable reader state in a **com.rti.dds.subscription.DataReader** (p. 450).*
- String **file_name** = null

  The file name where the durable writer history or durable reader state will be stored.
- String **trace_file_name** = null

  The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.
- **PersistentJournalKind journal_kind**

  Sets the journal mode of the persistent storage.
- **PersistentSynchronizationKind synchronization_kind**

  Sets the journal mode of the persistent storage.
- boolean **vacuum** = true

  Sets the auto-vacuum status of the storage.
- boolean **restore** = true

  Indicates if the persisted writer history or reader state must be restored.
- **AllocationSettings_t writer_instance_cache_allocation**

  Configures the resource limits associated with the instance durable writer history cache.
- **AllocationSettings_t writer_sample_cache_allocation**

  Configures the resource limits associated with the sample durable writer history cache.
- boolean **writer_memory_state** = true

  Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.
- int **reader_checkpoint_frequency** = 1

  Controls how often the reader state is stored into the database.

8.215.1 Detailed Description

Configures durable writer history and durable reader state.

In a **com.rti.dds.publication.DataWriter** (p. 553), this structure configures durable writer history. This feature allows a DataWriter to persist its historical cache, so that it can survive shutdowns, crashes, and restarts. When an application restarts, each DataWriter that has been configured to have durable writer history automatically loads all of the data in this cache from disk and can carry on sending data as if it had never stopped executing.

In a **com.rti.dds.subscription.DataReader** (p. 450), this structure configures durable reader state. This feature allows a DataReader to persist its state and remember which data it has already received. When an application restarts, each DataReader that has been configured to have durable reader state automatically loads its state from disk and can carry on receiving data as if it had never stopped executing. Data that had already been received by the DataReader before the restart will be suppressed so that it is not even sent over the network.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

QoS:

com.rti.dds.infrastructure.DurabilityQosPolicy (p. 830)

8.215.2 Constructor & Destructor Documentation

8.215.2.1 PersistentStorageSettings() [1/2]

```
PersistentStorageSettings ( )
```

Constructor.

8.215.2.2 PersistentStorageSettings() [2/2]

```
PersistentStorageSettings (
            PersistentStorageSettings src )
```

Copy constructor.

Parameters

src	<<***in***>> *(p. 156)* Source object.

8.215.3 Member Data Documentation

8.215.3.1 enable

```
boolean enable = false
```

Enables durable writer history in a **com.rti.dds.publication.DataWriter** (p. 553) and durable reader state in a **com.↩ rti.dds.subscription.DataReader** (p. 450).

When this field is set to com.rti.dds.infrastructure.true, the persistent storage configuration using this structure will take precedence over the configuration using the deprecated **dds.data_writer.history.odbc_plugin.builtin**.∗ and **dds.↩ data_reader.state**.∗ properties.

[default] com.rti.dds.infrastructure.false

8.215.3.2 file_name

```
String file_name = null
```

The file name where the durable writer history or durable reader state will be stored.

Setting this field to a value other than null is mandatory when enabling durable writer history or durable reader state.

If the file does not exist, it will be created.

If the file exists and **com.rti.dds.infrastructure.PersistentStorageSettings.restore** (p. 1374) is set to com.rti.dds.↩ infrastructure.true, the durable writer history or durable reader state will be restored from the file. Otherwise, the file will be overwritten.

Important: When the file exists, the fields **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 502) and **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid** (p. 598) will be set by RTI Connext based on the file content. If you change these fields, the value will be ignored.

RTI Connext uses SQLite to store the durable writer history and durable reader state.

[default] null

8.215.3.3 trace_file_name

```
String trace_file_name = null
```

The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.

Setting this field to a value other than null will enable tracing of the SQL statements executed when loading and storing the durable writer history or durable reader state.

Important: Enabling tracing will have a negative impact on performance. Use this feature only for debugging purposes.

[default] null

8.215.3.4 journal_kind

```
PersistentJournalKind journal_kind
```

Initial value:

```
=
            PersistentJournalKind.WAL_PERSISTENT_JOURNAL
```

Sets the journal mode of the persistent storage.

[default] com.rti.dds.infrastructure.PersistentJournalKind.PersistentJournalKind.WAL_PERSISTENT_JOURNAL

8.215.3.5 synchronization_kind

```
PersistentSynchronizationKind synchronization_kind
```

Initial value:

```
=
            PersistentSynchronizationKind.NORMAL_PERSISTENT_SYNCHRONIZATION
```

Sets the journal mode of the persistent storage.

[default] com.rti.dds.infrastructure.PersistentJournalKind.PersistentJournalKind.WAL_PERSISTENT_JOURNAL

8.215.3.6 vacuum

```
boolean vacuum = true
```

Sets the auto-vacuum status of the storage.

When auto-vacuum is com.rti.dds.infrastructure.true, the storage files will be compacted automatically with every transaction.

When auto-vacuum is com.rti.dds.infrastructure.false, after data is deleted from the storage files, the files remain the same size.

[default] com.rti.dds.infrastructure.true

8.215.3.7 restore

```
boolean restore = true
```

Indicates if the persisted writer history or reader state must be restored.

For a **com.rti.dds.publication.DataWriter** (p. 553), this field indicates whether or not the persisted writer history must be restored once the DataWriter is restarted.

For a **com.rti.dds.subscription.DataReader** (p. 450), this field indicates whether or not the persisted reader state must be restored once the DataReader is restarted.

[default] com.rti.dds.infrastructure.true

8.215.3.8 writer_instance_cache_allocation

```
AllocationSettings_t writer_instance_cache_allocation
```

Initial value:

```
=
            new AllocationSettings_t(
                    ReceiverPoolQosPolicy.LENGTH_AUTO,
                    ReceiverPoolQosPolicy.LENGTH_AUTO,
                    -1)
```

Configures the resource limits associated with the instance durable writer history cache.

This field only applies to **com.rti.dds.publication.DataWriter** (p. 553) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses an instance cache.

Do not confuse this limit with the initial and maximum number of instances that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.initial_instances** (p. 1593).

If **com.rti.dds.infrastructure.PersistentStorageSettings.writer_memory_state** (p. 1376) is set to com.rti.dds.infrastructure.true, then the value of **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) is set to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), overwriting any value set by the user.

com.rti.dds.infrastructure.AllocationSettings_t.incremental_count (p. 338) is ignored.

[range] com.rti.dds.infrastructure.AllocationSettings_t.max_count (p. 338) in interval [1, INT_MAX], **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257), or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259). **com.rti.dds.infrastructure.AllocationSettings_t.initial_count** (p. 337) in interval [1, INT_MAX], or **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257).

com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO (p. 257) means that the value will be set to the equivalent value of **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

[default] com.rti.dds.infrastructure.AllocationSettings_t.max_count (p. 338) = **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257) (= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)) and **com.rti.dds.infrastructure.AllocationSettings_t.initial_count** (p. 337) = **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257) (= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.initial_instances** (p. 1593)).

8.215.3.9 writer_sample_cache_allocation

```
AllocationSettings_t writer_sample_cache_allocation
```

Initial value:

```
=
            new AllocationSettings_t(
                    32,
                    32,
                    ReceiverPoolQosPolicy.LENGTH_AUTO)
```

Configures the resource limits associated with the sample durable writer history cache.

This field only applies to **com.rti.dds.publication.DataWriter** (p. 553) entities.

To minimize the number of accesses to the persisted storage, RTI Connext uses a sample cache.

Do not confuse this limit with the initial and maximum number of samples that can be maintained by a DataWriter in persistent storage. These resource limits are configured using **com.rti.dds.infrastructure.ResourceLimitsQos↩ Policy.max_samples** (p. 1592) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.initial_samples** (p. 1593).

com.rti.dds.infrastructure.AllocationSettings_t.incremental_count (p. 338) is ignored.

[range] com.rti.dds.infrastructure.AllocationSettings_t.max_count (p. 338) in interval [1, INT_MAX], **com.rti.↩ dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257), or **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.LENGTH_UNLIMITED** (p. 259). **com.rti.dds.infrastructure.AllocationSettings_t.initial_count** (p. 337) in interval [1, INT_MAX], or **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257).

com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO (p. 257) means that the value will be set to the equivalent value of **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

[default] com.rti.dds.infrastructure.AllocationSettings_t.max_count (p. 338) = 32 and **com.rti.dds.infrastructure.↩ AllocationSettings_t.initial_count** (p. 337) = 32.

8.215.3.10 writer_memory_state

```
boolean writer_memory_state = true
```

Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.

This field only applies to **com.rti.dds.publication.DataWriter** (p. 553) entities.

If this field is set to com.rti.dds.infrastructure.true, then **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) in **com.rti.dds.infrastructure.PersistentStorageSettings.writer_instance_cache_allocation** (p. 1374) is set to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), overwriting any value set by the user.

In addition, the durable writer history will keep a fixed state overhead per sample in memory. This mode provides the best durable writer history performance. However, the restore operation will be slower, and the maximum number of samples that the durable writer history can manage is limited by the available physical memory.

If this field is set to com.rti.dds.infrastructure.false, all the state will be kept in the underlying database. In this mode, the maximum number of samples in the durable writer history is not limited by the physical memory available.

This field is always set to com.rti.dds.infrastructure.false when the DataWriter is configured with **com.rti.↩ dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) set to **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 1531), or **com.rti.dds.infrastructure.AvailabilityQosPolicy.enable_required↩ _subscriptions** (p. 345) is set to com.rti.dds.infrastructure.true.

[default] com.rti.dds.infrastructure.true

8.215.3.11 reader_checkpoint_frequency

```
int reader_checkpoint_frequency = 1
```

Controls how often the reader state is stored into the database.

This field only applies to **com.rti.dds.subscription.DataReader** (p. 450) entities.

A value of N means store the state once every N received and processed samples.

The circumstances under which a data sample is considered "processed by the application" depends on the DataReader configuration.

For additional information on when a sample is considered "processed by the application" see `Durable Reader State`, in the Core Libraries User's Manual.

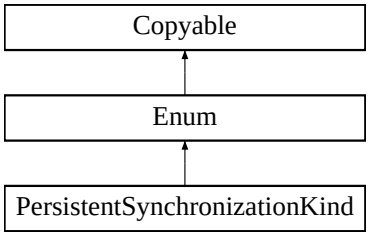
A high value will provide better performance. However, if the DataReader is restarted it may receive some duplicate samples.

[range] [1, 1000000] **[default]** 1

8.216 PersistentSynchronizationKind Class Reference

<<***extension***>> *(p. 155)* Whether instance state consistency is enabled.

Inheritance diagram for PersistentSynchronizationKind:



Static Public Attributes

- static final **PersistentSynchronizationKind NORMAL_PERSISTENT_SYNCHRONIZATION**

  Data (e.g., new sample) is written to disk at critical moments.
- static final **PersistentSynchronizationKind FULL_PERSISTENT_SYNCHRONIZATION**

  Data (e.g., new sample) is written to physical disk immediately.
- static final **PersistentSynchronizationKind OFF_PERSISTENT_SYNCHRONIZATION**

  No synchronization is enforced. Data will be written to physical disk when the operating system flushes its buffers.

Additional Inherited Members

8.216.1 Detailed Description

<<***extension***>> *(p. 155)* Whether instance state consistency is enabled.

QoS:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

8.216.2 Member Data Documentation

8.216.2.1 NORMAL_PERSISTENT_SYNCHRONIZATION

```
final PersistentSynchronizationKind NORMAL_PERSISTENT_SYNCHRONIZATION [static]
```

Initial value:

```
=
            new PersistentSynchronizationKind("NORMAL_PERSISTENT_SYNCHRONIZATION", 0)
```

Data (e.g., new sample) is written to disk at critical moments.

8.216.2.2 FULL_PERSISTENT_SYNCHRONIZATION

```
final PersistentSynchronizationKind FULL_PERSISTENT_SYNCHRONIZATION [static]
```

Initial value:

```
=
            new PersistentSynchronizationKind("FULL_PERSISTENT_SYNCHRONIZATION", 1)
```

Data (e.g., new sample) is written to physical disk immediately.

8.216.2.3 OFF_PERSISTENT_SYNCHRONIZATION

```
final PersistentSynchronizationKind OFF_PERSISTENT_SYNCHRONIZATION [static]
```

Initial value:

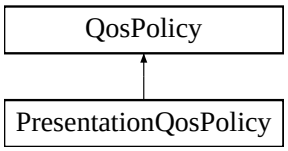
```
=
            new PersistentSynchronizationKind("OFF_PERSISTENT_SYNCHRONIZATION", 2)
```

No synchronization is enforced. Data will be written to physical disk when the operating system flushes its buffers.

8.217 PresentationQosPolicy Class Reference

Specifies how the samples representing changes to data instances are presented to a subscribing application.

Inheritance diagram for PresentationQosPolicy:



Public Attributes

- **PresentationQosPolicyAccessScopeKind access_scope**

  Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
- boolean **coherent_access**

  Specifies support for coherent access. Controls whether coherent access is supported within the scope `access_scope`*.*
- boolean **ordered_access**

  Specifies support for ordered access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`*.*
- boolean **drop_incomplete_coherent_set**

  *<<**extension**>> (*p. *155) Indicates whether or not a **com.rti.dds.subscription.DataReader** (*p. *450) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.*

8.217.1 Detailed Description

Specifies how the samples representing changes to data instances are presented to a subscribing application.

This QoS policy controls the extent to which changes to data instances can be made dependent on each other and also the kind of dependencies that can be propagated and maintained by RTI Connext. Specifically, this policy affects the application's ability to:

- specify and receive coherent changes to instances
- specify the relative order in which changes are presented

Entity:

com.rti.dds.publication.Publisher (p. 1466), **com.rti.dds.subscription.Subscriber** (p. 1730)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩
infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.217.2 Usage

A **com.rti.dds.subscription.DataReader** (p. 450) will usually receive data in the order that it was sent by a **com.rti.dds.publication.DataWriter** (p. 553), and the data is presented to the **com.rti.dds.subscription.DataReader** (p. 450) as soon as the application receives the next expected value. However, sometimes you may want a set of data for the same DataWriter or different DataWriters to be presented to the DataReader(s) only after *all* of the elements of the set have been received, but not before. Or you may want the data to be presented in a different order than that in which it was sent. Specifically for keyed data, you may want the middleware to present the data in keyed – or *instance* – order, such that samples pertaining to the same instance are presented together.

The Presentation QoS policy is a request-offered QoS policy that allows you to specify different *scopes* of presentation. It also controls whether or not a set of changes within the scope is delivered at the same time or can be delivered as soon as each element is received.

- `coherent_access` controls whether RTI Connext will preserve the groupings of changes made by a publishing application by means of the operations **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1483) and **com.rti.dds.publication.Publisher.end_coherent_changes** (p. 1483).

- `ordered_access` controls whether RTI Connext will preserve the order of changes.

- `access_scope` controls the granularity of the other settings. See below:

If `coherent_access` is set, then the `access_scope` set on the **com.rti.dds.subscription.Subscriber** (p. 1730) controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS (the default), the behavior is the same as com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS.

- If `access_scope` is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS, then coherent changes (indicated by their enclosure within calls to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1483) and **com.rti.dds.publication.Publisher.end_coherent_changes** (p. 1483)) will be made available as a unit to each remote **com.rti.dds.subscription.DataReader** (p. 450) independently. That is, changes made to instances within each individual **com.rti.dds.publication.DataWriter** (p. 553) will be presented as a unit. They will not be grouped with changes made to instances belonging to a different **com.rti.dds.publication.DataWriter** (p. 553).

- If access_scope is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS, then coherent changes made to instances through a set of **com.rti.dds.publication.DataWriter** (p. 553) entities attached to a common **com.rti.dds.publication.Publisher** (p. 1466) will be made available as a unit to the **com.rti.dds.subscription.DataReader** (p. 450) entities within a **com.rti.dds.subscription.Subscriber** (p. 1730).

If `ordered_access` is set, then the `access_scope` set on the **com.rti.dds.subscription.Subscriber** (p. 1730) controls the maximum extent to which order will be preserved by RTI Connext.

- If access_scope is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.Presentation↩ QosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS (the lowest level), then the relative order of DDS samples sent by a **com.rti.dds.publication.DataWriter** (p. 553) is only preserved on a per-instance basis. If two DDS samples refer to the same instance (identified by Topic and a particular value for the key), then the order in which they are stored in the **com.rti.dds.subscription.DataReader** (p. 450) queue is consistent with the order in which the changes occurred. However, if the two DDS samples belong to different instances, the order in which they are presented may or may not match the order in which the changes occurred. This is the case even if it is the same application thread making the changes using the same **com.rti.dds.publication.DataWriter** (p. 553).

- If access_scope is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.Presentation↩ QosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS, changes (creations, deletions, modifications) made by a single **com.rti.dds.publication.DataWriter** (p. 553) to one or more instances are presented to a **com.↩ rti.dds.subscription.DataReader** (p. 450) in the same order in which they occur. Changes made to instances though different **com.rti.dds.publication.DataWriter** (p. 553) entities are not required to be presented in the order in which they occur. This is the case even if the changes are made by a single application thread using **com.rti.dds.publication.DataWriter** (p. 553) objects attached to the same **com.rti.dds.publication.Publisher** (p. 1466).

- Finally, if access_scope is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.↩ PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS, changes made to instances via **com.rti.dds.publication.DataWriter** (p. 553) entities attached to the same **com.rti.dds.publication.Publisher** (p. 1466) object are presented to the **com.rti.dds.subscription.DataReader** (p. 450) entities within a **com.rti.↩ dds.subscription.Subscriber** (p. 1730) in the same order in which they occur. For this to happen, the application must take the samples using the Subscriber's **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) and **com.rti.dds.subscription.Subscriber.end_access** (p. 1750) operations (see the "Ordered Access" section in the "PRESENTATION **QosPolicy** (p. 1501)" section of the <A HREF="../../../manuals/connext_dds_↩ professional/users_manual/RTI_ConnextDDS_CoreLibraries_UsersManual.pdf" target="_blank">Core Libraries User's Manual).

If access_scope is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQos↩ PolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS on the **com.rti.dds.subscription.Subscriber** (p. 1730), then the **com.rti.dds.subscription.Subscriber** (p. 1730) will use the access_scope offered by the remote **com.rti.dds.publication.Publisher** (p. 1466).

Note that this QoS policy controls the scope at which related changes are made available to the subscriber. This means the subscriber **can** access the changes in a coherent manner and in the proper order; however, it does not necessarily imply that the **com.rti.dds.subscription.Subscriber** (p. 1730) **will** indeed access the changes in the correct order. For that to occur, the application at the subscriber end must use the proper logic in reading the **com.rti.dds.subscription.↩ DataReader** (p. 450) objects.

For com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.↩ GROUP_PRESENTATION_QOS the subscribing application must use the APIs **com.rti.dds.subscription.↩ Subscriber.begin_access** (p. 1749), **com.rti.dds.subscription.Subscriber.end_access** (p. 1750) and **com.rti.↩ dds.subscription.Subscriber.get_datareaders** (p. 1741) to access the changes in the proper order. If you do not use these operations, the data may not be ordered across DataWriters that belong to the same **com.rti.dds.publication.↩ Publisher** (p. 1466).

The field **com.rti.dds.subscription.SampleInfo.coherent_set_info** (p. 1647) is set when a sample is part of a coherent set. This field provides information to identify the coherent set that a sample is part of. In addition, com.rti.dds.↩ infrastructure.CoherentSetInfo.incomplete_coherent_set indicates if a sample is part of an incomplete coherent set (one for which not all samples have been received). Coherent sets for which some of the samples are filtered out by content or time on the **com.rti.dds.publication.DataWriter** (p. 553) are considered incomplete.

By default, the samples that are received from an incomplete coherent set are dropped by the DataReader(s) and they are not provided to the application. Such samples are reported as lost in the SAMPLE_LOST Status. By setting **com.rti.dds.infrastructure.PresentationQosPolicy.drop_incomplete_coherent_set** (p. 1383) to com.rti.dds.↩ infrastructure.false, you can change this behavior and, in this case, samples from incomplete coherent sets will be provided to the application. These samples have com.rti.dds.infrastructure.CoherentSetInfo.incomplete_coherent_set set to com.rti.dds.infrastructure.true.

8.217.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered* `access_scope` >= *requested* `access_scope` evaluates to 'TRUE' or requested `access_scope` is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQos↩ PolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS. For the purposes of this inequality, the values of `access_scope` are considered ordered such that com.rti.dds.infrastructure.PresentationQosPolicy↩ AccessScopeKind.PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS < com.rti.↩ dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC↩ _PRESENTATION_QOS < com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.Presentation↩ QosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS.

- requested `coherent_access` is com.rti.dds.infrastructure.false, or else both offered and requested `coherent_access` are com.rti.dds.infrastructure.true.

- requested `ordered_access` is com.rti.dds.infrastructure.false, or else both offered and requested `ordered_access` are com.rti.dds.infrastructure.true.

8.217.4 Member Data Documentation

8.217.4.1 access_scope

PresentationQosPolicyAccessScopeKind access_scope

Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScope↩ Kind.INSTANCE_PRESENTATION_QOS

8.217.4.2 coherent_access

```
boolean coherent_access
```

Specifies support for *coherent* access. Controls whether coherent access is supported within the scope `access_scope`.

That is, the ability to group a set of changes as a unit on the publishing end such that they are received as a unit at the subscribing end.

Note: To use this feature, the DataWriter must be configured for RELIABLE communication (see **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532)).

[default] com.rti.dds.infrastructure.false

8.217.4.3 ordered_access

```
boolean ordered_access
```

Specifies support for *ordered* access to the samples received at the subscription end. Controls whether ordered access is supported within the scope `access_scope`.

That is, the ability of the subscriber to see changes in the same order as they occurred on the publishing end.

[default] com.rti.dds.infrastructure.false

8.217.4.4 drop_incomplete_coherent_set

```
boolean drop_incomplete_coherent_set
```

<<***extension***>> *(p. 155)* Indicates whether or not a **com.rti.dds.subscription.DataReader** (p. 450) should drop samples from an incomplete coherent set (one for which not all the samples were received). Such samples are reported as lost in the SAMPLE_LOST Status.

Note that a coherent set will be considered incomplete if some of its samples are filtered by content or time on the DataWriter side.

By default, the samples that are received from an incomplete coherent set are dropped (and reported as lost) by the DataReader(s) and they are not provided to the application. By setting this parameter to com.rti.dds.infrastructure.false, you can change this behavior.

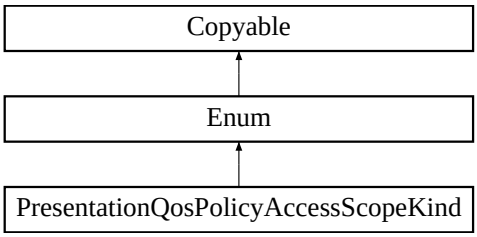
Samples from an incomplete coherent set have com.rti.dds.infrastructure.CoherentSetInfo.incomplete_coherent_set set to com.rti.dds.infrastructure.true in **com.rti.dds.subscription.SampleInfo.coherent_set_info** (p. 1647).

[default] com.rti.dds.infrastructure.true

8.218 PresentationQosPolicyAccessScopeKind Class Reference

Kinds of presentation "access scope".

Inheritance diagram for PresentationQosPolicyAccessScopeKind:



Static Public Attributes

- static final **PresentationQosPolicyAccessScopeKind INSTANCE_PRESENTATION_QOS**

  [default] *Scope spans only a single instance.*
- static final **PresentationQosPolicyAccessScopeKind TOPIC_PRESENTATION_QOS**

  *Scope spans all instances within the same **com.rti.dds.publication.DataWriter*** (p. *553), but not across instances in different **com.rti.dds.publication.DataWriter*** (p. *553) entities.*
- static final **PresentationQosPolicyAccessScopeKind GROUP_PRESENTATION_QOS**

  *Scope spans all instances belonging to **com.rti.dds.publication.DataWriter*** (p. *553) entities within the same **com.rti.↩ dds.publication.Publisher*** (p. *1466).*
- static final **PresentationQosPolicyAccessScopeKind HIGHEST_OFFERED_PRESENTATION_QOS**

  *This value only applies to a **com.rti.dds.subscription.Subscriber*** (p. *1730). The **com.rti.dds.subscription.Subscriber*** (p. *1730) will use the access scope specified by each remote **com.rti.dds.publication.Publisher*** (p. *1466).*

Additional Inherited Members

8.218.1 Detailed Description

Kinds of presentation "access scope".

Access scope determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

QoS:

com.rti.dds.infrastructure.PresentationQosPolicy (p. 1379)

8.218.2 Member Data Documentation

8.218.2.1 INSTANCE_PRESENTATION_QOS

final **PresentationQosPolicyAccessScopeKind** INSTANCE_PRESENTATION_QOS [static]

[default] Scope spans only a single instance.

Indicates that changes to one instance need not be coherent nor ordered with respect to changes to any other instance. In other words, order and coherent changes apply to each instance separately.

8.218.2.2 TOPIC_PRESENTATION_QOS

final **PresentationQosPolicyAccessScopeKind** TOPIC_PRESENTATION_QOS [static]

Scope spans all instances within the same **com.rti.dds.publication.DataWriter** (p. 553), but not across instances in different **com.rti.dds.publication.DataWriter** (p. 553) entities.

8.218.2.3 GROUP_PRESENTATION_QOS

final **PresentationQosPolicyAccessScopeKind** GROUP_PRESENTATION_QOS [static]

Scope spans all instances belonging to **com.rti.dds.publication.DataWriter** (p. 553) entities within the same **com.↩ rti.dds.publication.Publisher** (p. 1466).

8.218.2.4 HIGHEST_OFFERED_PRESENTATION_QOS

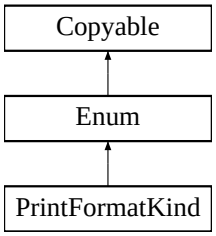
final **PresentationQosPolicyAccessScopeKind** HIGHEST_OFFERED_PRESENTATION_QOS [static]

This value only applies to a **com.rti.dds.subscription.Subscriber** (p. 1730). The **com.rti.dds.subscription.↩ Subscriber** (p. 1730) will use the access scope specified by each remote **com.rti.dds.publication.Publisher** (p. 1466).

8.219 PrintFormatKind Class Reference

Format kinds available when converting data samples to string representations.

Inheritance diagram for PrintFormatKind:



Static Public Attributes

- static final **PrintFormatKind DEFAULT_PRINT_FORMAT**

  Use a default format specific to RTI Connext to represent the data when converting to a string.
- static final **PrintFormatKind XML_PRINT_FORMAT**

  Use an XML format to represent the data when converting to a string.
- static final **PrintFormatKind JSON_PRINT_FORMAT**

  Use a JSON format to represent the data when converting to a string.

Additional Inherited Members

8.219.1 Detailed Description

Format kinds available when converting data samples to string representations.

8.219.2 Member Data Documentation

8.219.2.1 DEFAULT_PRINT_FORMAT

```
final PrintFormatKind DEFAULT_PRINT_FORMAT [static]
```

Use a default format specific to RTI Connext to represent the data when converting to a string.

8.219.2.2 XML_PRINT_FORMAT

```
final PrintFormatKind XML_PRINT_FORMAT [static]
```

Use an XML format to represent the data when converting to a string.

8.219.2.3 JSON_PRINT_FORMAT

```
final PrintFormatKind JSON_PRINT_FORMAT [static]
```

Use a JSON format to represent the data when converting to a string.

Referenced by **DynamicData.from_string()**.

8.220 PrintFormatProperty Class Reference

A collection of attributes used to configure how data samples will be formatted when converted to a string.

Public Attributes

- **PrintFormatKind kind** = **PrintFormatKind.DEFAULT_PRINT_FORMAT**

  The kind of format to be used when converting a data sample to a string.
- boolean **pretty_print** = true

  Choose whether to print a data sample in a more readable format or to eliminate all white space.
- boolean **enum_as_int** = false

  Choose whether to print enum values as integers or as strings.
- boolean **include_root_elements** = true

  Choose whether or not to include the root elements of the print format in the output string.

8.220.1 Detailed Description

A collection of attributes used to configure how data samples will be formatted when converted to a string.

8.220.2 Member Data Documentation

8.220.2.1 kind

```
PrintFormatKind kind =  PrintFormatKind.DEFAULT_PRINT_FORMAT
```

The kind of format to be used when converting a data sample to a string.

Data samples can be represented in a default, XML, or JSON format.

See also

com.rti.dds.topic.PrintFormatKind (p. 1385)

[default] com.rti.dds.topic.PrintFormatKind.DEFAULT_PRINT_FORMAT (p. 1386)

Referenced by **DynamicData.to_string()**.

8.220.2.2 pretty_print

```
boolean pretty_print = true
```

Choose whether to print a data sample in a more readable format or to eliminate all white space.

A value of com.rti.dds.infrastructure.true will add indentation and newlines to the string representation of the data sample making it more readable. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

A value of com.rti.dds.infrastructure.false will cause all white space in the string representation of the data sample to be eliminated. For example:

```
<FooType><myLong>5</myLong></FooType>
```

[default] com.rti.dds.infrastructure.true

Referenced by **DynamicData.to_string()**.

8.220.2.3 enum_as_int

```
boolean enum_as_int = false
```

Choose whether to print enum values as integers or as strings.

A value of com.rti.dds.infrastructure.true will cause enumeration literals to be printed using their integer value. For example:

```
<FooType>
   <myEnum>5</myEnum>
</FooType>
```

A value of com.rti.dds.infrastructure.false will cause enumeration literals to be printed as strings using their member names. For example:

```
<FooType>
   <myEnum>RED</myEnum>
</FooType>
```

[default] com.rti.dds.infrastructure.false

Referenced by **DynamicData.to_string()**.

8.220.2.4 include_root_elements

```
boolean include_root_elements = true
```

Choose whether or not to include the root elements of the print format in the output string.

This field only applies if the **com.rti.dds.topic.PrintFormatProperty.kind** (p. 1387) is **com.rti.dds.topic.PrintFormatKind.XML_PRINT_FORMAT** (p. 1386) or **com.rti.dds.topic.PrintFormatKind.JSON_PRINT_FORMAT** (p. 1386)

If the value is com.rti.dds.infrastructure.true and the kind is **com.rti.dds.topic.PrintFormatKind.XML_PRINT_FORMAT** (p. 1386) then a top-level tag with the type's name in it will be included in the output. For example:

```
<FooType>
   <myLong>5</myLong>
</FooType>
```

If the value is com.rti.dds.infrastructure.true and the kind is **com.rti.dds.topic.PrintFormatKind.JSON_PRINT_FORMAT** (p. 1386) then top-level opening and closing braces will be included in the output. For example:

```
{
   "myLong":5
}
```

If the value is com.rti.dds.infrastructure.false the elements described above will not be included in the output. For example:

```
<myLong>5</myLong>
"myLong":5
```

[default] com.rti.dds.infrastructure.true

Referenced by **DynamicData.to_string()**.

8.221 PRIVATE_MEMBER Class Reference

Constant used to indicate that a value type member is private.

Static Public Attributes

- static final short **VALUE**

8.221.1 Detailed Description

Constant used to indicate that a value type member is private.

See also

com.rti.dds.typecode.Visibility.PUBLIC_MEMBER

8.221.2 Member Data Documentation

8.221.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

8.222 ProductVersion_t Class Reference

<<***extension***>> *(p. 155)* Type used to represent the current version of RTI Connext.

Inherits Struct, and Comparable< ProductVersion_t >.

Public Member Functions

- **ProductVersion_t** ()
  Constructor.
- String **toVersionString** ()
- String **toVersionString** (boolean isMicro)
- String **toString** ()

Public Attributes

- char **major**
  Major product version.
- char **minor**
  Minor product version.
- char **release**
  Release letter for product version.
- char **revision**
  Revision number of product.

Static Public Attributes

- static final **ProductVersion_t PRODUCTVERSION_UNKNOWN**
  The value used when the product version is unknown.

8.222.1 Detailed Description

<<***extension***>> *(p. 155)* Type used to represent the current version of RTI Connext.

8.222.2 Constructor & Destructor Documentation

8.222.2.1 ProductVersion_t()

```
ProductVersion_t ( )
```

Constructor.

8.222.3 Member Function Documentation

8.222.3.1 toVersionString() [1/2]

```
String toVersionString ( )
```

A convenience method since most users of this API will not be using Micro.

Returns

Returns **toVersionString(boolean)** (p. 1391) with false as the argument.

References **ProductVersion_t.toVersionString()**.

Referenced by **ProductVersion_t.toVersionString()**.

8.222.3.2 toVersionString() [2/2]

```
String toVersionString (
            boolean isMicro )
```

Provide a simple string representation of an RTI version number. This method will drop revisions that are zero, following the RTI convention.

Parameters

isMicro	true if the string representation should be created for a Micro version, false otherwise.

Returns

The version as a string.

References **ProductVersion_t.major**, **ProductVersion_t.minor**, **ProductVersion_t.release**, and **ProductVersion_t.revision**.

8.222.3.3 toString()

```
String toString ( )
```

Provide a string representation for an instance of this type.

References **ProductVersion_t.major**, **ProductVersion_t.minor**, **ProductVersion_t.release**, and **ProductVersion_t.revision**.

8.222.4 Member Data Documentation

8.222.4.1 PRODUCTVERSION_UNKNOWN

```
final ProductVersion_t PRODUCTVERSION_UNKNOWN [static]
```

Initial value:

```
=
                new ProductVersion_t((char)0, (char)0, (char)0, (char)0)
```

The value used when the product version is unknown.

8.222.4.2 major

```
char major
```

Major product version.

Referenced by **ProductVersion_t.toString()**, and **ProductVersion_t.toVersionString()**.

8.222.4.3 minor

```
char minor
```

Minor product version.

Referenced by **ProductVersion_t.toString()**, and **ProductVersion_t.toVersionString()**.

8.222.4.4 release

```
char release
```

Release letter for product version.

Referenced by **ProductVersion_t.toString()**, and **ProductVersion_t.toVersionString()**.

8.222.4.5 revision

```
char revision
```

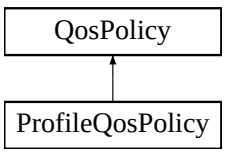
Revision number of product.

Referenced by **ProductVersion_t.toString()**, and **ProductVersion_t.toVersionString()**.

8.223 ProfileQosPolicy Class Reference

Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

Inheritance diagram for ProfileQosPolicy:



Public Attributes

- final **StringSeq string_profile**

  Sequence of strings containing a XML document to load.
- final **StringSeq url_profile**

  *Sequence of **URL groups** (p. 122) containing a set of XML documents to load.*
- boolean **ignore_user_profile**

  Ignores the file USER_QOS_PROFILES.xml in the current working directory.
- boolean **ignore_environment_profile**

  *Ignores the value of the **NDDS_QOS_PROFILES environment variable** (p. 122).*
- boolean **ignore_resource_profile**

  Ignores the file NDDS_QOS_PROFILES.xml.

8.223.1 Detailed Description

Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

All QoS values for Entities can be configured in QoS profiles defined in XML documents. XML documents can be passed to RTI Connext in string form or, more likely, through files found on a file system.

There are also default locations where DomainParticipants will look for files to load QoS profiles. These include the current working directory from where an application is started, a file in the distribution directory for RTI Connext, and the locations specified by an environment variable.

You may disable any or all of these default locations using the Profile QoS policy.

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 761)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **Changeable** (p. 256)

8.223.2 Member Data Documentation

8.223.2.1 string_profile

```
final  StringSeq string_profile
```

Sequence of strings containing a XML document to load.

The concatenation of the strings in this sequence must be a valid XML document according to the XML QoS profile schema.

[default] Empty sequence (zero-length).

8.223.2.2 url_profile

```
final  StringSeq url_profile
```

Sequence of **URL groups** (p. 122) containing a set of XML documents to load.

Only one of the elements of each group will be loaded by RTI Connext, starting from the left.

[default] Empty sequence (zero-length).

8.223.2.3 ignore_user_profile

```
boolean ignore_user_profile
```

Ignores the file USER_QOS_PROFILES.xml in the current working directory.

When this field is set to com.rti.dds.infrastructure.true, the QoS profiles contained in the file USER_QOS_PROFILES.↩ xml in the current working directory will be ignored.

[default] com.rti.dds.infrastructure.false

8.223.2.4 ignore_environment_profile

```
boolean ignore_environment_profile
```

Ignores the value of the **NDDS_QOS_PROFILES environment variable** (p. 122).

When this field is set to com.rti.dds.infrastructure.true, the value of the environment variable NDDS_QOS_PROFILES will be ignored.

[default] com.rti.dds.infrastructure.false

8.223.2.5 ignore_resource_profile

```
boolean ignore_resource_profile
```

Ignores the file NDDS_QOS_PROFILES.xml.

When this field is set to com.rti.dds.infrastructure.true, the QoS profiles contained in the file NDDS_QOS_PROFILES.↩ xml in $NDDSHOME/resource/xml will be ignored.

[default] com.rti.dds.infrastructure.false

8.224 Property_t Class Reference

Properties are name/value pairs objects.

Inherits Struct.

Public Member Functions

- **Property_t** ()

  Constructor.
- **Property_t** (**Property_t** src)

  Constructor.
- **Property_t** (String **name**, String **value**, boolean **propagate**)

  Constructor.
- boolean **valueAsBoolean** ()

  Converts the property value into a boolean.

Public Attributes

- String **name**

  Property name.
- String **value**

  Property value.
- boolean **propagate**

  Indicates if the property must be propagated on discovery.

8.224.1 Detailed Description

Properties are name/value pairs objects.

8.224.2 Constructor & Destructor Documentation

8.224.2.1 Property_t() [1/3]

```
Property_t ( )
```

Constructor.

8.224.2.2 Property_t() [2/3]

```
Property_t (
            Property_t src )
```

Constructor.

Parameters

src	<<***in***>> *(*p. *156)* Property used to initialized the new property.

References **Property_t.name**, **Property_t.propagate**, and **Property_t.value**.

8.224.2.3 Property_t() [3/3]

```
Property_t (
            String name,
            String value,
            boolean propagate )
```

Constructor.

Parameters

name	<<***in***>> *(*p. *156)* Property name.
value	<<***in***>> *(*p. *156)* Property value.
propagate	<<***in***>> *(*p. *156)* Parameter used to indicates whether or not the property must be propagated.

References **Property_t.name**, **Property_t.propagate**, and **Property_t.value**.

8.224.3 Member Function Documentation

8.224.3.1 valueAsBoolean()

```
boolean valueAsBoolean ( )
```

Converts the property value into a boolean.

The boolean returned represents the value true if the property value is not null and is equal, ignoring case, to the string "true", "yes", or "1".

Returns

the boolean represented by the property value

References **Property_t.value**.

8.224.4 Member Data Documentation

8.224.4.1 name

```
String name
```

Property name.

Referenced by **PropertyQosPolicyHelper.get_properties()**, **PropertyQosPolicyHelper.lookup_property()**, and **Property_t.Property_t()**.

8.224.4.2 value

```
String value
```

Property value.

Referenced by **PropertyQosPolicyHelper.get_properties()**, **Property_t.Property_t()**, and **Property_t.valueAs↩ Boolean()**.

8.224.4.3 propagate

```
boolean propagate
```

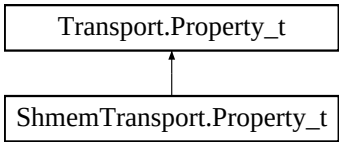
Indicates if the property must be propagated on discovery.

Referenced by **PropertyQosPolicyHelper.get_properties()**, and **Property_t.Property_t()**.

8.225 ShmemTransport.Property_t Class Reference

Subclass of **com.rti.ndds.transport.Transport.Property_t** (p. 1401) allowing specification of parameters that are specific to the shared-memory transport.

Inheritance diagram for ShmemTransport.Property_t:



Public Member Functions

- **Property_t** ()

Public Attributes

- int **received_message_count_max**

  Number of messages that can be buffered in the receive queue.
- int **receive_buffer_size**

  The total number of bytes that can be buffered in the receive queue.

Additional Inherited Members

8.225.1 Detailed Description

Subclass of **com.rti.ndds.transport.Transport.Property_t** (p. 1401) allowing specification of parameters that are specific to the shared-memory transport.

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.225.2 Constructor & Destructor Documentation

8.225.2.1 Property_t()

```
Property_t ( )
```

Create an empty **ShmemTransport** (p. 1681) property with default values

References **ShmemTransport.CLASSID**, **ShmemTransport.MESSAGE_SIZE_MAX_DEFAULT**, **Shmem↩ Transport.Property_t.receive_buffer_size**, **ShmemTransport.RECEIVE_BUFFER_SIZE_DEFAULT**, **Shmem↩ Transport.Property_t.received_message_count_max**, and **ShmemTransport.RECEIVED_MESSAGE_COUNT_↩ MAX_DEFAULT**.

8.225.3 Member Data Documentation

8.225.3.1 received_message_count_max

```
int received_message_count_max
```

Number of messages that can be buffered in the receive queue.

This does not guarantee that the Transport-Plugin will actually be able to buffer `received_message_count_max` messages of the maximum size set in **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404). The total number of bytes that can be buffered for a transport plug-in is actually controlled by `receive_buffer_size`.

See also

NDDS_Transport_Property_t, NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.225.3.2 receive_buffer_size

```
int receive_buffer_size
```

The total number of bytes that can be buffered in the receive queue.

This number controls how much memory is allocated by the plugin for the receive queue. The actual number of bytes allocated is:

```
size = receive_buffer_size + message_size_max +
       received_message_count_max * fixedOverhead
```

where `fixedOverhead` is some small number of bytes used by the queue data structure. The following rules are noted:

- `receive_buffer_size` < `message_size_max` * `received_message_count_max`, then the transport plugin will not be able to store `received_message_count_max` messages of size `message_size_max`.
- `receive_buffer_size` > `message_size_max` * `received_message_count_max`, then there will be memory allocated that cannot be used by the plugin and thus wasted.

To optimize memory usage, the user is allowed to specify a size for the receive queue to be less than that required to hold the maximum number of messages which are all of the maximum size.

In most situations, the average message size may be far less than the maximum message size. So for example, if the maximum message size is 64 K bytes, and the user configures the plugin to buffer at least 10 messages, then 640 K bytes of memory would be needed if all messages were 64 K bytes. Should this be desired, then `receive_buffer_size` should be set to 640 K bytes.

However, if the average message size is only 10 K bytes, then the user could set the `receive_buffer_size` to 100 K bytes. This allows the user to optimize the memory usage of the plugin for the average case and yet allow the plugin to handle the extreme case.

NOTE, the queue will always be able to hold 1 message of `message_size_max` bytes, no matter what the value of receive_buffer_size is.

See also

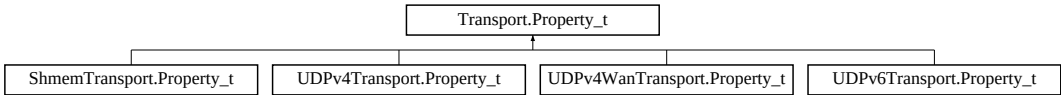
NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.226 Transport.Property_t Class Reference

Base configuration structure that must be inherited by derived **Transport** (p. 1841) Plugin classes.

Inheritance diagram for Transport.Property_t:



Public Attributes

- final int **classid**

  The Transport-Plugin Class ID.
- final int **address_bit_count**

  Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.
- final int **properties_bitmap**

  A bitmap that defines various properties of the transport to the RTI Connext core.
- int **gather_send_buffer_count_max**

  Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.
- int **message_size_max**

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.
- final **StringSeq allow_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.
- final **StringSeq deny_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` > 0), deny the use of these interfaces.
- final **StringSeq allow_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_multicast_interfaces_list_length` > 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.
- final **StringSeq deny_multicast_interfaces_list**

  A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_multicast_interfaces_list_length` > 0), deny the use of those interfaces for multicast.
- byte[ ] **transport_uuid** = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

  Univocally identifies a transport plugin.
- int **max_interface_count**

  Maximum number of addresses from the `allow_interfaces_list` that will be announced.

Static Public Attributes

- static final int **NDDS_TRANSPORT_CLASSID_INVALID** = -1

  *Invalid **Transport** (p. 1841) Class ID.*
- static final int **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** = 1000

  Transport-Plugin class IDs below this are reserved by RTI.
- static final int **NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED** = 0x2

  Specified zero-copy behavior of transport.
- static final int **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN** = 3

  *Minimum number of gather-send buffers that must be supported by a **Transport** (p. 1841) Plugin implementation.*

8.226.1 Detailed Description

Base configuration structure that must be inherited by derived **Transport** (p. 1841) Plugin classes.

This structure contains properties that must be set before registration of any transport plugin with RTI Connext. The RTI Connext core will configure itself to use the plugin based on the properties set within this structure.

A transport plugin may extend from this structure to add transport-specific properties.

WARNING: The transport properties of an instance of a **Transport** (p. 1841) Plugin should be considered immutable after the plugin has been created. That means the values contained in the property structure stored as a part of the transport plugin itself should not be changed. If those values are modified, the results are undefined.

8.226.2 Member Data Documentation

8.226.2.1 NDDS_TRANSPORT_CLASSID_INVALID

```
final int NDDS_TRANSPORT_CLASSID_INVALID = -1  [static]
```

Invalid **Transport** (p. 1841) Class ID.

Transport-Plugins implementations should set their class ID to a value different than this.

8.226.2.2 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE

```
final int NDDS_TRANSPORT_CLASSID_RESERVED_RANGE = 1000  [static]
```

Transport-Plugin class IDs below this are reserved by RTI.

User-defined Transport-Plugins should use a class ID greater than this number.

8.226.2.3 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED

```
final int NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED = 0x2  [static]
```

Specified zero-copy behavior of transport.

A **Transport** (p. 1841) Plugin may commit to one of three behaviors for zero copy receives:

1. Always does zero copy.
2. Sometimes does zero copy, up to the transport discretion.
3. Never does zero copy.

This bit should be set only if the **Transport** (p. 1841) Plugin commits to always doing a zero copy receive, or more specifically, always loaning a buffer through its `receive_rEA()` call.

In that case, the NDDS core will not need to allocate storage for a message that it retrieves with the `receive_rEA()` call.

8.226.2.4 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN

```
final int NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN = 3  [static]
```

Minimum number of gather-send buffers that must be supported by a **Transport** (p. 1841) Plugin implementation.

For the NDDS_Transport_Property_t structure to be valid, the value of **Transport.Property_t.gather_send_buffer_↩ count_max** (p. 1404) must be greater than or equal to this value.

8.226.2.5 classid

```
final int classid
```

The Transport-Plugin Class ID.

The transport plugin sets the value for this field.

Class IDs below **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 1402) are reserved for RTI (Real-Time Innovations) usage.

User-defined transports should set an ID above this range.

The ID should be globally unique for each Transport-Plugin class. Transport-Plugin implementors should ensure that the class IDs do not conflict with each other amongst different Transport-Plugin classes.

Invariant

The `classid` is invariant for the lifecycle of a transport plugin.

8.226.2.6 address_bit_count

```
final int address_bit_count
```

Number of bits in a 16-byte address that are used by the transport. Should be between -128 and 128.

The transport plugin sets the value for this field.

A transport plugin should define the range of addresses that are meaningful to the plugin. It does this by setting the number of bits of an IPv6 address that will be used to designate an address in the network to which the transport plugin is connected. The sign of this field determines whether the count of bits starts on the most significant bit (negative sign) or on the least significant bit (positive sign).

For example, for an address range of 0-255, the address_bit_count should be set to 8. For the range of addresses used by IPv4 (4 bytes), it should be set to 32.

See also

Transport Class Attributes (p. 96)

8.226.2.7 properties_bitmap

```
final int properties_bitmap
```

A bitmap that defines various properties of the transport to the RTI Connext core.

Currently, the only property supported is whether or not the transport plugin will always loan a buffer when RTI Connext tries to receive a message using the plugin. This is in support of a zero-copy interface.

See also

NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED (p. 1402)

8.226.2.8 gather_send_buffer_count_max

```
int gather_send_buffer_count_max
```

Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a transport plugin.

The transport plugin `send()` API supports a gather-send concept, where the `send()` call can take several discontiguous buffers, assemble and send them in a single message. This enables RTI Connext to send a message from parts obtained from different sources without first having to copy the parts into a single contiguous buffer.

However, most transports that support a gather-send concept have an upper limit on the number of buffers that can be gathered and sent. Setting this value will prevent RTI Connext from trying to gather too many buffers into a send call for the transport plugin.

RTI Connext requires all transport-plugin implementations to support a gather-send of least a minimum number of buffers. This minimum number is defined to be **NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_↩ COUNT_MIN** (p. 1402).

If the underlying transport does not support a gather-send concept directly, then the transport plugin itself must copy the separate buffers passed into the `send()` call into a single buffer for sending or otherwise send each buffer individually. However this is done by the transport plugin, the receive_rEA() call of the destination application should assemble, if needed, all of the pieces of the message into a single buffer before the message is passed to the RTI Connext layer.

8.226.2.9 message_size_max

```
int message_size_max
```

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin.

If the maximum size of a message that can be sent by a transport plugin is user configurable, the transport plugin should provide a default value for this property. In any case, this value must be set before the transport plugin is registered, so that RTI Connext can properly use the plugin.

For information about the default value for different transports, see the `Core Libraries User's Manual`.

8.226.2.10 allow_interfaces_list

`final` **StringSeq** `allow_interfaces_list`

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `allow_interfaces_list_length` > 0), allow the use of only these interfaces. If the list is empty, allow the use of all interfaces.

The "white" list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

The left-to-right order of this list matters if you are using the max_interface_count to limit the allowable interfaces further. See `max_interface_count`.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) is deleted.

[default] empty list that represents all available interfaces.

See also

com.rti.ndds.transport.Transport.Property_t.max_interface_count (p. 1406)

8.226.2.11 deny_interfaces_list

`final` **StringSeq** `deny_interfaces_list`

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., `deny_interfaces_list_length` > 0), deny the use of these interfaces.

This "black" list is applied *after* the `allow_interfaces_list` and filters out the interfaces that should not be used.

The resulting list restricts *reception* to a particular set of interfaces for unicast UDP.

For UDPv4 and UDPv6 transports, the resulting list also controls the interfaces over which the DomainParticipant will send multicast traffic to the remote DomainParticipants (if multicast is supported on the platform).

Note: This property does not affect the interfaces that the transport uses to send unicast data from that Domain↩ Participant. That decision is made by the OS based on the destination address.

It is up to the transport plugin to interpret the list of strings passed in.

For example, the following are acceptable strings in IPv4 format: 192.168.1.1, 192.168.1.∗, 192.168.∗, 192.∗, ether0

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) is deleted.

[default] empty list that represents no denied interfaces.

8.226.2.12 allow_multicast_interfaces_list

```
final StringSeq allow_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.↩ e., allow_multicast_interfaces_list_length > 0), allow the use of multicast only on these interfaces; otherwise allow the use of all the allowed interfaces.

This "white" list sub-selects from the allowed interfaces obtained *after* applying the allow_interfaces_list "white" list *and* the deny_interfaces_list "black" list.

After allow_multicast_interfaces_list, the deny_multicast_interfaces_list is applied. Multicast output will be sent and may be received over the interfaces in the resulting list.

If this list is empty, all the allowed interfaces will be potentially used for multicast. It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) is deleted.

[default] empty list that represents all available interfaces.

8.226.2.13 deny_multicast_interfaces_list

```
final StringSeq deny_multicast_interfaces_list
```

A list of strings, each identifying a range of interface addresses or an interface name. If the list is non-empty (i.e., deny_multicast_interfaces_list_length > 0), deny the use of those interfaces for multicast.

This "black" list is applied after allow_multicast_interfaces_list and filters out interfaces that should not be used for multicast.

Multicast output will be sent and may be received over the interfaces in the resulting list.

It is up to the transport plugin to interpret the list of strings passed in.

This property is not interpreted by the RTI Connext core; it is provided merely as a convenient and standardized way to specify the interfaces for the benefit of the transport plugin developer and user.

The caller (user) must manage the memory of the list. The memory may be freed after the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) is deleted.

[default] empty list that represents no denied interfaces.

8.226.2.14 transport_uuid

```
byte [] transport_uuid = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Univocally identifies a transport plugin.

It represents the prefix of the participant GUID.

8.226.2.15 max_interface_count

```
int max_interface_count
```

Maximum number of addresses from the allow_interfaces_list that will be announced.

By default, max_interface_count = LENGTH_UNLIMITED: all the interfaces are announced.

This feature is useful if you want to control the network interfaces on which your DomainParticipants receive data. For example, if you have one wired and one wireless interface both up and running, and max_interface_count is set to 1, the DomainParticipant will receive data over the interface you list first in the allow_interfaces_list -for example, the wired one.

RTI Connext selects the preferred interface(s) by iterating over the list of allowed interfaces until the first max_↩ interfaces_count of active interfaces encountered are announced. The order of iteration is left to right as specified in the allow_interfaces_list setting.

This setting applies only if the allow_interfaces_list is not empty.

The max_interface_count setting does not consider end-to-end connectivity to select interfaces. The decision is based purely on whether interfaces are up or down in a node. Therefore, this feature is not intended to be used in the following scenarios:

- A DomainParticipant is not reachable by other DomainParticipants in all the interfaces in the allow_↩ interfaces_list. This could occur if the DomainParticipant is in different subnets, and some of these subnets cannot be reached by other DomainParticipants.
- End-to-end connectivity issues lead to situations in which the interfaces selected after applying max_↩ interface_count cannot be reached by other DomainParticipants.

For UDPv4 and UDPv6 transports, this feature also affects multicast traffic by limiting the interfaces over which a DomainParticipant sends multicast traffic. The (allow/deny)_multicast_interfaces_list applies to the interfaces selected by using the max_interfaces_count property.

Note: If a pattern string in the allow_interfaces_list matches multiple interface addresses, and max_↩ interface_count is set to a finite value, the order for the matching allowed interfaces is decided based on the order in which the operating system provides these interfaces.

[default] LENGTH_UNLIMITED

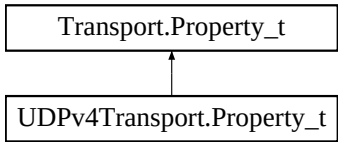
See also

com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list (p. 1404)

8.227 UDPv4Transport.Property_t Class Reference

Configurable IPv4/UDP Transport-Plugin properties.

Inheritance diagram for UDPv4Transport.Property_t:



Public Member Functions

- **Property_t** ()

Public Attributes

- int **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- int **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- int **unicast_enabled**

  Allows the transport plugin to use unicast for sending and receiving.
- int **multicast_enabled**

  Allows the transport plugin to use multicast for sending and receiving.
- int **multicast_ttl**

  Value for the time-to-live parameter for all multicast sends using this plugin.
- int **multicast_loopback_disabled**

  Prevents the transport plugin from putting multicast packets onto the loopback interface.
- int **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- int **ignore_nonup_interfaces**

  ***[DEPRECATED]** Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- int **ignore_nonrunning_interfaces**

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

  ***[DEPRECATED]** Prevents the transport plugin from doing a zero copy.*
- int **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- int **use_checksum**

  Configures whether to send UDP checksum.
- long **transport_priority_mask**

  Set mask for use of transport priority field.
- int **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- int **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- int **send_ping**

  Configures whether to send PING messages.
- int **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.
- long **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- int **reuse_multicast_receive_resource**

  Controls whether or not to reuse multicast receive resources.
- int **protocol_overhead_max**

Maximum size in bytes of protocol overhead, including headers.

- int **disable_interface_tracking**

  Disables detection of network interface changes.

- long **join_multicast_group_timeout**

  [Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

- String **public_address**

  Public IP address associated with the transport instantiation.

- int **port_offset**

  Port offset to allow multiple instances of UDPv4 transports in the same ***com.rti.dds.domain.DomainParticipant*** *(p. 670).*

Additional Inherited Members

8.227.1 Detailed Description

Configurable IPv4/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.227.2 Constructor & Destructor Documentation

8.227.2.1 Property_t()

```
Property_t ( )
```

Create an empty **UDPv4Transport** (p. 1943) property with default values

References **UDPv4Transport.BLOCKING_ALWAYS**, **UDPv4Transport.CLASSID**, **UDPv4Transport.Property_t.disable_interface_tracking**, **UDPv4Transport.Property_t.force_interface_poll_detection**, **UDPv4Transport.Property_t.ignore_loopback_interface**, **UDPv4Transport.Property_t.ignore_nonrunning_interfaces**, **UDPv4Transport.Property_t.ignore_nonup_interfaces**, **UDPv4Transport.Property_t.interface_poll_period**, **UDPv4Transport.Property_t.join_multicast_group_timeout**, **UDPv4Transport.MESSAGE_SIZE_MAX_DEFAULT**, **UDPv4Transport.Property_t.multicast_enabled**, **UDPv4Transport.Property_t.multicast_loopback_disabled**, **UDPv4Transport.Property_t.multicast_ttl**, **UDPv4Transport.Property_t.no_zero_copy**, **UDPv4Transport.Property_t.port_offset**, **UDPv4Transport.Property_t.protocol_overhead_max**, **UDPv4Transport.Property_t.public_address**, **UDPv4Transport.Property_t.recv_socket_buffer_size**, **UDPv4Transport.RECV_SOCKET_BUFFER_SIZE_DEFAULT**, **UDPv4Transport.Property_t.reuse_multicast_receive_resource**, **UDPv4Transport.Property_t.send_blocking**, **UDPv4Transport.Property_t.send_ping**, **UDPv4Transport.Property_t.send_socket_buffer_size**, **UDPv4Transport.SEND_SOCKET_BUFFER_SIZE_DEFAULT**, **UDPv4Transport.Property_t.transport_priority_mapping_high**, **UDPv4Transport.Property_t.transport_priority_mapping_low**, **UDPv4Transport.Property_t.transport_priority_mask**, **UDPv4Transport.Property_t.unicast_enabled**, and **UDPv4Transport.Property_t.use_checksum**.

8.227.3 Member Data Documentation

8.227.3.1 send_socket_buffer_size

```
int 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 **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404). The maximum value is operating system-dependent.

By default, it will be set to com.rti.ndds.transport.UDPv4Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT.

If you configure this parameter to be com.rti.ndds.transport.UDPv4Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT, then setsockopt() (or equivalent) will not be called to size the send buffer of the socket. The transport will use the OS default.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.2 recv_socket_buffer_size

```
int 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 **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404). The maximum value is operating system-dependent.

By default, it will be set to com.rti.ndds.transport.UDPv4Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT.

If you configure this parameter to be com.rti.ndds.transport.UDPv4Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT, then setsockopt() (or equivalent) will not be called to size the receive buffer of the socket. The transport will use the OS default.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.3 unicast_enabled

```
int unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.4 multicast_enabled

```
int multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use all the network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.5 multicast_ttl

```
int multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This value is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.6 multicast_loopback_disabled

```
int multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

[NOTE: Windows CE systems do not support multicast loopback. This field is ignored for Windows CE targets.]

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.7 ignore_loopback_interface

```
int ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.8 ignore_nonup_interfaces

```
int ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **com.rti.ndds.transport.UDPv4Transport.Property_t.disable_interface_tracking** (p. 1417) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.9 ignore_nonrunning_interfaces

```
int ignore_nonrunning_interfaces
```

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

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.10 no_zero_copy

```
int no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.11 send_blocking

```
int send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.12 use_checksum

```
int use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **com.rti.dds.domain.DomainParticipant.get_participant_protocol_status** (p. 743).

[default] 1 (enabled)

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.13 transport_priority_mask

```
long transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_low** (p. 1415) and **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_high** (p. 1415) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 274)) 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.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.14 transport_priority_mapping_low

```
int transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mask** (p. 1415) and **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_high** (p. 1415) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.15 transport_priority_mapping_high

```
int transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mask** (p. 1415) and **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_low** (p. 1415) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.16 send_ping

```
int send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.17 force_interface_poll_detection

```
int force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.18 interface_poll_period

```
long interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.19 reuse_multicast_receive_resource

```
int reuse_multicast_receive_resource
```

Controls whether or not to reuse multicast receive resources.

Setting this to 0 (FALSE) prevents multicast crosstalk by uniquely configuring a port and creating a receive thread for each multicast group address.

[default] 1.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.20 protocol_overhead_max

```
int protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size_max, to 65535 minus this overhead.

[default] 28.

See also

com.rti.ndds.transport.Transport.Property_t.message_size_max (p. 1404)

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.21 disable_interface_tracking

```
int disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.22 join_multicast_group_timeout

```
long join_multicast_group_timeout
```

[Windows only] Defines how much time (milliseconds) to wait to join a multicast group address when a new interface is detected.

On Windows, a network interface may be detected before it is allowed to join a multicast group address. This property adjusts how much time (milliseconds) to wait for the ADD_MEMBERSHIP multicast operation to succeed before withdrawing.

[default] 5000

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.23 public_address

```
String public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary to support communication over WAN that involves Network Address Translation (NAT).

Typically, the address is the public address of the IP NAT router that provides access to the WAN.

By default, the **com.rti.dds.domain.DomainParticipant** (p. 670) creating the transport will announce the IP addresses obtained from the NICs to other DomainParticipants in the system.

When this property is set, the DomainParticipant will announce the IP address corresponding to the property value instead of the LAN IP addresses associated with the NICs.

Note 1: Setting this property is necessary, but is not a sufficient condition for sending and receiving data over the WAN. You must also configure the IP NAT router to allow UDP traffic and to map the public IP address specified by this property to the DomainParticipant's private LAN IP address. This is typically done with one of the following mechanisms:

- Port Forwarding: You must map the private ports used to receive discovery and user data traffic to the corresponding public ports (see **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622)). Public and private ports must be the same since the transport does not allow you to change the mapping.
- 1:1 NAT: You must add a 1:1 NAT entry that maps the public IP address specified in this property to the private LAN IP address of the DomainParticipant.

Note 2: By setting this property, the **com.rti.dds.domain.DomainParticipant** (p. 670) only announces its public IP address to other DomainParticipants. Therefore, communication with DomainParticipants within the LAN that are running on different nodes will not work unless the NAT router is configured to enable NAT reflection (hairpin NAT).

There is another way to achieve simultaneous communication with DomainParticipants running in the LAN and WAN, that does not require hairpin NAT. This way uses a gateway application such as RTI Routing Service to provide access to the WAN.

[default] null (the transport uses the IP addresses obtained from the NICs)

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.227.3.24 port_offset

```
int port_offset
```

Port offset to allow multiple instances of UDPv4 transports in the same **com.rti.dds.domain.DomainParticipant** (p. 670).

By default, it is not possible to have multiple instances of the UDPv4 transport in the same **com.rti.dds.domain.↩ DomainParticipant** (p. 670). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv4 transport in the same **com.rti.dds.domain.Domain↩ Participant** (p. 670).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

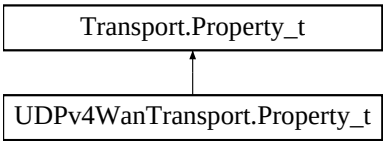
[default] 0

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.228 UDPv4WanTransport.Property_t Class Reference

Configurable IPv4/UDP WAN Transport-Plugin properties.

Inheritance diagram for UDPv4WanTransport.Property_t:



Public Member Functions

- **Property_t** ()

Public Attributes

- int **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- int **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- int **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- int **ignore_nonup_interfaces**

  ***[DEPRECATED]** Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.*
- int **ignore_nonrunning_interfaces**

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

  ***[DEPRECATED]** Prevents the transport plugin from doing a zero copy.*
- int **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- int **use_checksum**

  Configures whether to send UDP checksum.
- long **transport_priority_mask**

  Set mask for use of transport priority field.
- int **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.
- int **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.
- int **send_ping**

  Configures whether to send PING messages.
- int **force_interface_poll_detection**

  Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

- long **interface_poll_period**

  Specifies the period in milliseconds to query for changes in the state of all the interfaces.
- int **protocol_overhead_max**

  Maximum size in bytes of protocol overhead, including headers.
- int **disable_interface_tracking**

  Disables detection of network interface changes.
- String **public_address**

  Public IP address associated with the transport instantiation.
- **TransportUdpWanCommPortsMappingInfoSeq comm_ports**

  Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.
- int **port_offset**

  Port offset to allow coexistence with built-in UDPv4 transport.
- long **binding_ping_period**

  Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

Additional Inherited Members

8.228.1 Detailed Description

Configurable IPv4/UDP WAN Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.228.2 Constructor & Destructor Documentation

8.228.2.1 Property_t()

```
Property_t ( )
```

Create an empty **UDPv4WanTransport** (p. 1948) property with default values

References **UDPv4WanTransport.Property_t.binding_ping_period**, **UDPv4WanTransport.Property_t.comm_ports**, **UDPv4WanTransport.Property_t.disable_interface_tracking**, **UDPv4WanTransport.Property_t.force_interface_poll_detection**, **UDPv4WanTransport.Property_t.ignore_loopback_interface**, **UDPv4WanTransport.Property_t.ignore_nonrunning_interfaces**, **UDPv4WanTransport.Property_t.ignore_nonup_interfaces**, **UDPv4WanTransport.Property_t.interface_poll_period**, **UDPv4WanTransport.Property_t.no_zero_copy**, **UDPv4WanTransport.Property_t.port_offset**, **UDPv4WanTransport.Property_t.protocol_overhead_max**, **UDPv4WanTransport.Property_t.public_address**, **UDPv4WanTransport.Property_t.recv_socket_buffer_size**, **UDPv4WanTransport.Property_t.send_blocking**, **UDPv4WanTransport.Property_t.send_ping**, **UDPv4WanTransport.Property_t.send_socket_buffer_size**, **UDPv4WanTransport.Property_t.transport_priority_mapping_high**, **UDPv4WanTransport.Property_t.transport_priority_mapping_low**, **UDPv4WanTransport.Property_t.transport_priority_mask**, and **UDPv4WanTransport.Property_t.use_checksum**.

8.228.3 Member Data Documentation

8.228.3.1 send_socket_buffer_size

```
int send_socket_buffer_size
```

Size in bytes of the send buffer of a socket used for sending.

See also

com.rti.ndds.transport.UDPv4Transport.Property_t.send_socket_buffer_size (p. 1410)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.2 recv_socket_buffer_size

```
int recv_socket_buffer_size
```

Size in bytes of the receive buffer of a socket used for receiving.

See also

com.rti.ndds.transport.UDPv4Transport.Property_t.recv_socket_buffer_size (p. 1410)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.3 ignore_loopback_interface

```
int ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. The IP loopback interface is not used, even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient plugin (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows:

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UPV4 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv4 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.4 ignore_nonup_interfaces

```
int ignore_nonup_interfaces
```

[DEPRECATED] Prevents the transport plugin from using a network interface that is not reported as UP by the operating system.

DEPRECATED: this property has no effect. Non-UP interfaces are ignored until they change their status to UP, unless **com.rti.ndds.transport.UDPv4WanTransport.Property_t.disable_interface_tracking** (p. 1427) is set to 1, in which case interfaces are ignored even if they change their status at a later point.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows you to configure the transport to start using even the interfaces that were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces that were not reported as UP.
- **1**: Do not use interfaces that were not reported as UP.

[default] 1

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.5 ignore_nonrunning_interfaces

```
int ignore_nonrunning_interfaces
```

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

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- **1**: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.6 no_zero_copy

```
int no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing a zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may err or malfunction. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.7 send_blocking

```
int send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.8 use_checksum

```
int use_checksum
```

Configures whether to send UDP checksum.

This property specifies whether the UDP checksum will be computed. On Windows and Linux, the UDP protocol will not set the checksum when use_checksum is set to 0. This is useful when RTPS protocol statistics related to corrupted messages need to be collected through the API **com.rti.dds.domain.DomainParticipant.get_participant_protocol_status** (p. 743).

[default] 1 (enabled)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.9 transport_priority_mask

```
long transport_priority_mask
```

Set mask for use of transport priority field.

This is used in conjunction with **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mapping_low** (p. 1425) and **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mapping_high** (p. 1425) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 274)) 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.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.10 transport_priority_mapping_low

```
int transport_priority_mapping_low
```

Set low value of output range to IPv4 TOS.

This is used in conjunction with **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mask** (p. 1425) and **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mapping_high** (p. 1425) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.11 transport_priority_mapping_high

```
int transport_priority_mapping_high
```

Set high value of output range to IPv4 TOS.

This is used in conjunction with **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mask** (p. 1425) and **com.rti.ndds.transport.UDPv4WanTransport.Property_t.transport_priority_mapping_low** (p. 1425) to define the mapping from the DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.12 send_ping

```
int send_ping
```

Configures whether to send PING messages.

This property specifies whether to send a PING message before commencing the discovery process. On certain operating systems or with certain switches the initial UDP packet, while configuring the ARP table, can unfortunately be dropped. To avoid dropping the initial RTPS discovery sample, a PING message is sent to preconfigure the ARP table in those environments.

[default] 1 (enabled)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.13 force_interface_poll_detection

```
int force_interface_poll_detection
```

Forces the interface tracker to use a polling mechanism to detect changes on the UDPv4 interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. By setting this property on those OSes, the use of a polling mechanism to detect changes can be forced.

[default] 0 (disabled).

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.14 interface_poll_period

```
long interface_poll_period
```

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

When possible, the detection of an IP address change is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period can be configured by setting this property.

[default] 500 milliseconds.

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.15 protocol_overhead_max

```
int protocol_overhead_max
```

Maximum size in bytes of protocol overhead, including headers.

This value is the maximum size, in bytes, of protocol-related overhead. Normally, the overhead accounts for UDP and IP headers. The default value is set to accommodate the most common UDP/IP header size.

Note that when **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) plus this overhead is larger than the UDPv4 maximum message size (65535 bytes), the middleware will automatically reduce the effective message_size_max, to 65535 minus this overhead.

[default] 28.

See also

com.rti.ndds.transport.Transport.Property_t.message_size_max (p. 1404)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.16 disable_interface_tracking

```
int disable_interface_tracking
```

Disables detection of network interface changes.

By default, network interface changes are propagated in the form of locators to other applications. This is done to support IP mobility scenarios.

For example, you could start a RTI Connext application with Wi-Fi and move to a wired connection. In order to continue communicating with other applications, this interface change has to be propagated.

In RTI Connext 5.2 (the initial release) and earlier versions of the product, IP mobility scenarios were not supported. 5.2 applications will report errors if they detect locator changes in a DataWriter or DataReader.

You can disable the notification and propagation of interface changes by setting this property to 1.

This way, an interface change in a newer application will not trigger errors in an application running 5.2 or earlier. Of course, this will prevent the new application from being able to detect network interface changes.

[default] 0

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.17 public_address

```
String public_address
```

Public IP address associated with the transport instantiation.

Setting the public IP address is only necessary for the Real-Time WAN **Transport** (p. 1841) associated with an external **com.rti.dds.domain.DomainParticipant** (p. 670) (publicly reachable) in order to support the two communication scenarios shown in the Figure below.

For an external **com.rti.dds.domain.DomainParticipant** (p. 670) behind a NAT-enabled router, this address is the public IP address of the router.

When this property is set, the DomainParticipant will announce PUBLIC+UUID locators to other DomainParticipants. These locators are reachable locators because they contain this public IP address.

For additional information on Real-Time WAN **Transport** (p. 1841) locators, see the `Core Libraries User's Manual`.

with a Participant that has a Public Address"

[default] null (the transport will announce UUID locators)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.18 comm_ports

TransportUdpWanCommPortsMappingInfoSeq comm_ports

Configures the public and private UDP ports that a transport instance uses to receive/send RTPS data.

Array containing the mapping between "RTPS ports", "UDP receive host ports", and "UDP receive public ports".

When the transport is configured using properties, the port mapping array is provided using a JSON string.

For example:

```
{
    "default": {"host": 8192, "public": 9678},
    "mappings": [
        {"rtps": 1234, "host": 9999, "public": 5678},
        {"rtps": 1235, "host": 9990, "public": 5679},
    ]
}
```

It is also possible to configure the mapping with XML:

```
<transport_builtin>
    <udpv4_wan>
        <comm_ports>
```

```
            <default>
                <host>8192</host>
                <public>9678</public>
            </default>
            <mappings>
                <element>
                    <rtps>1234</rtps>
                    <host>9999</host>
                    <public>5678</public>
                </element>
                <element>
                    <rtps>1235</rtps>
                    <host>9990</host>
                    <public>5679</public>
                </element>
            </mappings>
        </comm_ports>
    </udpv4_wan>
<transport_builtin>
```

For additional information on how to set the value of this property, see the `Core Libraries User's Manual`.

[default] NULL (The UDP ports used for communications will be derived from the RTPS ports associated with the locators for the DomainParticipant and its Endpoints (DataWriters and DataReaders)).

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.19 port_offset

```
int port_offset
```

Port offset to allow coexistence with built-in UDPv4 transport.

This property allows using the built-in UDPv4 transport and the Real-Time WAN **Transport** (p. 1841) at the same time.

```
<transport_builtin>
    <mask>UDPv4_WAN|UDPv4</mask>
</transport_builtin>
```

When the UDP ports used by Real-Time WAN **Transport** (p. 1841) are not explicitly set, they are calculated as follows: RTPS port + port_offset.

[default] 125

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.228.3.20 binding_ping_period

```
long binding_ping_period
```

Specifies the period in milliseconds at which BINDING PINGS messages are sent to keep NAT mappings open.

Configures the period in milliseconds at which BINDING_PING messages are sent by a local transport instance to a remote transport instance. For example, 1000 means to send BINDING_PING messages every second.

BINDING_PING messages are used on the sending side to open NAT bindings from a local transport instance to a remote transport instance and they are sent periodically to keep the bindings open.

On the receiving side, BINDING_PINGS are used to calculate the public IP transport address of an UUID locator. This address will be used to send data to the locator.

For additional information on the role of BINDING_PING, see the `Core Libraries User's Manual`.

From a configuration point of view, and to avoid communication disruptions, the period at which a transport instance sends BINDING_PING messages should be smaller than the NAT binding session timeout. This timeout depends on the NAT router configuration.

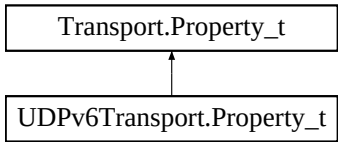
[default] 1000 (1 sec)

Referenced by **UDPv4WanTransport.Property_t.Property_t()**.

8.229 UDPv6Transport.Property_t Class Reference

Configurable IPv6/UDP Transport-Plugin properties.

Inheritance diagram for UDPv6Transport.Property_t:



Public Member Functions

- **Property_t** ()

Public Attributes

- int **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.
- int **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.
- int **unicast_enabled**

  Allows the transport plugin to use unicast for sending and receiving.
- int **multicast_enabled**

  Allows the transport plugin to use multicast for sending and receiving.
- int **multicast_ttl**

  Value for the time-to-live parameter for all multicast sends using this plugin.
- int **multicast_loopback_disabled**

  Prevents the transport plugin from putting multicast packets onto the loopback interface.
- int **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.
- int **ignore_nonrunning_interfaces**

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

  ***[DEPRECATED]** Prevents the transport plugin from doing zero copy.*
- int **send_blocking**

  Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.
- int **enable_v4mapped**

  Specify whether UDPv6 transport will process IPv4 addresses.
- long **transport_priority_mask**

  Set mask for use of transport priority field.
- int **transport_priority_mapping_low**

  Set low value of output range to IPv6 TCLASS.
- int **transport_priority_mapping_high**

  Set high value of output range to IPv6 TCLASS.
- int **port_offset**

  *Port offset to allow multiple instances of UDPv6 transports in the same **com.rti.dds.domain.DomainParticipant** (p. 670).*

Additional Inherited Members

8.229.1 Detailed Description

Configurable IPv6/UDP Transport-Plugin properties.

You can modify the properties in this structure to configure the plugin. However, you must set the properties before the plugin is instantiated.

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.229.2 Constructor & Destructor Documentation

8.229.2.1 Property_t()

```
Property_t ( )
```

Create an empty **UDPv6Transport** (p. 1952) property with default values

References **UDPv6Transport.BLOCKING_ALWAYS**, **UDPv6Transport.CLASSID**, **UDPv6Transport.Property_t.enable_v4mapped**, **UDPv6Transport.Property_t.ignore_loopback_interface**, **UDPv6Transport.Property_t.ignore_nonrunning_interfaces**, **UDPv6Transport.Property_t.multicast_enabled**, **UDPv6Transport.Property_t.multicast_loopback_disabled**, **UDPv6Transport.Property_t.multicast_ttl**, **UDPv6Transport.Property_t.no_zero_copy**, **UDPv6Transport.Property_t.port_offset**, **UDPv6Transport.Property_t.recv_socket_buffer_size**, **UDPv6Transport.Property_t.send_blocking**, **UDPv6Transport.Property_t.send_socket_buffer_size**, **UDPv6Transport.Property_t.transport_priority_mapping_high**, **UDPv6Transport.Property_t.transport_priority_mapping_low**, **UDPv6Transport.Property_t.transport_priority_mask**, and **UDPv6Transport.Property_t.unicast_enabled**.

8.229.3 Member Data Documentation

8.229.3.1 send_socket_buffer_size

```
int 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 **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404). The maximum value is operating system-dependent.

By default, it will be set to com.rti.ndds.transport.UDPv6Transport_SEND_SOCKET_BUFFER_SIZE_DEFAULT.

If you configure this parameter to be com.rti.ndds.transport.UDPv6Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT, then setsockopt() (or equivalent) will not be called to size the send buffer of the socket. The transport will use the OS default.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.2 recv_socket_buffer_size

```
int 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 **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404). The maximum value is operating system-dependent.

By default, it will be set to com.rti.ndds.transport.UDPv6Transport_RECV_SOCKET_BUFFER_SIZE_DEFAULT.

If you configure this parameter to be com.rti.ndds.transport.UDPv6Transport_SOCKET_BUFFER_SIZE_OS_DEFAULT, then setsockopt() (or equivalent) will not be called to size the receive buffer of the socket. The transport will use the OS default.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.3 unicast_enabled

```
int unicast_enabled
```

Allows the transport plugin to use unicast for sending and receiving.

This value turns unicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1). Also by default, the plugin will use all the allowed network interfaces that it finds up and running when the plugin is instantiated.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.4 multicast_enabled

```
int multicast_enabled
```

Allows the transport plugin to use multicast for sending and receiving.

This value turns multicast UDP on (if set to 1) or off (if set to 0) for this plugin. By default, it will be turned on (1) for those platforms that support multicast. Also by default, the plugin will use the all network interfaces allowed for multicast that it finds up and running when the plugin is instantiated.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.5 multicast_ttl

```
int multicast_ttl
```

Value for the time-to-live parameter for all multicast sends using this plugin.

This is used to set the TTL of multicast packets sent by this transport plugin.

[default] 1

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.6 multicast_loopback_disabled

```
int multicast_loopback_disabled
```

Prevents the transport plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.7 ignore_loopback_interface

```
int ignore_loopback_interface
```

Prevents the transport plugin from using the IP loopback interface.

Currently three values are allowed:

- **0**: Forces local traffic to be sent over loopback, even if a more efficient transport (such as shared memory) is installed (in which case traffic will be sent over both transports).
- **1**: Disables local traffic via this plugin. Do not use the IP loopback interface even if no NICs are discovered. This is useful when you want applications running on the same node to use a more efficient transport (such as shared memory) instead of the IP loopback.
- **-1**: Automatic. Lets RTI Connext decide between the above two choices.

The current "automatic" (-1) RTI Connext policy is as follows.

- If a shared memory transport plugin is available for local traffic and there is a locator on the initial peers list that can use shared memory, the effective value is 1 (i.e., disable UDPv6 local traffic).
- Otherwise, the effective value is 0 (i.e., use UDPv6 for local traffic also).

[default] -1 Automatic RTI Connext policy based on availability of the shared memory transport.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.8 ignore_nonrunning_interfaces

```
int ignore_nonrunning_interfaces
```

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

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface that is not reported as UP will not be used. This property allows the same check to be extended to the IFF_↩ RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated," and may be off if there is no link detected, e.g., the network cable is unplugged.

Two values are allowed:

- **0**: Do not check the RUNNING flag when enumerating interfaces, just make sure interface is UP.
- **1**: Check flag when enumerating interfaces and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1 (i.e., check RUNNING flag)

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.9 no_zero_copy

```
int no_zero_copy
```

[DEPRECATED] Prevents the transport plugin from doing zero copy.

DEPRECATED: This property has no effect. By default, this plugin will use the zero copy on OSes that offer it. While this is good for performance, it may sometimes tax the OS resources in a manner that cannot be overcome by the application.

The best example is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may error or malfunction. If you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off the use of zero copy.

By default this is set to 0, so RTI Connext will use the zero copy API if offered by the OS.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.10 send_blocking

```
int send_blocking
```

Control blocking behavior of send sockets. CHANGING THIS FROM THE DEFAULT CAN CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

Currently two values are defined:

- **NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS:** Sockets are blocking (default socket options for the operating system).
- **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.11 enable_v4mapped

```
int enable_v4mapped
```

Specify whether UDPv6 transport will process IPv4 addresses.

Set this to 1 to turn on processing of IPv4 addresses. Note that this may make it incompatible with use of the UDPv4 transport within the same domain participant.

[default] 0.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.12 transport_priority_mask

```
long transport_priority_mask
```

Set mask for use of transport priority field.

If transport priority mapping is supported on the platform, this mask is used in conjunction with **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mapping_low** (p. 1436) and **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mapping_high** (p. 1437) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p. 274)) to the IPv6 TCLASS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv6 TCLASS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv6 TCLASS for send sockets.

[default] 0.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.13 transport_priority_mapping_low

```
int transport_priority_mapping_low
```

Set low value of output range to IPv6 TCLASS.

This is used in conjunction with **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mask** (p. 1436) and **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mapping_high** (p. 1437) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the low value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.14 transport_priority_mapping_high

```
int transport_priority_mapping_high
```

Set high value of output range to IPv6 TCLASS.

This is used in conjunction with **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mask** (p. 1436) and **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mapping_low** (p. 1436) to define the mapping from the DDS transport priority to the IPv6 TCLASS field. Defines the high value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0xff.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.229.3.15 port_offset

```
int port_offset
```

Port offset to allow multiple instances of UDPv6 transports in the same **com.rti.dds.domain.DomainParticipant** (p. 670).

By default, it is not possible to have multiple instances of the UDPv6 transport in the same **com.rti.dds.domain.↩ DomainParticipant** (p. 670). This is because all instances will try to bind to the same UDP port(s) for receiving data.

With this property, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. This way, you can have multiple instances of the UDPv6 transport in the same **com.rti.dds.domain.Domain↩ Participant** (p. 670).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

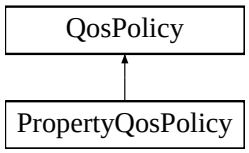
[default] 0

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.230 PropertyQosPolicy Class Reference

Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Inheritance diagram for PropertyQosPolicy:



Public Attributes

- final **PropertySeq value**

  Sequence of properties.

8.230.1 Detailed Description

Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connext that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670) **com.rti.dds.subscription.DataReader** (p. 450) **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = N/A;
Changeable (p. 256) = **YES** (p. 256)

See also

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.230.2 Usage

The PROPERTY QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553), or **com.rti.dds.domain.DomainParticipant** (p. 670). This is similar to the **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1959), except this policy uses (name, value) pairs, and you can select whether or not a particular pair should be propagated (included in the builtin topic).

You can find a complete list of predefined properties in the `Property Reference Guide`.

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 116)
- Durable Reader State, see **Configuring Durable Reader State** (p. 118)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1944), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1953), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1684)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 100)
- **Clock Selection** (p. 39)

In addition, you may add your own name/value pairs to the Property QoS policy of an **Entity** (p. 1029). Via this QoS policy, you can direct RTI Connext to propagate these name/value pairs with the discovery information for the **Entity** (p. 1029). Applications that discover the **Entity** (p. 1029) can then access the user-specific name/value pairs in the discovery information of the remote **Entity** (p. 1029). This allows you to add meta-information about an **Entity** (p. 1029) for application-specific use, for example, authentication/authorization certificates (which can also be done using the **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1959) or **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1127)).

8.230.2.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports
- Supports multiple instances of the builtin transports
- Allows full pluggable transport configuration for non-C/C++ language bindings (Java, .NET, etc.)
- Avoids the process of creating entities disabled, changing their QoS settings, then enabling them
- Allows selection of clock

Some of the RTI Connext capabilities configurable via the Property QoS policy can also be configured in code via APIs. However, the Property QoS policy allows you to configure those parameters via XML files. In addition, some of the configuration APIs will only work if the **Entity** (p. 1029) was created in a disabled state and then enabled after the configuration change was applied. By configuring those parameters using the Property QoS policy during entity creation, you avoid the additional work of first creating a disabled entity and then enabling it afterwards.

There are helper functions to facilitate working with properties, see the **PROPERTY** (p. 248) page.

8.230.3 Member Data Documentation

8.230.3.1 value

```
final PropertySeq value
```

Initial value:

```
=
        new PropertySeq()
```

Sequence of properties.

[default] An empty list.

Referenced by **PropertyQosPolicyHelper.add_property()**, **PropertyQosPolicyHelper.assert_property()**, **PropertyQosPolicyHelper.get_number_of_properties()**, **PropertyQosPolicyHelper.get_properties()**, **Property↩ QosPolicyHelper.get_qos_resource_limits_property_string_max_length()**, **PropertyQosPolicyHelper.lookup↩ _property()**, and **PropertyQosPolicyHelper.remove_property()**.

8.231 PropertyQosPolicyHelper Class Reference

Policy helpers that facilitate management of the properties in the input policy.

Static Public Member Functions

- static int **get_number_of_properties** (**PropertyQosPolicy** policy)

  Gets the number of properties in the input policy.
- static void **assert_property** (**PropertyQosPolicy** policy, String name, String value, boolean propagate)

  Asserts the property identified by name in the input policy.
- static void **add_property** (**PropertyQosPolicy** policy, String name, String value, boolean propagate)

  Adds a new property to the input policy.
- static **Property_t lookup_property** (**PropertyQosPolicy** policy, String name)

  Searches for a property in the input policy given its name.
- static void **remove_property** (**PropertyQosPolicy** policy, String name)

  Removes a property from the input policy.
- static void **get_properties** (**PropertyQosPolicy** policy, **PropertySeq** properties, String name_prefix)

  Retrieves a list of properties whose names match the input prefix.
- static void **configure_pki_secure_transport_properties** (**PropertyQosPolicy** policy, String transport_↩ plugin_prefix, java.security.cert.Certificate[ ] root_ca_certificates, java.security.cert.Certificate[ ] certificate_chain, java.security.PrivateKey private_key)

  Configures the public-key infrastructure elements required by a secure transport specified by its prefix.
- static int **get_qos_resource_limits_property_string_max_length** (**PropertyQosPolicy** policy)

  Returns the maximum cumulative string length of all the properties within the specified policy.
- static **PropertyQosPolicyMutability get_property_mutability** (String name, **PropertyQosPolicy** policy)

  Returns the mutability type of a property.

8.231.1 Detailed Description

Policy helpers that facilitate management of the properties in the input policy.

8.231.2 Member Function Documentation

8.231.2.1 get_number_of_properties()

```
static int get_number_of_properties (
              PropertyQosPolicy policy )  [static]
```

Gets the number of properties in the input policy.

Precondition

policy cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.

Returns

Number of properties.

References **PropertyQosPolicy.value**.

8.231.2.2 assert_property()

```
static void assert_property (
              PropertyQosPolicy policy,
             String name,
             String value,
             boolean propagate )  [static]
```

Asserts the property identified by name in the input policy.

If the property already exists, this function replaces its current value with the new one.

If the property identified by name does not exist, this function adds it to the property set.

This function increases the maximum number of elements of the policy sequence when this number is not enough to store the new property.

Precondition

policy, name and value cannot be null.

Parameters

policy	<<***in***>> *(p. 156)* Input policy.
name	<<***in***>> *(p. 156)* Property name.
value	<<***in***>> *(p. 156)* Property value.
propagate	<<***in***>> *(p. 156)* Indicates if the property will be propagated on discovery.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

References **PropertyQosPolicyHelper.lookup_property()**, and **PropertyQosPolicy.value**.

Referenced by **PropertyQosPolicyHelper.configure_pki_secure_transport_properties()**.

8.231.2.3 add_property()

```
static void add_property (
              PropertyQosPolicy policy,
             String name,
             String value,
             boolean propagate )  [static]
```

Adds a new property to the input policy.

This function will allocate memory to store the (name, value) pair. The memory allocated is owned by RTI Connext.

If the maximum number of elements of the policy sequence is not enough to store the new property, this function will increase it.

If the property already exists the function fails with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_↩ MET** (p. 1598).

Precondition

policy, name and value cannot be null.

The property is not in the policy.

Parameters

policy	<<***in***>> *(p. 156)* Input policy.
name	<<***in***>> *(p. 156)* Property name.
value	<<***in***>> *(p. 156)* Property value.
propagate	<<***in***>> *(p. 156)* Indicates if the property will be propagated on discovery.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598)

References **PropertyQosPolicyHelper.lookup_property()**, and **PropertyQosPolicy.value**.

8.231.2.4 lookup_property()

```
static  Property_t lookup_property (
              PropertyQosPolicy policy,
             String name )  [static]
```

Searches for a property in the input policy given its name.

Precondition

policy, name and value cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* Property name.

Returns

The function returns the first property with the given name. If such a property does not exist, the function returns NULL.

References **Property_t.name**, and **PropertyQosPolicy.value**.

Referenced by **PropertyQosPolicyHelper.add_property()**, **PropertyQosPolicyHelper.assert_property()**, and **PropertyQosPolicyHelper.remove_property()**.

8.231.2.5 remove_property()

```
static void remove_property (
              PropertyQosPolicy policy,
             String name )  [static]
```

Removes a property from the input policy.

If the property does not exist, the function fails with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_↩ MET** (p. 1598).

Precondition

policy and name cannot be null.

The property is in the policy.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
name	<<***in***>> *(*p. *156)* Property name.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

References **PropertyQosPolicyHelper.lookup_property()**, and **PropertyQosPolicy.value**.

8.231.2.6 get_properties()

```
static void get_properties (
              PropertyQosPolicy policy,
              PropertySeq properties,
             String name_prefix )  [static]
```

Retrieves a list of properties whose names match the input prefix.

If the properties sequence doesn't own its buffer, and its maximum is less than the total number of properties matching the input prefix, it will be filled up to its maximum and fail with an error of **com.rti.dds.infrastructure.RETCODE_↩ OUT_OF_RESOURCES** (p. 1598).

Precondition

policy, properties and name_prefix cannot be null.

Parameters

policy	<<***in***>> *(*p. *156)* Input policy.
properties	<<***inout***>> *(*p. *156)* A **com.rti.dds.infrastructure.PropertySeq** (p. 1448) object where the set or list of properties will be returned.
name_prefix	Name prefix.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

References **Property_t.name**, **Property_t.propagate**, **Property_t.value**, and **PropertyQosPolicy.value**.

8.231.2.7 configure_pki_secure_transport_properties()

```
static void configure_pki_secure_transport_properties (
              PropertyQosPolicy policy,
             String transport_plugin_prefix,
             java.security.cert.Certificate[] root_ca_certificates,
             java.security.cert.Certificate[] certificate_chain,
             java.security.PrivateKey private_key )  [static]
```

Configures the public-key infrastructure elements required by a secure transport specified by its prefix.

This operation asserts the necessary set of properties that specify public-key elements required by a secure transport. These elements are the root CA certificates, the identifying certificate chain and the private key.

The operation sets the corresponding properties for each of these public-key elements, replacing an existing property if present in the policy. The set of properties contain both certificates and private key in PEM format.

These properties are:

- root_ca_certificates: <transport_plugin_prefix>.tls.verify.ca
- certificate_chain: <transport_plugin_prefix>.tls.identity.certificate_chain
- private_key <transport_plugin_prefix>.tls.identity.private_ky

The properties are asserted with the propagate option set to false;

The supported secure transport plugin is: RTI Secure TCP Transport (TCP + TLS). Library: nddstransporttcp

Parameters

policy	<<***inout***>> *(p. 156)* Policy to be populated.
transport_plugin_prefix	<<***in***>> *(p. 156)* Plugin prefix that is given to the transport plugin when the plugin is loaded. This is done by setting the property 'dds.transport.load_plugins'.
root_ca_certificates	<<***in***>> *(p. 156)* List of CA certificates that will be used to verify the validity of the identifying certificate chain. At least one root CA certificate is required.
certificate_chain	<<***in***>> *(p. 156)* Identifying certificate chain. At least one identifying certificate is required.
private_key	<<***in***>> *(p. 156)* Private key associated with the public key of the identifying certificates and that is required for the authentication process. It must be unencrypted.

References **PropertyQosPolicyHelper.assert_property()**.

8.231.2.8 get_qos_resource_limits_property_string_max_length()

```
static int get_qos_resource_limits_property_string_max_length (
              PropertyQosPolicy policy )  [static]
```

Returns the maximum cumulative string length of all the properties within the specified policy.

This operation computes the total cumulative length of the properties. This represents the minimum value that must be set in the resource limits qos so that it is consistent.

References **PropertyQosPolicy.value**.

8.231.2.9 get_property_mutability()

```
static  PropertyQosPolicyMutability get_property_mutability (
             String name,
              PropertyQosPolicy policy )  [static]
```

Returns the mutability type of a property.

If the property does not exist, **com.rti.dds.infrastructure.PropertyQosPolicyMutability.PROPERTY_QOS_↩ MUTABLE** (p. 1447) is returned. Otherwise, it returns the mutability type of the specified property. See **com.↩ rti.dds.infrastructure.PropertyQosPolicyMutability** (p. 1446) for more information of the different mutability types.

Parameters

name	<<***in***>> *(p. 156)*. The name of the property for which we want to know the mutability type.
policy	<<***in***>> *(p. 156)*. These are the properties in which we are going to look for the property with the provided name.

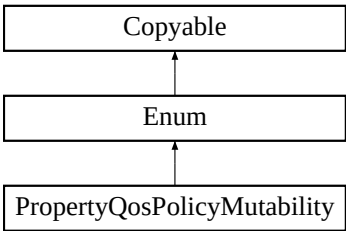
Returns

The mutability type of the input property.

8.232 PropertyQosPolicyMutability Class Reference

Determines if, and when, the value of a property can change.

Inheritance diagram for PropertyQosPolicyMutability:



Static Public Attributes

- static final **PropertyQosPolicyMutability PROPERTY_QOS_MUTABLE**

  The property is mutable: it can be changed at any time.
- static final **PropertyQosPolicyMutability PROPERTY_QOS_MUTABLE_UNTIL_ENABLE**

  The property can only be changed before enabling the entity.
- static final **PropertyQosPolicyMutability PROPERTY_QOS_IMMUTABLE**

  The property is immutable: it can only be specified when creating the entity.

Additional Inherited Members

8.232.1 Detailed Description

Determines if, and when, the value of a property can change.

8.232.2 Member Data Documentation

8.232.2.1 PROPERTY_QOS_MUTABLE

```
final PropertyQosPolicyMutability PROPERTY_QOS_MUTABLE [static]
```

The property is mutable: it can be changed at any time.

8.232.2.2 PROPERTY_QOS_MUTABLE_UNTIL_ENABLE

```
final PropertyQosPolicyMutability PROPERTY_QOS_MUTABLE_UNTIL_ENABLE [static]
```

The property can only be changed before enabling the entity.

8.232.2.3 PROPERTY_QOS_IMMUTABLE

```
final PropertyQosPolicyMutability PROPERTY_QOS_IMMUTABLE [static]
```

The property is immutable: it can only be specified when creating the entity.

8.233 PropertySeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.Property_t** (p. 1395) >

Inherits ArraySequence.

8.233.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.Property_t** (p. 1395) >

See also

com.rti.dds.infrastructure.Property_t (p. 1395)

8.234 ProtocolVersion_t Class Reference

<<***extension***>> *(p. 155)* Type used to represent the version of the RTPS protocol.

Inherits Struct.

Public Member Functions

- **ProtocolVersion_t** ()
  Constructor.

Public Attributes

- byte **major**
  Major protocol version number.
- byte **minor**
  Minor protocol version number.

Static Public Attributes

- static final **ProtocolVersion_t PROTOCOLVERSION_1_0**
  The protocol version 1.0.
- static final **ProtocolVersion_t PROTOCOLVERSION_1_1**
  The protocol version 1.1.
- static final **ProtocolVersion_t PROTOCOLVERSION_1_2**
  The protocol version 1.2.
- static final **ProtocolVersion_t PROTOCOLVERSION_2_0**
  The protocol version 2.0.
- static final **ProtocolVersion_t PROTOCOLVERSION_2_1**
  The protocol version 2.1.
- static final **ProtocolVersion_t PROTOCOLVERSION**
  The most recent protocol version. Currently 2.1.

8.234.1 Detailed Description

<<***extension***>> *(*p. *155)* Type used to represent the version of the RTPS protocol.

8.234.2 Constructor & Destructor Documentation

8.234.2.1 ProtocolVersion_t()

```
ProtocolVersion_t ( )
```

Constructor.

8.234.3 Member Data Documentation

8.234.3.1 PROTOCOLVERSION_1_0

```
final  ProtocolVersion_t PROTOCOLVERSION_1_0  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)1, (byte)0)
```

The protocol version 1.0.

8.234.3.2 PROTOCOLVERSION_1_1

```
final  ProtocolVersion_t PROTOCOLVERSION_1_1  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)1, (byte)1)
```

The protocol version 1.1.

8.234.3.3 PROTOCOLVERSION_1_2

```
final  ProtocolVersion_t PROTOCOLVERSION_1_2  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)1, (byte)2)
```

The protocol version 1.2.

8.234.3.4 PROTOCOLVERSION_2_0

```
final  ProtocolVersion_t PROTOCOLVERSION_2_0  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)2, (byte)0)
```

The protocol version 2.0.

8.234.3.5 PROTOCOLVERSION_2_1

```
final  ProtocolVersion_t PROTOCOLVERSION_2_1  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)2, (byte)1)
```

The protocol version 2.1.

8.234.3.6 PROTOCOLVERSION

```
final  ProtocolVersion_t PROTOCOLVERSION  [static]
```

Initial value:

```
=
    new ProtocolVersion_t((byte)2, (byte)1)
```

The most recent protocol version. Currently 2.1.

8.234.3.7 major

```
byte major
```

Major protocol version number.

8.234.3.8 minor

```
byte minor
```

Minor protocol version number.

8.235 PUBLIC_MEMBER Class Reference

Constant used to indicate that a value type member is public.

Static Public Attributes

- static final short **VALUE**

8.235.1 Detailed Description

Constant used to indicate that a value type member is public.

See also

com.rti.dds.typecode.Visibility.PRIVATE_MEMBER

8.235.2 Member Data Documentation

8.235.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

Referenced by **TypeCode.add_member()**, and **TypeCode.member_visibility()**.

8.236 PublicationBuiltinTopicData Class Reference

Entry created when a **com.rti.dds.publication.DataWriter** (p. 553) is discovered in association with its **Publisher** (p. 1466).

Inherits AbstractBuiltinTopicData.

Public Attributes

- final **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- final **BuiltinTopicKey_t participant_key**

  *DCPS key of the participant to which the **DataWriter** (p. 553) belongs.*
- String **topic_name**

  *Name of the related **com.rti.dds.topic.Topic** (p. 1807).*
- String **type_name**

  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).*
- final **DurabilityQosPolicy durability**

  *durability policy of the corresponding **DataWriter** (p. 553)*
- final **DurabilityServiceQosPolicy durability_service**

  *durability_service policy of the corresponding **DataWriter** (p. 553)*
- final **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **LifespanQosPolicy lifespan**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **UserDataQosPolicy user_data**

*Policy of the corresponding **DataWriter** (p. 553).*

- final **OwnershipQosPolicy ownership**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **OwnershipStrengthQosPolicy ownership_strength**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **DestinationOrderQosPolicy destination_order**

  *Policy of the corresponding **DataWriter** (p. 553).*
- final **PresentationQosPolicy presentation**

  *Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.*
- final **PartitionQosPolicy partition**

  *Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.*
- final **TopicDataQosPolicy topic_data**

  Policy of the related Topic.
- final **GroupDataQosPolicy group_data**

  *Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.*
- **TypeCode type_code**

  *<<**extension**>> (p. 155) Type code information of the corresponding Topic*
- final **BuiltinTopicKey_t publisher_key**

  *<<**extension**>> (p. 155) DCPS key of the publisher to which the **DataWriter** (p. 553) belongs*
- final **PropertyQosPolicy property**

  *<<**extension**>> (p. 155) Properties of the corresponding **DataWriter** (p. 553).*
- final **LocatorSeq unicast_locators**

  *<<**extension**>> (p. 155) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- final **GUID_t virtual_guid**

  *<<**extension**>> (p. 155) Virtual GUID associated to the **DataWriter** (p. 553).*
- final **ServiceQosPolicy service**

  *<<**extension**>> (p. 155) Policy of the corresponding **DataWriter** (p. 553).*
- final **ProtocolVersion_t rtps_protocol_version**

  *<<**extension**>> (p. 155) Version number of the RTPS wire protocol used.*
- final **VendorId_t rtps_vendor_id**

  *<<**extension**>> (p. 155) ID of vendor implementing the RTPS wire protocol.*
- final **ProductVersion_t product_version**

  *<<**extension**>> (p. 155) This is a vendor specific parameter. It gives the current version for rti-dds.*
- final **DataRepresentationQosPolicy representation**

  *Data representation policy of the corresponding **DataWriter** (p. 553).*
- final **LocatorFilterQosPolicy locator_filter**

  *<<**extension**>> (p. 155) Policy of the corresponding **DataWriter** (p. 553)*
- boolean **disable_positive_acks**

  *<<**extension**>> (p. 155) This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.*
- final **EntityNameQosPolicy publication_name**

  *<<**extension**>> (p. 155) The publication name and role name.*
- final **EndpointTrustProtectionInfo trust_protection_info**

  *<<**extension**>> (p. 155) Trust Plugins protection information associated with the discovered **DataWriter** (p. 553).*
- final **EndpointTrustAlgorithmInfo trust_algorithm_info**

  *<<**extension**>> (p. 155) Trust Plugins algorithms associated with the discovered **DataWriter** (p. 553).*
- final **DataTagQosPolicy data_tags**

  *Tags of the corresponding **DataWriter** (p. 553).*

8.236.1 Detailed Description

Entry created when a **com.rti.dds.publication.DataWriter** (p. 553) is discovered in association with its **Publisher** (p. 1466).

Data associated with the built-in topic **com.rti.dds.publication.builtin.PublicationBuiltinTopicDataTypeSupport.↩ PUBLICATION_TOPIC_NAME** (p. 163). It contains QoS policies and additional information that apply to the remote **com.rti.dds.publication.DataWriter** (p. 553) the related **com.rti.dds.publication.Publisher** (p. 1466).

See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME (p. 163)

builtin.PublicationBuiltinTopicDataDataReader (p. 1461)

8.236.2 Member Data Documentation

8.236.2.1 key

```
final BuiltinTopicKey_t key
```

DCPS key to distinguish entries.

8.236.2.2 participant_key

```
final BuiltinTopicKey_t participant_key
```

DCPS key of the participant to which the **DataWriter** (p. 553) belongs.

8.236.2.3 topic_name

```
String topic_name
```

Name of the related **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.236.2.4 type_name

```
String type_name
```

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.236.2.5 durability

```
final DurabilityQosPolicy durability
```

durability policy of the corresponding **DataWriter** (p. 553)

8.236.2.6 durability_service

```
final DurabilityServiceQosPolicy durability_service
```

durability_service policy of the corresponding **DataWriter** (p. 553)

8.236.2.7 deadline

```
final DeadlineQosPolicy deadline
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.8 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.9 liveliness

```
final LivelinessQosPolicy liveliness
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.10 reliability

```
final ReliabilityQosPolicy reliability
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.11 lifespan

```
final LifespanQosPolicy lifespan
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.12 user_data

```
final UserDataQosPolicy user_data
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.13 ownership

```
final OwnershipQosPolicy ownership
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.14 ownership_strength

```
final OwnershipStrengthQosPolicy ownership_strength
```

Policy of the corresponding **DataWriter** (p. 553).

8.236.2.15 destination_order

```
final DestinationOrderQosPolicy destination_order
```

Policy of the corresponding **DataWriter** (p. 553).

Warning

Only the field **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is propagated during discovery. The other fields always contain their default values.

8.236.2.16 presentation

```
final PresentationQosPolicy presentation
```

Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.

8.236.2.17 partition

```
final PartitionQosPolicy partition
```

Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.

8.236.2.18 topic_data

```
final TopicDataQosPolicy topic_data
```

Policy of the related Topic.

8.236.2.19 group_data

```
final GroupDataQosPolicy group_data
```

Policy of the **Publisher** (p. 1466) to which the **DataWriter** (p. 553) belongs.

8.236.2.20 type_code

```
TypeCode type_code
```

<<***extension***>> *(*p. *155)* Type code information of the corresponding Topic

8.236.2.21 publisher_key

```
final BuiltinTopicKey_t publisher_key
```

<<***extension***>> *(*p. *155)* DCPS key of the publisher to which the **DataWriter** (p. 553) belongs

8.236.2.22 property

```
final PropertyQosPolicy property
```

<<***extension***>> *(*p. *155)* Properties of the corresponding **DataWriter** (p. 553).

8.236.2.23 unicast_locators

```
final LocatorSeq unicast_locators
```

<<***extension***>> *(*p. *155)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.236.2.24 virtual_guid

```
final GUID_t virtual_guid
```

<<***extension***>> *(*p. *155)* Virtual GUID associated to the **DataWriter** (p. 553).

See also

com.rti.dds.infrastructure.GUID_t (p. 1132)

8.236.2.25 service

```
final  ServiceQosPolicy service
```

<<***extension***>> *(p. 155)* Policy of the corresponding **DataWriter** (p. 553).

8.236.2.26 rtps_protocol_version

```
final  ProtocolVersion_t rtps_protocol_version
```

<<***extension***>> *(p. 155)* Version number of the RTPS wire protocol used.

8.236.2.27 rtps_vendor_id

```
final  VendorId_t rtps_vendor_id
```

<<***extension***>> *(p. 155)* ID of vendor implementing the RTPS wire protocol.

8.236.2.28 product_version

```
final  ProductVersion_t product_version
```

<<***extension***>> *(p. 155)* This is a vendor specific parameter. It gives the current version for rti-dds.

8.236.2.29 representation

```
final  DataRepresentationQosPolicy representation
```

Data representation policy of the corresponding **DataWriter** (p. 553).

8.236.2.30 locator_filter

```
final  LocatorFilterQosPolicy locator_filter
```

<<***extension***>> *(p. 155)* Policy of the corresponding **DataWriter** (p. 553)

Related to **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1318).

8.236.2.31 disable_positive_acks

```
boolean disable_positive_acks
```

<<***extension***>> *(p. 155)* This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.

8.236.2.32 publication_name

```
final  EntityNameQosPolicy publication_name
```

<<***extension***>> *(p. 155)* The publication name and role name.

This member contains the name and the role name of the discovered publication.

8.236.2.33 trust_protection_info

```
final  EndpointTrustProtectionInfo trust_protection_info
```

<<***extension***>> *(p. 155)* Trust Plugins protection information associated with the discovered **DataWriter** (p. 553).

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.236.2.34 trust_algorithm_info

```
final  EndpointTrustAlgorithmInfo trust_algorithm_info
```

<<***extension***>> *(p. 155)* Trust Plugins algorithms associated with the discovered **DataWriter** (p. 553).

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.236.2.35 data_tags

```
final DataTagQosPolicy data_tags
```

Tags of the corresponding **DataWriter** (p. 553).

8.237 PublicationBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) > .

Inherits DataReaderImpl.

8.237.1 Detailed Description

Instantiates `DataReader` < **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) > .

com.rti.dds.subscription.DataReader (p. 450) of topic **com.rti.dds.publication.builtin.PublicationBuiltinTopic↩ DataTypeSupport.PUBLICATION_TOPIC_NAME** (p. 163) used for accessing **com.rti.dds.publication.builtin.↩ PublicationBuiltinTopicData** (p. 1452) of the remote **com.rti.dds.publication.DataWriter** (p. 553) and the associated **com.rti.dds.publication.Publisher** (p. 1466).

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooDataReader** (p. 1067)

See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

com.rti.dds.publication.builtin.PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME (p. 163)

8.238 PublicationBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.publication.↩ builtin.PublicationBuiltinTopicData** (p. 1452) > .

Inherits AbstractBuiltinTopicDataSeq.

8.238.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

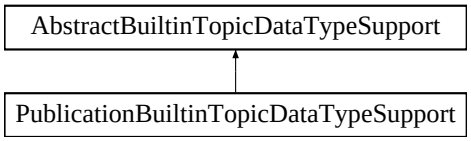
See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

8.239 PublicationBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) > .

Inheritance diagram for PublicationBuiltinTopicDataTypeSupport:



Static Public Attributes

- static final String **PUBLICATION_TOPIC_NAME** = DDS_PUBLICATION_TOPIC_NAME()
  Publication topic name.

Additional Inherited Members

8.239.1 Detailed Description

Instantiates `TypeSupport` < **com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) > .

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooTypeSupport** (p. 1118)

See also

com.rti.dds.publication.builtin.PublicationBuiltinTopicData (p. 1452)

8.240 PublicationMatchedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS

Inherits Status.

Public Attributes

- int **total_count** = 0

  *The total cumulative number of times that this **com.rti.dds.publication.DataWriter** (p. 553) discovered a "match" with a **com.rti.dds.subscription.DataReader** (p. 450).*
- int **total_count_change** = 0

  The changes in total_count since the last time the listener was called or the status was read.
- int **current_count** = 0

  *The current number of DataReaders with which this **com.rti.dds.publication.DataWriter** (p. 553) is matched.*
- int **current_count_peak** = 0

  *<<**extension**>> (p. 155) Greatest number of DataReaders that matched this **com.rti.dds.publication.DataWriter** (p. 553) simultaneously.*
- int **current_count_change** = 0

  The change in current_count since the last time the listener was called or the status was read.
- final **InstanceHandle_t** **last_subscription_handle** = new **InstanceHandle_t**()

  *This InstanceHandle can be used to look up which remote **com.rti.dds.subscription.DataReader** (p. 450) was the last to cause this **DataWriter** (p. 553)'s status to change, using **com.rti.dds.publication.DataWriter.get_matched_↩ subscription_data** (p. 568).*

8.240.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS

A "match" happens when the **com.rti.dds.publication.DataWriter** (p. 553) finds a **com.rti.dds.subscription.Data↩ Reader** (p. 450) with the same **com.rti.dds.topic.Topic** (p. 1807), same or compatible data type, and requested QoS that is compatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 553). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **com.rti.dds.publication.↩ DataWriter** (p. 553) will become "unmatched" from a remote **com.rti.dds.subscription.DataReader** (p. 450) when that **com.rti.dds.subscription.DataReader** (p. 450) goes away for any of the following reasons:

- The matched **com.rti.dds.subscription.DataReader** (p. 450)'s **com.rti.dds.domain.DomainParticipant** (p. 670) has lost liveliness.
- This **DataWriter** (p. 553) or the matched DataReader has changed QoS such that the entities are now incompatible.
- The matched DataReader has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **com.rti.dds.publication.Publication↩ MatchedStatus.current_count_change** (p. 1464) can be used to determine the number of changes since the listener was called back or the status was checked.

8.240.2 Member Data Documentation

8.240.2.1 total_count

```
int total_count = 0
```

The total cumulative number of times that this **com.rti.dds.publication.DataWriter** (p. 553) discovered a "match" with a **com.rti.dds.subscription.DataReader** (p. 450).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463).

8.240.2.2 total_count_change

```
int total_count_change = 0
```

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times this **DataWriter** (p. 553) ever matched with a DataReader).

8.240.2.3 current_count

```
int current_count = 0
```

The current number of DataReaders with which this **com.rti.dds.publication.DataWriter** (p. 553) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463).

8.240.2.4 current_count_peak

```
int current_count_peak = 0
```

<<***extension***>> *(p. 155)* Greatest number of DataReaders that matched this **com.rti.dds.publication.DataWriter** (p. 553) simultaneously.

That is, there was no moment in time when more than this many DataReaders matched this **DataWriter** (p. 553). (As a result, total_count can be higher than current_count_peak.)

8.240.2.5 current_count_change

```
int current_count_change = 0
```

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataReaders have become unmatched for one or more of the reasons described in **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463).

8.240.2.6 last_subscription_handle

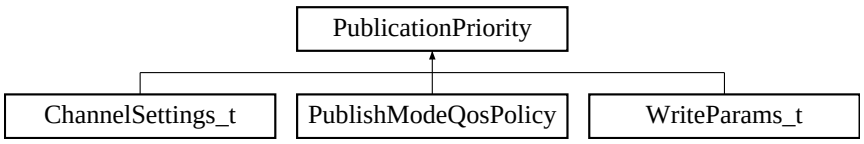
```
final InstanceHandle_t last_subscription_handle = new InstanceHandle_t()
```

This InstanceHandle can be used to look up which remote **com.rti.dds.subscription.DataReader** (p. 450) was the last to cause this **DataWriter** (p. 553)'s status to change, using **com.rti.dds.publication.DataWriter.get_matched_subscription_data** (p. 568).

If the DataReader no longer matches this **DataWriter** (p. 553) due to any of the reasons in **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463) except incompatible QoS, then the DataReader has been purged from this **DataWriter** (p. 553)'s DomainParticipant discovery database. (See the "Discovery Overview" section of the `User's Manual`.) In that case, the **com.rti.dds.publication.DataWriter.get_matched_subscription_data** (p. 568) method will not be able to return information about the DataReader. The only way to get information about the lost DataReader is if you cached the information previously.

8.241 PublicationPriority Interface Reference

Inheritance diagram for PublicationPriority:



Static Public Attributes

- static final int **UNDEFINED**

  *Initializer value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).*

- static final int **AUTOMATIC**

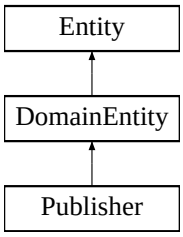
  *Constant value for **com.rti.dds.infrastructure.PublishModeQosPolicy.priority** (p. 1498) and/or **com.rti.dds.infrastructure.ChannelSettings_t.priority** (p. 414).*

8.241.1 Detailed Description

8.242 Publisher Interface Reference

<<***interface***>> *(*p. *156)* A publisher is the object responsible for the actual dissemination of publications.

Inheritance diagram for Publisher:



Public Member Functions

- void **get_default_datawriter_qos** (**DataWriterQos** qos)

  Copies the default ***com.rti.dds.publication.DataWriterQos*** *(*p. *612) values into the provided* ***com.rti.dds.publication.DataWriterQos*** *(*p. *612) instance.*
- void **set_default_datawriter_qos** (**DataWriterQos** qos)

  Sets the default ***com.rti.dds.publication.DataWriterQos*** *(*p. *612) values for this publisher.*
- void **set_default_datawriter_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Set the default* ***com.rti.dds.publication.DataWriterQos*** *(*p. *612) values for this publisher based on the input XML QoS profile.*
- **DataWriter** **create_datawriter** (**Topic** topic, **DataWriterQos** qos, **DataWriterListener** listener, int mask)

  Creates a ***com.rti.dds.publication.DataWriter*** *(*p. *553) that will be attached and belong to the* ***com.rti.dds.publication.Publisher*** *(*p. *1466).*
- **DataWriter** **create_datawriter_with_profile** (**Topic** topic, String library_name, String profile_name, **DataWriterListener** listener, int mask)

  <<***extension***>> *(*p. *155) Creates a* ***com.rti.dds.publication.DataWriter*** *(*p. *553) object using the* ***com.rti.dds.publication.DataWriterQos*** *(*p. *612) associated with the input XML QoS profile.*
- void **delete_datawriter** (**DataWriter** a_datawriter)

  Deletes a ***com.rti.dds.publication.DataWriter*** *(*p. *553) that belongs to the* ***com.rti.dds.publication.Publisher*** *(*p. *1466).*
- **DataWriter** **lookup_datawriter** (String topic_name)

  Retrieves the ***com.rti.dds.publication.DataWriter*** *(*p. *553) for a specific* ***com.rti.dds.topic.Topic*** *(*p. *1807).*
- void **set_qos** (**PublisherQos** qos)

  Sets the publisher QoS.
- void **set_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Change the QoS of this publisher using the input XML QoS profile.*
- void **get_qos** (**PublisherQos** qos)

  Gets the publisher QoS.
- String **get_default_library** ()

  <<***extension***>> *(*p. *155) Gets the default XML library associated with a* ***com.rti.dds.publication.Publisher*** *(*p. *1466).*
- void **set_default_library** (String library_name)

*<<**extension**>> (p. 155) Sets the default XML library for a **com.rti.dds.publication.Publisher** (p. 1466).*

- String **get_default_profile** ()

  *<<**extension**>> (p. 155) Gets the default XML profile associated with a **com.rti.dds.publication.Publisher** (p. 1466).*

- void **set_default_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Sets the default XML profile for a **com.rti.dds.publication.Publisher** (p. 1466).*

- String **get_default_profile_library** ()

  *<<**extension**>> (p. 155) Gets the library where the default XML QoS profile is contained for a **com.rti.dds.publication.Publisher** (p. 1466).*

- void **set_listener** (**PublisherListener** l, int mask)

  Sets the publisher listener.

- **PublisherListener** **get_listener** ()

  Get the publisher listener.

- void **suspend_publications** ()

  *Indicates to RTI Connext that the application is about to make multiple modifications using **com.rti.dds.publication.DataWriter** (p. 553) objects belonging to the **com.rti.dds.publication.Publisher** (p. 1466).*

- void **resume_publications** ()

  *Indicates to RTI Connext that the application has completed the multiple changes initiated by the previous **com.rti.dds.publication.Publisher.suspend_publications** (p. 1481).*

- void **begin_coherent_changes** ()

  *Indicates that the application will begin a coherent set of modifications using **com.rti.dds.publication.DataWriter** (p. 553) objects attached to the **com.rti.dds.publication.Publisher** (p. 1466).*

- void **end_coherent_changes** ()

  *Terminates the coherent set initiated by the matching call to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1483).*

- void **copy_from_topic_qos** (**DataWriterQos** a_datawriter_qos, **TopicQos** a_topic_qos)

  *Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.dds.publication.DataWriterQos** (p. 612).*

- void **get_all_datawriters** (DataWriterSeq writers)

  *Retrieve all the DataWriters created from this **Publisher** (p. 1466).*

- **DomainParticipant** **get_participant** ()

  *Returns the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.publication.Publisher** (p. 1466) belongs.*

- void **delete_contained_entities** ()

  *Deletes all the entities that were created by means of the "create" operation on the **com.rti.dds.publication.Publisher** (p. 1466).*

- void **wait_for_acknowledgments** (**Duration_t** max_wait)

  *Blocks the calling thread until all data written by the **Publisher** (p. 1466)'s reliable DataWriters is acknowledged, or until timeout expires.*

- void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

  *<<**extension**>> (p. 155) Blocks the calling thread until asynchronous sending is complete.*

- **DataWriter** **lookup_datawriter_by_name** (String datawriter_name)

  *<<**extension**>> (p. 155) Retrieves a **com.rti.dds.publication.DataWriter** (p. 553) contained within the **com.rti.dds.publication.Publisher** (p. 1466) the **com.rti.dds.publication.DataWriter** (p. 553) entity name.*

Static Public Attributes

- static final **DataWriterQos DATAWRITER_QOS_DEFAULT**

  *Special value for creating **com.rti.dds.publication.DataWriter** (p. 553) with default QoS.*
- static final **DataWriterQos DATAWRITER_QOS_USE_TOPIC_QOS** = new **DataWriterQos**()

  *Special value for creating **com.rti.dds.publication.DataWriter** (p. 553) with a combination of the default **com.rti.dds.publication.DataWriterQos** (p. 612) and the **com.rti.dds.topic.TopicQos** (p. 1824).*
- static final **DataWriterQos DATAWRITER_QOS_PRINT_ALL** = new **DataWriterQos**()

  *Special value which can be supplied to **com.rti.dds.publication.DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)** (p. 615) indicating that all of the QoS should be printed.*

8.242.1 Detailed Description

<<***interface***>> *(p. 156)* A publisher is the object responsible for the actual dissemination of publications.

QoS:

com.rti.dds.publication.PublisherQos (p. 1490)

Listener:

com.rti.dds.publication.PublisherListener (p. 1488)

A publisher acts on the behalf of one or several **com.rti.dds.publication.DataWriter** (p. 553) objects that belong to it. When it is informed of a change to the data associated with one of its **com.rti.dds.publication.DataWriter** (p. 553) objects, it decides when it is appropriate to actually send the data-update message. In making this decision, it considers any extra information that goes with the data (timestamp, writer, etc.) as well as the QoS of the **com.rti.dds.publication.Publisher** (p. 1466) and the **com.rti.dds.publication.DataWriter** (p. 553).

The following operations may be called even if the **com.rti.dds.publication.Publisher** (p. 1466) is not enabled. Other operations will fail with the value **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) if called on a disabled **com.rti.dds.publication.Publisher** (p. 1466):

- **com.rti.dds.infrastructure.Entity.enable** (p. 1032),
- **com.rti.dds.publication.Publisher.set_qos** (p. 1476), **com.rti.dds.publication.Publisher.get_qos** (p. 1477) , **com.rti.dds.publication.Publisher.set_qos_with_profile** (p. 1476)
- **com.rti.dds.publication.Publisher.set_listener** (p. 1480), **com.rti.dds.publication.Publisher.get_listener** (p. 1481),
- **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034)
- **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471), **com.rti.dds.publication.Publisher.create_datawriter_with_profile** (p. 1473), **com.rti.dds.publication.Publisher.delete_contained_entities** (p. 1485), **com.rti.dds.publication.Publisher.delete_datawriter** (p. 1474),
- **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469), **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1470), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1469), **com.rti.dds.publication.Publisher.wait_for_acknowledgments** (p. 1486), **com.rti.dds.publication.Publisher.set_default_library** (p. 1478), **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479),

See also

Operations Allowed in Listener Callbacks (p. 1238)

8.242.2 Member Function Documentation

8.242.2.1 get_default_datawriter_qos()

```
void get_default_datawriter_qos (
            DataWriterQos qos )
```

Copies the default **com.rti.dds.publication.DataWriterQos** (p. 612) values into the provided **com.rti.dds.↩ publication.DataWriterQos** (p. 612) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.publication.↩ Publisher.set_default_datawriter_qos** (p. 1469) or **com.rti.dds.publication.Publisher.set_default_datawriter_↩ qos_with_profile** (p. 1470), or else, if the call was never made, the default values from its owning **com.rti.dds.↩ domain.DomainParticipant** (p. 670).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a **com.rti.dds.publication.Publisher** (p. 1466) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469).

Parameters

qos	<<***inout***>> *(p. 156)* **com.rti.dds.publication.DataWriterQos** (p. 612) to be filled-up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT (p. 71)

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

8.242.2.2 set_default_datawriter_qos()

```
void set_default_datawriter_qos (
            DataWriterQos qos )
```

Sets the default **com.rti.dds.publication.DataWriterQos** (p. 612) values for this publisher.

This call causes the default values inherited from the owning **com.rti.dds.domain.DomainParticipant** (p. 670) to be overridden.

This default value will be used for newly created **com.rti.dds.publication.DataWriter** (p. 553) if **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) is specified as the `qos` parameter when **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.publication.Publisher** (p. 1466) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1469) or calling **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471) with **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 156)* Default qos to be set. The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.242.2.3 set_default_datawriter_qos_with_profile()

```
void set_default_datawriter_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.publication.DataWriterQos** (p. 612) values for this publisher based on the input XML QoS profile.

This default value will be used for newly created **com.rti.dds.publication.DataWriter** (p. 553) if **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) is specified as the `qos` parameter when **com.rti.dds.publication.Publisher.create_datawriter** (p. 1471) is called.

Precondition

The **com.rti.dds.publication.DataWriterQos** (p. 612) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.publication.Publisher** (p. 1466) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1469) or calling **com.rti.dds.↩ publication.Publisher.create_datawriter** (p. 1471) with **com.rti.dds.publication.Publisher.DATAWRITER_↩ QOS_DEFAULT** (p. 71) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.publication.Publisher.set_default_library** (p. 1478)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479)).

If the input profile cannot be found, the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT (p. 71)

com.rti.dds.publication.Publisher.create_datawriter_with_profile (p. 1473)

8.242.2.4 create_datawriter()

```
DataWriter create_datawriter (
             Topic topic,
             DataWriterQos qos,
             DataWriterListener listener,
            int mask )
```

Creates a **com.rti.dds.publication.DataWriter** (p. 553) that will be attached and belong to the **com.rti.dds.↩ publication.Publisher** (p. 1466).

For each application-defined type, **com.rti.ndds.example.Foo** (p. 1066), there is an implied, auto-generated class **com.rti.ndds.example.FooDataWriter** (p. 1097) that extends **com.rti.dds.publication.DataWriter** (p. 553) and contains the operations to write data of type **com.rti.ndds.example.Foo** (p. 1066).

Note that a common application pattern to construct the QoS for the **com.rti.dds.publication.DataWriter** (p. 553) is to:

- Retrieve the QoS policies on the associated **com.rti.dds.topic.Topic** (p. 1807) by means of the **com.rti.dds.topic.Topic.get_qos** (p. 1810) operation.
- Retrieve the default **com.rti.dds.publication.DataWriter** (p. 553) qos by means of the **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1469) operation.
- Combine those two QoS policies (for example, using **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1484)) and selectively modify policies as desired.

When a **com.rti.dds.publication.DataWriter** (p. 553) is created, only those transports already registered are available to the **com.rti.dds.publication.DataWriter** (p. 553). See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **com.rti.dds.publication.DataWriter** (p. 553) will be created.

The given **com.rti.dds.topic.Topic** (p. 1807) must have been created from the same participant as this publisher. If it was created from a different participant, this method will fail.

MT Safety:

UNSAFE. If **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) is used for the `qos` parameter, it is not safe to create the datawriter while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1469).

Parameters

topic	<<***in***>> *(p. 156)* The **com.rti.dds.topic.Topic** (p. 1807) that the **com.rti.dds.publication.DataWriter** (p. 553) will be associated with. Cannot be NULL.
qos	<<***in***>> *(p. 156)* QoS to be used for creating the new **com.rti.dds.publication.DataWriter** (p. 553). The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT** (p. 71) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 553) should be created with the default **com.rti.dds.publication.DataWriterQos** (p. 612) set in the **com.rti.dds.publication.Publisher** (p. 1466). The special value **com.rti.dds.publication.Publisher.DATAWRITER_QOS_USE_TOPIC_QOS** (p. 71) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 553) should be created with the combination of the default **com.rti.dds.publication.DataWriterQos** (p. 612) set on the **com.rti.dds.publication.Publisher** (p. 1466) and the **com.rti.dds.topic.TopicQos** (p. 1824) of the **com.rti.dds.topic.Topic** (p. 1807). Cannot be NULL.
listener	<<***in***>> *(p. 156)* The listener of the **com.rti.dds.publication.DataWriter** (p. 553).
mask	<<***in***>> *(p. 156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.publication.DataWriter** (p. 553) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

com.rti.dds.publication.Publisher.DATAWRITER_QOS_DEFAULT (p. 71)

com.rti.dds.publication.Publisher.DATAWRITER_QOS_USE_TOPIC_QOS (p. 71)

com.rti.dds.publication.Publisher.create_datawriter_with_profile (p. 1473)

com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1469)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1484)

com.rti.dds.publication.DataWriter.set_listener (p. 558)

8.242.2.5 create_datawriter_with_profile()

```
DataWriter create_datawriter_with_profile (
             Topic topic,
            String library_name,
            String profile_name,
             DataWriterListener listener,
            int mask )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.publication.DataWriter** (p. 553) object using the **com.rti.dds.publication.DataWriterQos** (p. 612) associated with the input XML QoS profile.

The **com.rti.dds.publication.DataWriter** (p. 553) will be attached and belong to the **com.rti.dds.publication.Publisher** (p. 1466).

For each application-defined type, **com.rti.ndds.example.Foo** (p. 1066), there is an implied, auto-generated class **com.rti.ndds.example.FooDataWriter** (p. 1097) that extends **com.rti.dds.publication.DataWriter** (p. 553) and contains the operations to write data of type **com.rti.ndds.example.Foo** (p. 1066).

When a **com.rti.dds.publication.DataWriter** (p. 553) is created, only those transports already registered are available to the **com.rti.dds.publication.DataWriter** (p. 553). See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

Precondition

If publisher is enabled, topic must have been enabled. Otherwise, this operation will fail and no **com.rti.dds.publication.DataWriter** (p. 553) will be created.

The given **com.rti.dds.topic.Topic** (p. 1807) must have been created from the same participant as this publisher. If it was created from a different participant, this method will return NULL.

Parameters

topic	<<***in***>> *(*p. *156)* The **com.rti.dds.topic.Topic** (p. 1807) that the **com.rti.dds.publication.DataWriter** (p. 553) will be associated with. Cannot be NULL.
library_name	<<***in***>> *(*p. *156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.publication.Publisher.set_default_library** (p. 1478)).
profile_name	<<***in***>> *(*p. *156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479)).
listener	<<***in***>> *(*p. *156)* The listener of the **com.rti.dds.publication.DataWriter** (p. 553).
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.publication.DataWriter** (p. 553) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.publication.DataWriterQos (p. 612) for rules on consistency among QoS

com.rti.dds.publication.Publisher.create_datawriter (p. 1471)

com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1469)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1484)

com.rti.dds.publication.DataWriter.set_listener (p. 558)

8.242.2.6 delete_datawriter()

```
void delete_datawriter (
               DataWriter a_datawriter )
```

Deletes a **com.rti.dds.publication.DataWriter** (p. 553) that belongs to the **com.rti.dds.publication.Publisher** (p. 1466).

The deletion of the **com.rti.dds.publication.DataWriter** (p. 553) will automatically unregister all instances.

Precondition

If the **com.rti.dds.publication.DataWriter** (p. 553) does not belong to the **com.rti.dds.publication.Publisher** (p. 1466), the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.publication.DataWriter** (p. 553) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datawriter	<<***in***>> *(*p. *156)* The **com.rti.dds.publication.DataWriter** (p. 553) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.242.2.7 lookup_datawriter()

```
DataWriter lookup_datawriter (
            String topic_name )
```

Retrieves the **com.rti.dds.publication.DataWriter** (p. 553) for a specific **com.rti.dds.topic.Topic** (p. 1807).

This returned **com.rti.dds.publication.DataWriter** (p. 553) is either enabled or disabled.

If more than one **com.rti.dds.publication.DataWriter** (p. 553) is attached to the **com.rti.dds.publication.Publisher** (p. 1466) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **com.rti.dds.publication.DataWriter** (p. 553) in one thread while another thread is simultaneously creating or destroying that **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

topic_name	<<***in***>> *(*p. *156)* Name of the **com.rti.dds.topic.Topic** (p. 1807) associated with the **com.rti.dds.publication.DataWriter** (p. 553) that is to be looked up. Cannot be NULL.

Returns

A **com.rti.dds.publication.DataWriter** (p. 553) that belongs to the **com.rti.dds.publication.Publisher** (p. 1466) attached to the **com.rti.dds.topic.Topic** (p. 1807) with `topic_name`. If no such **com.rti.dds.publication.DataWriter** (p. 553) exists, this operation returns NULL.

8.242.2.8 set_qos()

```
void set_qos (
              PublisherQos qos )
```

Sets the publisher QoS.

This operation modifies the QoS of the **com.rti.dds.publication.Publisher** (p. 1466).

The **com.rti.dds.publication.PublisherQos.group_data** (p. 1493), **com.rti.dds.publication.PublisherQos.partition** (p. 1493) and **com.rti.dds.publication.PublisherQos.entity_factory** (p. 1494) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 156)* **com.rti.dds.publication.PublisherQos** (p. 1490) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.publication.Publisher** (p. 1466) is enabled. The special value **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 46) can be used to indicate that the QoS of the **com.rti.dds.publication.Publisher** (p. 1466) should be changed to match the current default **com.rti.dds.publication.PublisherQos** (p. 1490) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.publication.PublisherQos (p. 1490) for rules on consistency among QoS

set_qos (abstract) (p. 1030)

Operations Allowed in Listener Callbacks (p. 1238)

8.242.2.9 set_qos_with_profile()

```
void set_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Change the QoS of this publisher using the input XML QoS profile.

This operation modifies the QoS of the **com.rti.dds.publication.Publisher** (p. 1466).

The **com.rti.dds.publication.PublisherQos.group_data** (p. 1493), **com.rti.dds.publication.PublisherQos.partition** (p. 1493) and **com.rti.dds.publication.PublisherQos.entity_factory** (p. 1494) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.publication.Publisher.set_default_library** (p. 1478)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.publication.PublisherQos (p. 1490) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1238)

8.242.2.10 get_qos()

```
void get_qos (
              PublisherQos qos )
```

Gets the publisher QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***in***>> *(p. 156)* **com.rti.dds.publication.PublisherQos** (p. 1490) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.242.2.11 get_default_library()

```
String get_default_library ( )
```

<<***extension***>> *(p. 155)* Gets the default XML library associated with a **com.rti.dds.publication.Publisher** (p. 1466).

Returns

The default library or null if the default library was not set.

See also

com.rti.dds.publication.Publisher.set_default_library (p. 1478)

8.242.2.12 set_default_library()

```
void set_default_library (
             String library_name )
```

<<***extension***>> *(p. 155)* Sets the default XML library for a **com.rti.dds.publication.Publisher** (p. 1466).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this **Publisher** (p. 1466)'s operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **com.rti.dds.publication.Publisher** (p. 1466) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 670) (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).

Parameters

library_name	<<***in***>> *(p. 156)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.publication.Publisher.get_default_library (p. 1478)

8.242.2.13 get_default_profile()

```
String get_default_profile ( )
```

<<***extension***>> *(p. 155)* Gets the default XML profile associated with a **com.rti.dds.publication.Publisher** (p. 1466).

Returns

The default profile or null if the default profile was not set.

See also

com.rti.dds.publication.Publisher.set_default_profile (p. 1479)

8.242.2.14 set_default_profile()

```
void set_default_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Sets the default XML profile for a **com.rti.dds.publication.Publisher** (p. 1466).

This method specifies the profile that will be used as the default the next time a default **Publisher** (p. 1466) profile is needed during a call to one of this **Publisher** (p. 1466)'s operations. When calling a **com.rti.dds.publication.Publisher** (p. 1466) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **com.rti.dds.publication.Publisher** (p. 1466) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 670) (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

This method does not set the default QoS for **com.rti.dds.publication.DataWriter** (p. 553) objects created by the **com.rti.dds.publication.Publisher** (p. 1466); for this functionality, use **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1470) (you may pass in NULL after having called **set_default_profile()** (p. 1479)).

This method does not set the default QoS for newly created Publishers; for this functionality, use **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos_with_profile** (p. 683).

Parameters

library_name	<<***in***>> *(p. 156)* The library name containing the profile.
profile_name	<<***in***>> *(p. 156)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.publication.Publisher.get_default_profile (p. 1479)

com.rti.dds.publication.Publisher.get_default_profile_library (p. 1480)

8.242.2.15 get_default_profile_library()

```
String get_default_profile_library ( )
```

<<***extension***>> *(p. 155)* Gets the library where the default XML QoS profile is contained for a **com.rti.dds.publication.Publisher** (p. 1466).

The default profile library is automatically set when **com.rti.dds.publication.Publisher.set_default_profile** (p. 1479) is called.

This library can be different than the **com.rti.dds.publication.Publisher** (p. 1466) default library (see **com.rti.dds.publication.Publisher.get_default_library** (p. 1478)).

Returns

The default profile library or null if the default profile was not set.

See also

com.rti.dds.publication.Publisher.set_default_profile (p. 1479)

8.242.2.16 set_listener()

```
void set_listener (
             PublisherListener l,
            int mask )
```

Sets the publisher listener.

Parameters

l	<<***in***>> *(p. 156)* **com.rti.dds.publication.PublisherListener** (p. 1488) to set to.
mask	<<***in***>> *(p. 156)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.publication.PublisherListener** (p. 1488).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.242.2.17 get_listener()

```
PublisherListener get_listener ( )
```

Get the publisher listener.

Returns

com.rti.dds.publication.PublisherListener (p. 1488) of the **com.rti.dds.publication.Publisher** (p. 1466).

See also

get_listener (abstract) (p. 1032)

8.242.2.18 suspend_publications()

```
void suspend_publications ( )
```

Indicates to RTI Connext that the application is about to make multiple modifications using **com.rti.dds.publication.DataWriter** (p. 553) objects belonging to the **com.rti.dds.publication.Publisher** (p. 1466).

It is a **hint** to RTI Connext so it can optimize its performance by e.g., holding the dissemination of the modifications and then batching them.

The use of this operation must be matched by a corresponding call to **com.rti.dds.publication.Publisher.resume_publications** (p. 1482) indicating that the set of modifications has completed.

If the **com.rti.dds.publication.Publisher** (p. 1466) is deleted before **com.rti.dds.publication.Publisher.resume_publications** (p. 1482) is called, any suspended updates yet to be published will be discarded.

RTI Connext is not required and does not currently make use of this hint in any way. However, similar results can be achieved by using *asynchronous publishing*. Combined with **com.rti.dds.publication.FlowController** (p. 1055), com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 553) instances allow the user even finer control of traffic shaping and sample coalescing.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.dds.publication.FlowController (p. 1055)

com.rti.dds.publication.FlowController.trigger_flow (p. 1058)

com.rti.dds.publication.FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME (p. 76)

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

8.242.2.19 resume_publications()

```
void resume_publications ( )
```

Indicates to RTI Connext that the application has completed the multiple changes initiated by the previous **com.rti.↩ dds.publication.Publisher.suspend_publications** (p. 1481).

This is a **hint** to RTI Connext that can be used for example, to batch all the modifications made since the **com.rti.↩ dds.publication.Publisher.suspend_publications** (p. 1481).

RTI Connext is not required and does not currently make use of this hint in any way. However, similar results can be achieved by using *asynchronous publishing*. Combined with **com.rti.dds.publication.FlowController** (p. 1055), com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_↩ MODE_QOS **com.rti.dds.publication.DataWriter** (p. 553) instances allow the user even finer control of traffic shaping and sample coalescing.

Precondition

A call to **com.rti.dds.publication.Publisher.resume_publications** (p. 1482) must match a previous call to **com.rti.dds.publication.Publisher.suspend_publications** (p. 1481). Otherwise the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.dds.publication.FlowController (p. 1055)

com.rti.dds.publication.FlowController.trigger_flow (p. 1058)

com.rti.dds.publication.FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME (p. 76)

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

8.242.2.20 begin_coherent_changes()

```
void begin_coherent_changes ( )
```

Indicates that the application will begin a coherent set of modifications using **com.rti.dds.publication.DataWriter** (p. 553) objects attached to the **com.rti.dds.publication.Publisher** (p. 1466).

A 'coherent set' is a set of modifications that must be propagated in such a way that they are interpreted at the receiver's side as a consistent set of modifications; that is, the receiver will only be able to access the data after all the modifications in the set are available at the receiver end.

A connectivity change may occur in the middle of a set of coherent changes; for example, the set of partitions used by the **com.rti.dds.publication.Publisher** (p. 1466) or one of its subscribers (**com.rti.dds.subscription.Subscriber** (p. 1730)) may change, a late-joining **com.rti.dds.subscription.DataReader** (p. 450) may appear on the network, or a communication failure may occur. In the event that such a change prevents an entity from receiving the entire set of coherent changes, that entity must behave as if it had received none of the set.

These calls can be nested. In that case, the coherent set terminates only with the last call to **com.rti.dds.publication.↩ Publisher.end_coherent_changes** (p. 1483). **Publisher** (p. 1466)'s samples (samples published by any of the Data↩ Writers within the **Publisher** (p. 1466)) that are not published within a begin_coherent_changes/end_coherent_changes block will not be provided to the DataReaders as a set.

The support for coherent changes enables a publishing application to change the value of several data-instances that could belong to the same or different topics and have those changes be seen *atomically* by the readers. This is useful in cases where the values are inter-related (for example, if there are two data-instances representing the altitude and velocity vector of the same aircraft and both are changed, it may be useful to communicate those values in a way the reader can see both together; otherwise, it may, e.g., erroneously interpret that the aircraft is on a collision course).

Note

Coherent sets don't apply to Topic Queries. If a **com.rti.dds.subscription.TopicQuery** (p. 1830) selects only a subset of samples that was published as a coherent set, the subscribing application will receive them regardless of their membership to the coherent set.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

com.rti.dds.infrastructure.PresentationQosPolicy (p. 1379)

8.242.2.21 end_coherent_changes()

```
void end_coherent_changes ( )
```

Terminates the coherent set initiated by the matching call to **com.rti.dds.publication.Publisher.begin_coherent_↩ changes** (p. 1483).

Precondition

If there is no matching call to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1483) the operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.242.2.22 copy_from_topic_qos()

```
void copy_from_topic_qos (
              DataWriterQos a_datawriter_qos,
              TopicQos a_topic_qos )
```

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.dds.↩ publication.DataWriterQos** (p. 612).

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.↩ dds.publication.DataWriterQos** (p. 612) (replacing values in the **com.rti.dds.publication.DataWriterQos** (p. 612), if present).

This is a "convenience" operation most useful in combination with the operations **com.rti.dds.publication.Publisher.↩ get_default_datawriter_qos** (p. 1469) and **com.rti.dds.topic.Topic.get_qos** (p. 1810). The operation **com.rti.dds.↩ publication.Publisher.copy_from_topic_qos** (p. 1484) can be used to merge the **com.rti.dds.publication.Data↩ Writer** (p. 553) default QoS policies with the corresponding ones on the **com.rti.dds.topic.Topic** (p. 1807). The resulting QoS can then be used to create a new **com.rti.dds.publication.DataWriter** (p. 553), or set its QoS.

This operation does not check the resulting **com.rti.dds.publication.DataWriterQos** (p. 612) for consistency. This is because the 'merged' **com.rti.dds.publication.DataWriterQos** (p. 612) may not be the final one, as the application can still modify some policies prior to applying the policies to the **com.rti.dds.publication.DataWriter** (p. 553).

Parameters

a_datawriter_qos	*<<**inout**>> (p. 156)* **com.rti.dds.publication.DataWriterQos** (p. 612) to be filled-up. Cannot be NULL.
a_topic_qos	*<<**in**>> (p. 156)* **com.rti.dds.topic.TopicQos** (p. 1824) to be merged with **com.rti.dds.publication.DataWriterQos** (p. 612). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.242.2.23 get_all_datawriters()

```
void get_all_datawriters (
             DataWriterSeq writers )
```

Retrieve all the DataWriters created from this **Publisher** (p. 1466).

Parameters

writers	<<***inout***>> *(p. 156)* Sequence where the DataWriters will be added

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.242.2.24 get_participant()

```
DomainParticipant get_participant ( )
```

Returns the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.publication.Publisher** (p. 1466) belongs.

Returns

the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.publication.Publisher** (p. 1466) belongs.

8.242.2.25 delete_contained_entities()

```
void delete_contained_entities ( )
```

Deletes all the entities that were created by means of the "create" operation on the **com.rti.dds.publication.Publisher** (p. 1466).

Deletes all contained **com.rti.dds.publication.DataWriter** (p. 553) objects. Once **com.rti.dds.publication.↩ Publisher.delete_contained_entities** (p. 1485) completes successfully, the application may delete the **com.rti.↩ dds.publication.Publisher** (p. 1466), knowing that it has no contained **com.rti.dds.publication.DataWriter** (p. 553) objects.

The operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if any of the contained entities is in a state where it cannot be deleted.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.242.2.26 wait_for_acknowledgments()

```
void wait_for_acknowledgments (
              Duration_t max_wait )
```

Blocks the calling thread until all data written by the **Publisher** (p. 1466)'s reliable DataWriters is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable DataWriters entities is acknowledged by (a) all reliable **com.rti.dds.subscription.DataReader** (p. 450) entities that are matched and alive and (b) by all required subscriptions, or until the duration specified by the `max_wait` parameter elapses, whichever happens first. A successful completion indicates that all the samples written have been acknowledged; a timeout indicates that max_wait elapsed before all the data was acknowledged.

Note that if a thread is blocked in the call to this operation on a **com.rti.dds.publication.Publisher** (p. 1466) and a different thread writes new samples on any of the reliable DataWriters that belong to this **Publisher** (p. 1466), the new samples must be acknowledged before unblocking the thread that is waiting on this operation.

If none of the **com.rti.dds.publication.DataWriter** (p. 553) instances have **com.rti.dds.infrastructure.Reliability↩ QosPolicy** (p. 1526) kind set to RELIABLE, the operation will complete successfully.

Parameters

max_wait	<<***in***>> *(*p. *156)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 840) .

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)

8.242.2.27 wait_for_asynchronous_publishing()

```
void wait_for_asynchronous_publishing (
```

```
Duration_t max_wait )
```

<<***extension***>> *(*p. *155)* Blocks the calling thread until asynchronous sending is complete.

This operation blocks the calling thread (up to `max_wait`) until all data written by the asynchronous **com.rti.dds.publication.DataWriter** (p. 553) entities is sent and acknowledged (if reliable) by all matched **com.rti.dds.subscription.DataReader** (p. 450) entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; if it times out, this indicates that `max_wait` elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort **com.rti.dds.subscription.DataReader** (p. 450) is complete in addition to what **com.rti.dds.publication.Publisher.wait_for_acknowledgments** (p. 1486) provides.

If none of the **com.rti.dds.publication.DataWriter** (p. 553) instances have **com.rti.dds.infrastructure.PublishModeQosPolicy.kind** (p. 1497) set to com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS, the operation will complete immediately , with com.rti.dds.infrastructure.RETCODE_OK.

Parameters

max_wait	<<***in***>> *(*p. *156)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 840).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597), **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599)

8.242.2.28 lookup_datawriter_by_name()

```
DataWriter lookup_datawriter_by_name (
            String datawriter_name )
```

<<***extension***>> *(*p. *155)* Retrieves a **com.rti.dds.publication.DataWriter** (p. 553) contained within the **com.rti.dds.publication.Publisher** (p. 1466) the **com.rti.dds.publication.DataWriter** (p. 553) entity name.

Every **com.rti.dds.publication.DataWriter** (p. 553) in the system has an entity name which is configured and stored in the <<***extension***>> *(*p. *155)* EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves the **com.rti.dds.publication.DataWriter** (p. 553) within the **com.rti.dds.publication.Publisher** (p. 1466) whose name matches the one specified. If there are several **com.rti.dds.publication.DataWriter** (p. 553) with the same name within the **com.rti.dds.publication.Publisher** (p. 1466), the operation returns the first matching occurrence.

Parameters

datawriter_name	<<***in***>> *(*p. *156)* Entity name of the **com.rti.dds.publication.DataWriter** (p. 553).

Returns

The first **com.rti.dds.publication.DataWriter** (p. 553) found with the specified name or NULL if it is not found.

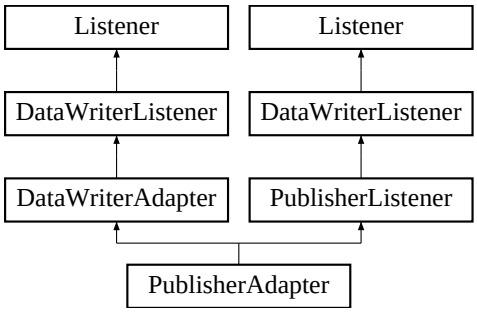
See also

com.rti.dds.domain.DomainParticipant.lookup_datawriter_by_name (p. 746)

8.243 PublisherAdapter Class Reference

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Inheritance diagram for PublisherAdapter:



Additional Inherited Members

8.243.1 Detailed Description

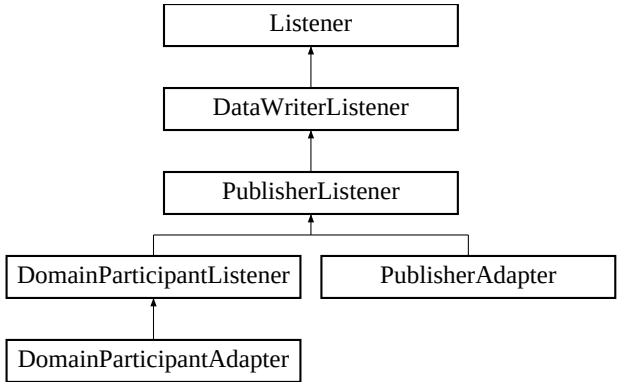
<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.244 PublisherListener Interface Reference

<<***interface***>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.publication.Publisher** (p. 1466) status.

Inheritance diagram for PublisherListener:



Additional Inherited Members

8.244.1 Detailed Description

<<*interface*>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.publication.Publisher** (p. 1466) status.

Entity:

com.rti.dds.publication.Publisher (p. 1466)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.LivelinessLostStatus** (p. 1242);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1339);
com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1340);
com.rti.dds.infrastructure.StatusKind.StatusKind.PUBLICATION_MATCHED_STATUS, **com.rti.dds.publication.PublicationMatchedStatus** (p. 1463);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1532);
com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS, **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1535)

See also

com.rti.dds.infrastructure.Listener (p. 1236)

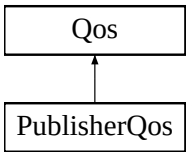
Status Kinds (p. 262)

Operations Allowed in Listener Callbacks (p. 1238)

8.245 PublisherQos Class Reference

QoS policies supported by a **com.rti.dds.publication.Publisher** (p. 1466) entity.

Inheritance diagram for PublisherQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**PublisherQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **PublisherQos** (p. 1490) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **PublisherQos** (p. 1490) object.*
- String **toString** (**PublisherQos** baseQos)

  *Obtains a string representation of a **PublisherQos** (p. 1490) object.*

Public Attributes

- final **PresentationQosPolicy presentation**

  *Presentation policy, **PRESENTATION** (p. 247).*
- final **PartitionQosPolicy partition**

  *Partition policy, **PARTITION** (p. 246).*
- final **GroupDataQosPolicy group_data**

  *Group data policy, **GROUP_DATA** (p. 236).*
- final **EntityFactoryQosPolicy entity_factory**

  *Entity factory policy, **ENTITY_FACTORY** (p. 234).*
- final **AsynchronousPublisherQosPolicy asynchronous_publisher**

  *<<**extension**>> (p. 155) Asynchronous publishing settings for the **com.rti.dds.publication.Publisher** (p. 1466) and all entities that are created by it.*
- final **EntityNameQosPolicy publisher_name**

  *<<**extension**>> (p. 155) EntityName policy, **ENTITY_NAME** (p. 234).*

8.245.1 Detailed Description

QoS policies supported by a **com.rti.dds.publication.Publisher** (p. 1466) entity.

You must set certain members in a consistent manner:

length of **com.rti.dds.infrastructure.GroupDataQosPolicy.value** (p. 1128) <= **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.publisher_group_data_max_length** (p. 815)

length of **com.rti.dds.infrastructure.PartitionQosPolicy.name** (p. 1367) <= **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.max_partitions** (p. 816)

combined number of characters (including terminating 0) in **com.rti.dds.infrastructure.PartitionQosPolicy.↩ name** (p. 1367) <= **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.max_partition_↩ cumulative_characters** (p. 816)

If any of the above are not true, **com.rti.dds.publication.Publisher.set_qos** (p. 1476) and **com.rti.dds.publication.↩ Publisher.set_qos_with_profile** (p. 1476) will fail with **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_↩ POLICY** (p. 1596) and **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 693) will\ return NULL.

8.245.2 Member Function Documentation

8.245.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 1491) overloads allow formatting the output and printing only the differences with respect to another **PublisherQos** (p. 1490) object.

```
PublisherQos qos = new PublisherQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
PublisherQos baseQos = new PublisherQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value PUBLISHER_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(PUBLISHER_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **PublisherQos** (p. 1490) and the documented default.

Returns

The string representation of the Qos.

References **PublisherQos.toString()**.

Referenced by **PublisherQos.toString()**.

8.245.2.2 toString() [2/4]

```
String toString (
              PublisherQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **PublisherQos** (p. 1490) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_PRINT_ALL** (p. 46) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipant.PUBLISHER_QOS_PRINT_ALL**.

8.245.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **PublisherQos** (p. 1490) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **PublisherQos.toString()**.

8.245.2.4 toString() [4/4]

```
String toString (
              PublisherQos baseQos )
```

Obtains a string representation of a **PublisherQos** (p. 1490) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.PUBLISHER_QOS_PRINT_ALL** (p. 46) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **PublisherQos.toString()**.

8.245.3 Member Data Documentation

8.245.3.1 presentation

```
final PresentationQosPolicy presentation
```

Presentation policy, **PRESENTATION** (p. 247).

8.245.3.2 partition

```
final PartitionQosPolicy partition
```

Partition policy, **PARTITION** (p. 246).

8.245.3.3 group_data

```
final GroupDataQosPolicy group_data
```

Group data policy, **GROUP_DATA** (p. 236).

8.245.3.4 entity_factory

```
final EntityFactoryQosPolicy entity_factory
```

Entity factory policy, **ENTITY_FACTORY** (p. 234).

8.245.3.5 asynchronous_publisher

```
final AsynchronousPublisherQosPolicy asynchronous_publisher
```

<<***extension***>> *(p. 155)* Asynchronous publishing settings for the **com.rti.dds.publication.Publisher** (p. 1466) and all entities that are created by it.

8.245.3.6 publisher_name

```
final EntityNameQosPolicy publisher_name
```

<<***extension***>> *(p. 155)* EntityName policy, **ENTITY_NAME** (p. 234).

8.246 PublisherSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.publication.Publisher** (p. 1466) > .

Inherits AbstractNativeSequence.

Public Member Functions

- **PublisherSeq** (Collection<?> publishers)
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.246.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.publication.Publisher** (p. 1466) > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.246.2 Constructor & Destructor Documentation

8.246.2.1 PublisherSeq()

```
PublisherSeq (
            Collection<?> publishers )
```

Exceptions

NullPointerException	if the given collection is null

8.246.3 Member Function Documentation

8.246.3.1 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **add()** (p. 329), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

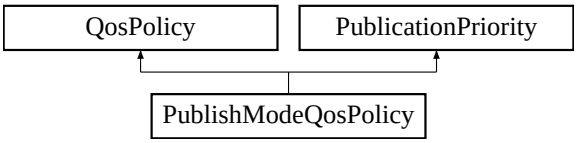
com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

8.247 PublishModeQosPolicy Class Reference

Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Inheritance diagram for PublishModeQosPolicy:



Public Attributes

- **PublishModeQosPolicyKind kind**
  Publishing mode.
- String **flow_controller_name**
  Name of the associated flow controller.
- int **priority** = **UNDEFINED**
  Publication priority.

Additional Inherited Members

8.247.1 Detailed Description

Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

The publishing mode of a **com.rti.dds.publication.DataWriter** (p. 553) determines whether data is written synchronously in the context of the user thread when calling **com.rti.ndds.example.FooDataWriter.write** (p. 1105) or asynchronously in the context of a separate thread internal to the middleware.

Each **com.rti.dds.publication.Publisher** (p. 1466) spawns a single asynchronous publishing thread (**com.rti.dds.↩ infrastructure.AsynchronousPublisherQosPolicy.thread** (p. 341)) to serve all its asynchronous **com.rti.dds.↩ publication.DataWriter** (p. 553) instances.

See also

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 339)

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

com.rti.dds.publication.FlowController (p. 1055)

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.247.2 Usage

The fastest way for RTI Connext to send data is for the user thread to execute the middleware code that actually sends the data itself. However, there are times when user applications may need or want an internal middleware thread to send the data instead. For instance, to send large data reliably, you must use an asynchronous thread.

When data is written asynchronously, a **com.rti.dds.publication.FlowController** (p. 1055), identified by `flow_controller_name`, can be used to shape the network traffic. Shaping a data flow usually means limiting the maximum data rates at which the middleware will send data for a **com.rti.dds.publication.DataWriter** (p. 553). The flow controller will buffer any excess data and only send it when the send rate drops below the maximum rate. The flow controller's properties determine when the asynchronous publishing thread is allowed to send data and how much.

Asynchronous publishing may increase latency, but offers the following advantages:

- The **com.rti.ndds.example.FooDataWriter.write** (p. 1105) call does not make any network calls and is therefore faster and more deterministic. This becomes important when the user thread is executing time-critical code.
- When data is written in bursts or when sending large data types as multiple fragments, a flow controller can throttle the send rate of the asynchronous publishing thread to avoid flooding the network.
- Asynchronously written samples for the same destination will be coalesced into a single network packet which reduces bandwidth consumption.

The maximum number of samples that will be coalesced depends on **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404) (each sample requires at least 2-4 gather-send buffers). Performance can be improved by increasing **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404). Note that the maximum value is operating system dependent.

The middleware must queue samples until they can be sent by the asynchronous publishing thread (as determined by the corresponding **com.rti.dds.publication.FlowController** (p. 1055)). The number of samples that will be queued is determined by the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144). When using com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, only the most recent **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) samples are kept in the queue. Once unsent samples are removed from the queue, they are no longer available to the asynchronous publishing thread and will therefore never be sent.

8.247.3 Member Data Documentation

8.247.3.1 kind

PublishModeQosPolicyKind kind

Publishing mode.

[default] com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.SYNCHRONOUS_PUBLISH_MODE_QOS

8.247.3.2 flow_controller_name

```
String flow_controller_name
```

Name of the associated flow controller.

The following builitin values are supported:

- **com.rti.dds.publication.FlowController.DEFAULT_FLOW_CONTROLLER_NAME** (p. 75)
- **com.rti.dds.publication.FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 75)
- **com.rti.dds.publication.FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 76)

See also

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 691)

[default] com.rti.dds.publication.FlowController.DEFAULT_FLOW_CONTROLLER_NAME (p. 75)

8.247.3.3 priority

```
int priority = UNDEFINED
```

Publication priority.

A positive integer value designating the relative priority of the **com.rti.dds.publication.DataWriter** (p. 553), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (com.rti.dds.infrastructure.PublishModeQosPolicy↩ Kind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS) with **com.rti.dds.publication.Flow↩ ControllerProperty_t.scheduling_policy** (p. 1059) set to com.rti.dds.publication.FlowControllerSchedulingPolicy.↩ FlowControllerSchedulingPolicy.HPF_FLOW_CONTROLLER_SCHED_POLICY.

Larger numbers have higher priority.

For multi-channel DataWriters, if the publication priority of any channel is set to any value other than com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will take precedence over that of the DataWriter.

For multi-channel DataWriters, if the publication priority of any channel is com.rti.dds.infrastructure.PUBLICATION_↩ PRIORITY_UNDEFINED, then the channel will inherit the publication priority of the DataWriter.

If the publicaton priority of the DataWriter, and of any channel of a multi-channel DataWriter, are com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the priority of the DataWriter or DataWriter channel will be assigned the lowest priority value.

If the publication priority of the DataWriter is com.rti.dds.infrastructure.PUBLICATION_PRIORITY_AUTOMATIC, then the DataWriter will be assigned the priority of the largest publication priority of all samples in the DataWriter.

The publication priority of each sample can be set in the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of the **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110) function.

For dispose and unregister samples, use the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of **com.rti.ndds.↩ example.FooDataWriter.dispose_w_params** (p. 1114) and **com.rti.ndds.example.FooDataWriter.unregister_↩ instance_w_params** (p. 1104).

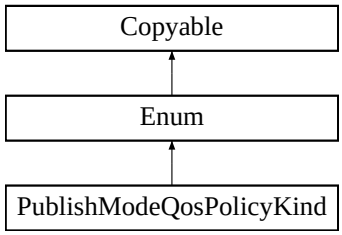
[default] com.rti.dds.infrastructure.PUBLICATION_PRIORITY_UNDEFINED

[range] [-1, MAX_INT]

8.248 PublishModeQosPolicyKind Class Reference

Kinds of publishing mode.

Inheritance diagram for PublishModeQosPolicyKind:



Static Public Attributes

- static final **PublishModeQosPolicyKind SYNCHRONOUS_PUBLISH_MODE_QOS**
  Indicates to send data synchronously.
- static final **PublishModeQosPolicyKind ASYNCHRONOUS_PUBLISH_MODE_QOS**
  Indicates to send data asynchronously.

Additional Inherited Members

8.248.1 Detailed Description

Kinds of publishing mode.

QoS:

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1496)

8.248.2 Member Data Documentation

8.248.2.1 SYNCHRONOUS_PUBLISH_MODE_QOS

```
final PublishModeQosPolicyKind SYNCHRONOUS_PUBLISH_MODE_QOS [static]
```

Indicates to send data synchronously.

If **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 598) is com.rti.dds.infrastructure.true, data is sent immediately in the context of **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

As data is sent immediately in the context of the user thread, no flow control is applied.

See also

com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write (p. 598)

[default] for **com.rti.dds.publication.DataWriter** (p. 553)

8.248.2.2 ASYNCHRONOUS_PUBLISH_MODE_QOS

```
final  PublishModeQosPolicyKind ASYNCHRONOUS_PUBLISH_MODE_QOS  [static]
```

Indicates to send data asynchronously.

Configures the **com.rti.dds.publication.DataWriter** (p. 553) to delegate the task of data transmission to a separate publishing thread. The **com.rti.ndds.example.FooDataWriter.write** (p. 1105) call does not send the data, but instead schedules the data to be sent later by its associated **com.rti.dds.publication.Publisher** (p. 1466).

Each **com.rti.dds.publication.Publisher** (p. 1466) uses its dedicated publishing thread (**com.rti.dds.publication.↩ PublisherQos.asynchronous_publisher** (p. 1494)) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated **com.rti.dds.publication.FlowController** (p. 1055) determines when the publishing thread is allowed to send the data.

com.rti.dds.publication.DataWriter.wait_for_asynchronous_publishing (p. 571) and **com.rti.dds.publication.↩ Publisher.wait_for_asynchronous_publishing** (p. 1486) enable you to determine when the data has actually been sent.

See also

com.rti.dds.publication.FlowController (p. 1055)

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

com.rti.dds.publication.DataWriter.wait_for_asynchronous_publishing (p. 571)

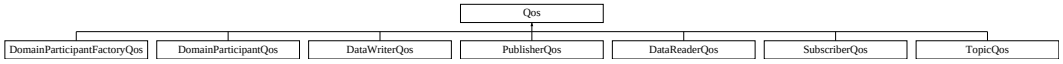
com.rti.dds.publication.Publisher.wait_for_asynchronous_publishing (p. 1486)

com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max (p. 1404)

8.249 Qos Class Reference

An abstract base class for all QoS types.

Inheritance diagram for Qos:



Public Member Functions

- final boolean **equals** (Object other)

8.249.1 Detailed Description

An abstract base class for all QoS types.

8.249.2 Member Function Documentation

8.249.2.1 equals()

```
final boolean equals (
             Object other )
```

Parameters

other	<<***in***>> *(p. 156)* Other object to compare with this QoS.

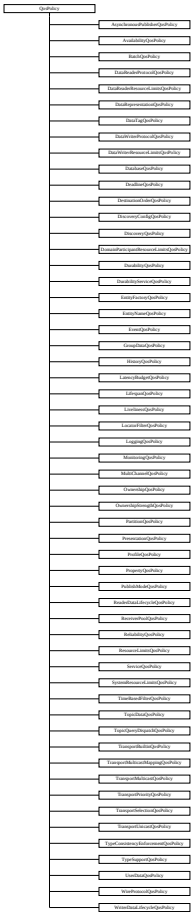
Returns

true if the given object is a QoS of the same concrete type as this QoS and all policies are equal, false otherwise

8.250 QosPolicy Class Reference

The base class for all QoS policies.

Inheritance diagram for QosPolicy:



Public Attributes

- final **QosPolicyId_t id**

  The ID of this QoS policy.

- final String **policy_name**

  The name of this QoS policy.

8.250.1 Detailed Description

The base class for all QoS policies.

8.250.2 Member Data Documentation

8.250.2.1 id

```
final  QosPolicyId_t id
```

The ID of this QoS policy.

This attribute is provided for more efficient comparisons of policy types that comparing strings.

8.250.2.2 policy_name

```
final String policy_name
```

The name of this QoS policy.

8.251 QosPolicyCount Class Reference

Type to hold a counter for a **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504).

Inherits Struct.

Public Member Functions

- **QosPolicyCount** (**QosPolicyCount** src)
  Copy constructor.

Public Attributes

- **QosPolicyId_t policy_id**
  *The **QosPolicy** (p. 1501) ID.*
- int **count**
  a counter

8.251.1 Detailed Description

Type to hold a counter for a **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504).

8.251.2 Constructor & Destructor Documentation

8.251.2.1 QosPolicyCount()

```
QosPolicyCount (
            QosPolicyCount src )
```

Copy constructor.

Parameters

src	**QosPolicyCount** (p. 1502) to copy from.

Exceptions

NullPointerException	if the source object is null.

References **QosPolicyCount.count**, and **QosPolicyCount.policy_id**.

8.251.3 Member Data Documentation

8.251.3.1 policy_id

```
QosPolicyId_t policy_id
```

The **QosPolicy** (p. 1501) ID.

Referenced by **QosPolicyCount.QosPolicyCount()**.

8.251.3.2 count

```
int count
```

a counter

Referenced by **QosPolicyCount.QosPolicyCount()**.

8.252 QosPolicyCountSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.QosPolicyCount** (p. 1502) >

Inherits ArraySequence.

8.252.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.QosPolicyCount** (p. 1502) >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

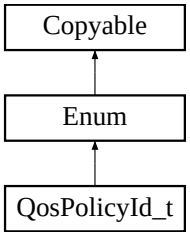
See also

com.rti.dds.infrastructure.QosPolicyCount (p. 1502)

8.253 QosPolicyId_t Class Reference

Type to identify `QosPolicies`.

Inheritance diagram for QosPolicyId_t:



Static Public Attributes

- static final **QosPolicyId_t INVALID_QOS_POLICY_ID**
  Identifier for an invalid QoS policy.
- static final **QosPolicyId_t USERDATA_QOS_POLICY_ID**
  Identifier for ***com.rti.dds.infrastructure.UserDataQosPolicy*** *(*p. *1959).*
- static final **QosPolicyId_t DURABILITY_QOS_POLICY_ID**
  Identifier for ***com.rti.dds.infrastructure.DurabilityQosPolicy*** *(*p. *830).*
- static final **QosPolicyId_t PRESENTATION_QOS_POLICY_ID**
  Identifier for ***com.rti.dds.infrastructure.PresentationQosPolicy*** *(*p. *1379).*
- static final **QosPolicyId_t DEADLINE_QOS_POLICY_ID**
  Identifier for ***com.rti.dds.infrastructure.DeadlineQosPolicy*** *(*p. *632).*

- static final **QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LatencyBudgetQosPolicy*** (p. *1231*).
- static final **QosPolicyId_t OWNERSHIP_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.OwnershipQosPolicy*** (p. *1342*).
- static final **QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy*** (p. *1348*).
- static final **QosPolicyId_t LIVELINESS_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LivelinessQosPolicy*** (p. *1243*).
- static final **QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy*** (p. *1804*).
- static final **QosPolicyId_t PARTITION_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.PartitionQosPolicy*** (p. *1365*).
- static final **QosPolicyId_t RELIABILITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.ReliabilityQosPolicy*** (p. *1526*).
- static final **QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DestinationOrderQosPolicy*** (p. *635*).
- static final **QosPolicyId_t HISTORY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.HistoryQosPolicy*** (p. *1144*).
- static final **QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.ResourceLimitsQosPolicy*** (p. *1590*).
- static final **QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.EntityFactoryQosPolicy*** (p. *1035*).
- static final **QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy*** (p. *2006*).
- static final **QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy*** (p. *1520*).
- static final **QosPolicyId_t TOPICDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TopicDataQosPolicy*** (p. *1819*).
- static final **QosPolicyId_t GROUPDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.GroupDataQosPolicy*** (p. *1127*).
- static final **QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TransportPriorityQosPolicy*** (p. *1859*).
- static final **QosPolicyId_t LIFESPAN_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LifespanQosPolicy*** (p. *1234*).
- static final **QosPolicyId_t DURABILITY_SERVICE_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DurabilityServiceQosPolicy*** (p. *837*).
- static final **QosPolicyId_t DATA_REPRESENTATION_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DataRepresentationQosPolicy*** (p. *544*).
- static final **QosPolicyId_t TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy*** (p. *1935*).
- static final **QosPolicyId_t DATATAG_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DataTagQosPolicy*** (p. *547*).
- static final **QosPolicyId_t WIREPROTOCOL_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for **com.rti.dds.infrastructure.WireProtocolQosPolicy*** (p. *1986*)
- static final **QosPolicyId_t DISCOVERY_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for **com.rti.dds.infrastructure.DiscoveryQosPolicy*** (p. *665*)
- static final **QosPolicyId_t DATAREADERRESOURCELIMITS_QOS_POLICY_ID**

<<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy*** (p. *529*)

- static final **QosPolicyId_t DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy*** (p. *626*)
- static final **QosPolicyId_t DATAREADERPROTOCOL_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DataReaderProtocolQosPolicy*** (p. *501*)
- static final **QosPolicyId_t DATAWRITERPROTOCOL_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DataWriterProtocolQosPolicy*** (p. *596*)
- static final **QosPolicyId_t DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy*** (p. *803*)
- static final **QosPolicyId_t EVENT_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.EventQosPolicy*** (p. *1044*)
- static final **QosPolicyId_t DATABASE_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DatabaseQosPolicy*** (p. *445*)
- static final **QosPolicyId_t RECEIVERPOOL_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.ReceiverPoolQosPolicy*** (p. *1523*)
- static final **QosPolicyId_t DISCOVERYCONFIG_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy*** (p. *646*)
- static final **QosPolicyId_t USEROBJECT_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for com.rti.dds.infrastructure.UserObjectQosPolicy*
- static final **QosPolicyId_t SYSTEMRESOURCELIMITS_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy*** (p. *1782*)
- static final **QosPolicyId_t TRANSPORTSELECTION_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TransportSelectionQosPolicy*** (p. *1860*)
- static final **QosPolicyId_t TRANSPORTUNICAST_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TransportUnicastQosPolicy*** (p. *1867*)
- static final **QosPolicyId_t TRANSPORTMULTICAST_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TransportMulticastQosPolicy*** (p. *1853*)
- static final **QosPolicyId_t TRANSPORTBUILTIN_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy*** (p. *1843*)
- static final **QosPolicyId_t PUBLISHMODE_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.PublishModeQosPolicy*** (p. *1496*)
- static final **QosPolicyId_t ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy*** (p. *339*)
- static final **QosPolicyId_t TYPESUPPORT_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** (p. *1941*)
- static final **QosPolicyId_t ENTITYNAME_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** (p. *1941*)
- static final **QosPolicyId_t SERVICE_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.ServiceQosPolicy*** (p. *1672*)
- static final **QosPolicyId_t BATCH_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.BatchQosPolicy*** (p. *355*)
- static final **QosPolicyId_t PROFILE_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.ProfileQosPolicy*** (p. *1393*)
- static final **QosPolicyId_t LOCATORFILTER_QOS_POLICY_ID**

  <<***extension***>> (p. *155*) *Identifier for* ***com.rti.dds.infrastructure.LocatorFilterQosPolicy*** (p. *1260*)
- static final **QosPolicyId_t MULTICHANNEL_QOS_POLICY_ID**

<<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.MultiChannelQosPolicy*** *(p. 1318)*

- static final **QosPolicyId_t AVAILABILITY_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.AvailabilityQosPolicy*** *(p. 343)*
- static final **QosPolicyId_t LOGGING_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.LoggingQosPolicy*** *(p. 1275)*
- static final **QosPolicyId_t TOPICQUERYDISPATCH_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy*** *(p. 1833)*
- static final **QosPolicyId_t MONITORING_QOS_POLICY_ID**

  <<***extension***>> *(p. 155) Identifier for* ***com.rti.dds.infrastructure.MonitoringQosPolicy*** *(p. 1314)*

Additional Inherited Members

8.253.1 Detailed Description

Type to identify `QosPolicies`.

8.253.2 Member Data Documentation

8.253.2.1 INVALID_QOS_POLICY_ID

```
final  QosPolicyId_t INVALID_QOS_POLICY_ID  [static]
```

Identifier for an invalid QoS policy.

8.253.2.2 USEROBJECT_QOS_POLICY_ID

```
final  QosPolicyId_t USEROBJECT_QOS_POLICY_ID  [static]
```

<<***extension***>> *(p. 155)* Identifier for com.rti.dds.infrastructure.UserObjectQosPolicy

8.254 QosPrintFormat Class Reference

A collection of attributes used to configure how a QoS appears when printed.

Public Attributes

- int **indent** = 0

  Conifgures how much additional indent should be added to the string representation of a QoS.
- boolean **print_private** = false

  Configures how private QoS policies should be printed.
- boolean **is_standalone** = false

  Controls whether or not to print valid XML for this QoS policy (through the inclusion of a preamble).

8.254.1 Detailed Description

A collection of attributes used to configure how a QoS appears when printed.

8.254.2 Member Data Documentation

8.254.2.1 indent

```
int indent = 0
```

Conifgures how much additional indent should be added to the string representation of a QoS.

Configures how much additional indent is applied when converting a QoS to a string. This value acts as a total offset on the string, increasing the indent which is applied to all elements by the same amount. With indent set to 0, a string representation of a QoS may appear as:

```
<datareader_qos>
    <durability>
        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
    </durability>
</datareader_qos>
```

Setting the indent property to 1, the same QoS would be printed as:

```
    <datareader_qos>
        <durability>
            <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
        </durability>
    </datareader_qos>
```

I.e., the entire structure is indented.

[default] 0

8.254.2.2 print_private

```
boolean print_private = false
```

Configures how private QoS policies should be printed.

There are some QoS policies which are not intended for external use. By default, these QoS policies are only printed if they are different to the default value for that policy. When true, private policies are treated like any other policy, and printed if they are different to the supplied base QoS. These private policies cannot be parsed from XML, and are always printed as XML comments.

[default] false

8.254.2.3 is_standalone

```
boolean is_standalone = false
```

Controls whether or not to print valid XML for this QoS policy (through the inclusion of a preamble).

The default value for this property (false) results in QoS being printed starting from the <ENTITY_qos> tag (where the <ENTITY_qos> tag is, e.g., <domain_participant_qos>, or, <datareader_qos>. This result cannot be directly parsed as valid XML (as it lacks <dds>, <qos_library>, and <qos_profile> tags). If is_standalone is set to true, these additional tags are printed, resulting in valid, parseable XML.

For example, with is_standalone set to false, a **Qos** (p. 1500) may appear as:

```
<datareader_qos>

    <durability>

        <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>

    </durability>

</datareader_qos>
```

Setting is_standalone to true would result in the same **Qos** (p. 1500) being printed as:

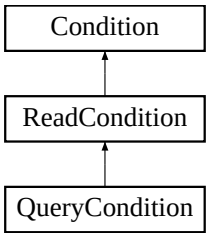
```
<?xml version="1.0"?>

<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <qos_library name="QosLibrary">

        <qos_profile name="QosProfile">

            <datareader_qos>

                <durability>

                    <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>

                </durability>

            </datareader_qos>

        </qos_profile>

    </qos_library>

</dds>
```

[default] false

8.255 QueryCondition Interface Reference

<<***interface***>> *(*p. *156)* These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1514) objects that allow the application to also specify a filter on the locally available data.

Inheritance diagram for QueryCondition:



Public Member Functions

- String **get_query_expression** ()

  Retrieves the query expression.
- void **get_query_parameters** (**StringSeq** query_parameters)

  Retrieves the query parameters.
- void **set_query_parameters** (**StringSeq** query_parameters)

  Sets the query parameters.

8.255.1 Detailed Description

<<***interface***>> *(*p. *156)* These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1514) objects that allow the application to also specify a filter on the locally available data.

Each query condition filter is composed of a **com.rti.dds.subscription.ReadCondition** (p. 1514) state filter and a content filter expressed as a `query_expression` and `query_parameters`.

The query (`query_expression`) is similar to an SQL WHERE clause and can be parameterised by arguments that are dynamically changeable by the **set_query_parameters()** (p. 1511) operation.

Two query conditions that have the same `query_expression` will require unique query condition content filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition content filter.

Queries and Filters Syntax (p. 104) describes the syntax of `query_expression` and `query_parameters`.

8.255.2 Member Function Documentation

8.255.2.1 get_query_expression()

```
String get_query_expression ( )
```

Retrieves the query expression.

8.255.2.2 get_query_parameters()

```
void get_query_parameters (
              StringSeq query_parameters )
```

Retrieves the query parameters.

Parameters

query_parameters	<<***inout***>> *(p. 156)* the query parameters are returned here.

8.255.2.3 set_query_parameters()

```
void set_query_parameters (
              StringSeq query_parameters )
```

Sets the query parameters.

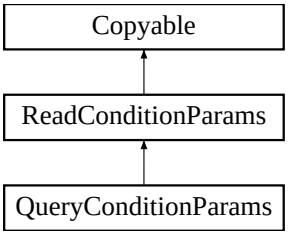
Parameters

query_parameters	<<***in***>> *(p. 156)* the new query parameters

8.256 QueryConditionParams Class Reference

<<***extension***>> *(p. 155)* Input parameters for **com.rti.dds.subscription.DataReader.create_querycondition_w↩ _params** (p. 456)

Inheritance diagram for QueryConditionParams:



Public Member Functions

- **QueryConditionParams** ()

  *The no-argument constructor for this **com.rti.dds.subscription.QueryConditionParams** (p. 1511) object.*
- **QueryConditionParams** (int **sample_states**, int **view_states**, int **instance_states**, int **stream_kinds**, String **query_expression**, **StringSeq query_parameters**)

  *The constructor for this **com.rti.dds.subscription.QueryConditionParams** (p. 1511) object.*
- Object **copy_from** (Object src)

  Copy value of a data type from source.

Public Attributes

- String **query_expression** = null

  Expression for the query.
- **StringSeq query_parameters** = new **StringSeq**()

  Parameters for the query expression.

8.256.1 Detailed Description

<<***extension***>> *(p. 155)* Input parameters for **com.rti.dds.subscription.DataReader.create_querycondition_w↩ _params** (p. 456)

8.256.2 Constructor & Destructor Documentation

8.256.2.1 QueryConditionParams() [1/2]

```
QueryConditionParams ( )
```

The no-argument constructor for this **com.rti.dds.subscription.QueryConditionParams** (p. 1511) object.

Referenced by **QueryConditionParams.copy_from()**.

8.256.2.2 QueryConditionParams() [2/2]

```
QueryConditionParams (
            int sample_states,
            int view_states,
            int instance_states,
            int stream_kinds,
            String query_expression,
             StringSeq query_parameters )
```

The constructor for this **com.rti.dds.subscription.QueryConditionParams** (p. 1511) object.

References **StringSeq.copy_from()**, **ReadConditionParams.instance_states**, **QueryConditionParams.query_expression**, **QueryConditionParams.query_parameters**, **ReadConditionParams.sample_states**, **ReadConditionParams.stream_kinds**, and **ReadConditionParams.view_states**.

8.256.3 Member Function Documentation

8.256.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Reimplemented from **ReadConditionParams** (p. 1518).

References **StringSeq.copy_from()**, **QueryConditionParams.query_expression**, **QueryConditionParams.↩ query_parameters**, and **QueryConditionParams.QueryConditionParams()**.

8.256.4 Member Data Documentation

8.256.4.1 query_expression

```
String query_expression = null
```

Expression for the query.

Cannot be NULL.

Referenced by **QueryConditionParams.copy_from()**, and **QueryConditionParams.QueryConditionParams()**.

8.256.4.2 query_parameters

```
StringSeq query_parameters = new StringSeq()
```

Parameters for the query expression.

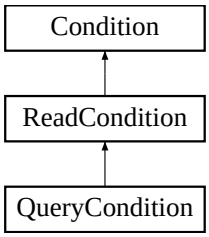
Cannot be NULL. .

Referenced by **QueryConditionParams.copy_from()**, and **QueryConditionParams.QueryConditionParams()**.

8.257 ReadCondition Interface Reference

<<***interface***>> *(*p. *156)* Conditions specifically dedicated to read operations and attached to one **com.rti.dds.↩ subscription.DataReader** (p. 450).

Inheritance diagram for ReadCondition:



Public Member Functions

- int **get_sample_state_mask** ()

  Retrieves the set of `sample_states` *for the condition.*
- int **get_view_state_mask** ()

  Retrieves the set of `view_states` *for the condition.*
- int **get_instance_state_mask** ()

  Retrieves the set of `instance_states` *for the condition.*
- int **get_stream_kind_mask** ()

  Retrieves the set of stream kind mask for the condition.
- **DataReader get_datareader** ()

  Returns the ***com.rti.dds.subscription.DataReader*** *(p. 450) associated with the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1514).*

8.257.1 Detailed Description

<<***interface***>> *(p. 156)* Conditions specifically dedicated to read operations and attached to one **com.rti.dds.subscription.DataReader** (p. 450).

com.rti.dds.subscription.ReadCondition (p. 1514) objects allow an application to specify the data samples it is interested in (by specifying the desired `sample_states`, `view_states` as well as `instance_states` in **com.rti.ndds.example.FooDataReader.read** (p. 1069) and **com.rti.ndds.example.FooDataReader.take** (p. 1071) variants.

This allows RTI Connext to enable the condition only when suitable information is available. They are to be used in conjunction with a WaitSet as normal conditions.

More than one **com.rti.dds.subscription.ReadCondition** (p. 1514) may be attached to the same **com.rti.dds.subscription.DataReader** (p. 450).

8.257.2 Member Function Documentation

8.257.2.1 get_sample_state_mask()

```
int get_sample_state_mask ( )
```

Retrieves the set of `sample_states` for the condition.

8.257.2.2 get_view_state_mask()

```
int get_view_state_mask ( )
```

Retrieves the set of `view_states` for the condition.

8.257.2.3 get_instance_state_mask()

```
int get_instance_state_mask ( )
```

Retrieves the set of `instance_states` for the condition.

8.257.2.4 get_stream_kind_mask()

```
int get_stream_kind_mask ( )
```

Retrieves the set of stream kind mask for the condition.

8.257.2.5 get_datareader()

```
DataReader get_datareader ( )
```

Returns the **com.rti.dds.subscription.DataReader** (p. 450) associated with the **com.rti.dds.subscription.ReadCondition** (p. 1514).

There is exactly one **com.rti.dds.subscription.DataReader** (p. 450) assicated with each **com.rti.dds.subscription.ReadCondition** (p. 1514).

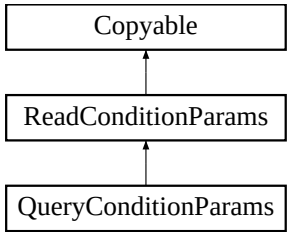
Returns

com.rti.dds.subscription.DataReader (p. 450) associated with the **com.rti.dds.subscription.ReadCondition** (p. 1514).

8.258 ReadConditionParams Class Reference

<<***extension***>> *(p. 155)* Input parameters for **com.rti.dds.subscription.DataReader.create_readcondition_w_params** (p. 455)

Inheritance diagram for ReadConditionParams:



Public Member Functions

- **ReadConditionParams** ()

  *The no-argument constructor for this **com.rti.dds.subscription.ReadConditionParams** (p. 1516) object.*
- **ReadConditionParams** (int **sample_states**, int **view_states**, int **instance_states**, int **stream_kinds**)

  *The constructor for this **com.rti.dds.subscription.ReadConditionParams** (p. 1516) object.*
- Object **copy_from** (Object src)

  Copy value of a data type from source.

Public Attributes

- int **sample_states** = **SampleStateKind.ANY_SAMPLE_STATE**

  Sample state.
- int **view_states** = **ViewStateKind.ANY_VIEW_STATE**

  View state.
- int **instance_states** = **InstanceStateKind.ANY_INSTANCE_STATE**

  Instance state.
- int **stream_kinds** = **StreamKind.ANY_STREAM**

  Stream kind.

8.258.1 Detailed Description

<<***extension***>> *(p. 155)* Input parameters for **com.rti.dds.subscription.DataReader.create_readcondition_w_↩params** (p. 455)

8.258.2 Constructor & Destructor Documentation

8.258.2.1 ReadConditionParams() [1/2]

```
ReadConditionParams ( )
```

The no-argument constructor for this **com.rti.dds.subscription.ReadConditionParams** (p. 1516) object.

Referenced by **ReadConditionParams.copy_from()**.

8.258.2.2 ReadConditionParams() [2/2]

```
ReadConditionParams (
            int sample_states,
            int view_states,
            int instance_states,
            int stream_kinds )
```

The constructor for this **com.rti.dds.subscription.ReadConditionParams** (p. 1516) object.

References **ReadConditionParams.instance_states**, **ReadConditionParams.sample_states**, **ReadCondition↩ Params.stream_kinds**, and **ReadConditionParams.view_states**.

8.258.3 Member Function Documentation

8.258.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

Reimplemented in **QueryConditionParams** (p. 1513).

References **ReadConditionParams.instance_states**, **ReadConditionParams.ReadConditionParams()**, **ReadConditionParams.sample_states**, **ReadConditionParams.stream_kinds**, and **ReadConditionParams.view_states**.

8.258.4 Member Data Documentation

8.258.4.1 sample_states

```
int sample_states = SampleStateKind.ANY_SAMPLE_STATE
```

Sample state.

Sample state of the data samples that are of interest.

Referenced by **ReadConditionParams.copy_from()**, **QueryConditionParams.QueryConditionParams()**, and **ReadConditionParams.ReadConditionParams()**.

8.258.4.2 view_states

```
int view_states = ViewStateKind.ANY_VIEW_STATE
```

View state.

View state of the data samples that are of interest.

Referenced by **ReadConditionParams.copy_from()**, **QueryConditionParams.QueryConditionParams()**, and **ReadConditionParams.ReadConditionParams()**.

8.258.4.3 instance_states

```
int instance_states = InstanceStateKind.ANY_INSTANCE_STATE
```

Instance state.

Instance state of the data samples that are of interest.

Referenced by **ReadConditionParams.copy_from()**, **QueryConditionParams.QueryConditionParams()**, and **ReadConditionParams.ReadConditionParams()**.

8.258.4.4 stream_kinds

```
int stream_kinds =  StreamKind.ANY_STREAM
```

Stream kind.

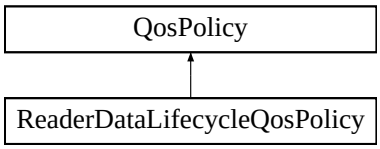
Stream kind of the data samples that are of interest.

Referenced by **ReadConditionParams.copy_from()**, **QueryConditionParams.QueryConditionParams()**, and **ReadConditionParams.ReadConditionParams()**.

8.259 ReaderDataLifecycleQosPolicy Class Reference

Controls how a DataReader manages the lifecycle of the data that it has received.

Inheritance diagram for ReaderDataLifecycleQosPolicy:



Public Attributes

- final **Duration_t autopurge_nowriter_samples_delay**

  *Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding an instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).*
- final **Duration_t autopurge_disposed_samples_delay**

  *Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain samples for an instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161).*
- final **Duration_t autopurge_disposed_instances_delay**

  *<<**extension**>> (p. 155) Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information about a received instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) and there are no samples for the instance in the DataReader queue.*
- final **Duration_t autopurge_nowriter_instances_delay**

  *<<**extension**>> (p. 155) Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information about a received instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) and there are no samples for the instance in the DataReader queue.*

8.259.1 Detailed Description

Controls how a DataReader manages the lifecycle of the data that it has received.

When a DataReader receives data, it is stored in a receive queue for the DataReader. The user application may either take the data from the queue or leave it there.

This QoS policy controls whether or not RTI Connext will automatically remove data from the receive queue (so that user applications cannot access it afterwards) when it detects that there are no more DataWriters alive for that data. It specifies how long a **com.rti.dds.subscription.DataReader** (p. 450) must retain information regarding instances that have the instance_state **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **com.rti.dds.subscription.DataReader** (p. 450) internally maintains the samples that have not been "taken" by the application, subject to the constraints imposed by other QoS policies such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

The **com.rti.dds.subscription.DataReader** (p. 450) also maintains information regarding the identity, `view_state` and `instance_state` of data instances even after all samples have been taken. This is needed to properly compute the states when future samples arrive.

Under normal circumstances the **com.rti.dds.subscription.DataReader** (p. 450) can only reclaim all resources for instances for which there are no writers and for which all samples have been 'taken'. The last sample the **com.rti.dds.subscription.DataReader** (p. 450) will have taken for that instance will have an `instance_state` of either **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) or **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 256), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **com.rti.dds.subscription.DataReader** (p. 450) from reclaiming the resources and they would remain in the **com.rti.dds.subscription.DataReader** (p. 450) indefinitely.

A DataReader can also reclaim all resources for instances that have an instance state of **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) and for which all DDS samples have been 'taken'. DataReaders will only reclaim resources in this situation when the autopurge_disposed_instances_delay has been set to zero.

For keyed Topics, the consideration of removing data samples from the receive queue is done on a per instance (key) basis. Thus when RTI Connext detects that there are no longer DataWriters alive for a certain key value of a Topic (an instance of the Topic), it can be configured to remove all data samples for that instance (key).

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **YES** (p. 256)

8.259.2 Member Data Documentation

8.259.2.1 autopurge_nowriter_samples_delay

```
final Duration_t autopurge_nowriter_samples_delay
```

Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information regarding an instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_↩ WRITERS_INSTANCE_STATE** (p. 1161).

At some point after this time elapses, the **com.rti.dds.subscription.DataReader** (p. 450) will purge all internal information regarding the instance, any "untaken" samples will also be dropped.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 nanosec, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.259.2.2 autopurge_disposed_samples_delay

```
final Duration_t autopurge_disposed_samples_delay
```

Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain samples for an instance once its `instance_state` becomes **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_↩ INSTANCE_STATE** (p. 1161).

After this time elapses, the **com.rti.dds.subscription.DataReader** (p. 450) will purge all samples for the instance even if they have not been read by the application. This purge is done lazily when space is needed for other samples or instances.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

[range] [1 nanosec, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.259.2.3 autopurge_disposed_instances_delay

```
final Duration_t autopurge_disposed_instances_delay
```

<<***extension***>> *(p. 155)* Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information about a received instance once its `instance_state` becomes **com.rti.dds.subscription.Instance↩ StateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) and there are no samples for the instance in the DataReader queue.

After this time elapses, when the last sample for the disposed instance is taken, the **com.rti.dds.subscription.Data↩ Reader** (p. 450) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529).

The only currently supported values are com.rti.dds.infrastructure.Duration_t.ZERO and **com.rti.dds.infrastructure.↩ Duration_t.DURATION_INFINITE** (p. 846). A value of com.rti.dds.infrastructure.Duration_t.ZERO will purge an instance's state immediately after the instance state transitions to **com.rti.dds.subscription.InstanceStateKind.NOT↩ _ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161), as long as all samples, including the dispose sample, associated with that instance have been 'taken'.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

8.259.2.4 autopurge_nowriter_instances_delay

```
final  Duration_t autopurge_nowriter_instances_delay
```

<<***extension***>> *(p. 155)* Minimum duration for which the **com.rti.dds.subscription.DataReader** (p. 450) will maintain information about a received instance once its `instance_state` becomes **com.rti.dds.subscription.Instance↩ StateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) and there are no samples for the instance in the DataReader queue.

An instance will transition to the **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_↩ INSTANCE_STATE** (p. 1161) `instance_state` when all known writers for the instance have either lost liveliness or have unregistered themselves from the instance. After this time elapses, when the last sample for the instance without writers is taken, the **com.rti.dds.subscription.DataReader** (p. 450) will keep only the minimum state about the instance.

If you do not want to keep this minimum state after the delay period, also set keep_minimum_state_for_instances to FALSE in **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 529).

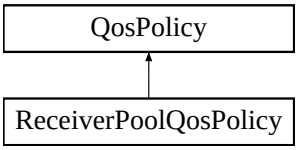
The only currently supported values are com.rti.dds.infrastructure.Duration_t.ZERO and **com.rti.dds.infrastructure.↩ Duration_t.DURATION_INFINITE** (p. 846). A value of com.rti.dds.infrastructure.Duration_t.ZERO will purge an instance's state immediately after the instance state transitions to **com.rti.dds.subscription.InstanceStateKind.NOT↩ _ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161), as long as all samples, including the no_writers sample, associated with that instance have been 'taken'.

[default] com.rti.dds.infrastructure.Duration_t.ZERO

8.260 ReceiverPoolQosPolicy Class Reference

Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Inheritance diagram for ReceiverPoolQosPolicy:



Public Attributes

- final **ThreadSettings_t thread**

  Receiver pool thread(s).
- int **buffer_size**

  The receive buffer size in bytes.
- int **buffer_alignment**

  The receive buffer alignment.

Static Public Attributes

- static final int **LENGTH_AUTO**

  A special value indicating that the actual value will be automatically resolved.

8.260.1 Detailed Description

Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

See also

Controlling CPU Core Affinity for RTI Threads (p. 1795)

8.260.2 Usage

This QoS policy sets the thread properties such as priority level and stack size for the threads used by the middleware to receive and process data from transports.

RTI uses a separate receive thread per port per transport plug-in. To force RTI Connext to use a separate thread to process the data for a **com.rti.dds.subscription.DataReader** (p. 450), set a unique port for the **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1867) or **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853) for the **com.rti.dds.subscription.DataReader** (p. 450).

This QoS policy also sets the size of the buffer used to store packets received from a transport. This buffer size will limit the largest single packet of data that a **com.rti.dds.domain.DomainParticipant** (p. 670) will accept from a transport. Users will often set this size to the largest packet that any of the transports used by their application will deliver. For many applications, the value 65,536 (64 K) is a good choice; this value is the largest packet that can be sent/received via UDP.

8.260.3 Member Data Documentation

8.260.3.1 thread

```
final ThreadSettings_t thread
```

Receiver pool thread(s).

There is at least one receive thread, possibly more.

[default] priority above normal.
The actual value depends on your architecture:

For Windows: 2
For Linux: OS default priority
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] The actual value depends on your architecture:

For Windows: OS default stack size
For Linux: OS default stack size
For a complete list of platform specific values, please refer to `Platform Notes`.

[default] mask **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_FLOATING_POINT** (p. 1797) | **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1797)

8.260.3.2 buffer_size

```
int buffer_size
```

The receive buffer size in bytes.

The receive buffer is used by the receive thread to store the raw data that arrives over the transports in non-zero-copy transports.

Zero-copy transports do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in buffers allocated by the transports themselves. Only the shared memory built-in transport (SHMEM) supports zero-copy.

buffer_size must always be at least as large as the maximum **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) across *all* of the transports being used that are not doing zero-copy.

By default (**com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257)): the size is equal to the maximum **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) across *all* of the non-zero-copy transports.

You may want the value to be greater than the default if you try to limit the largest data packet that can be sent through the transport(s) in one application, but you still want to receive data from other applications that have not made the same change.

For example, to avoid IP fragmentation, you may want to set **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) for IP-based transports to a small value, such as 1400 bytes. However, you may not be able to apply this change to all the applications at the same time. To receive data from these other applications the buffer_size should be equal to the original **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404).

[default] com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO (p. 257)

[range] [1, 1 GB] or **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.LENGTH_AUTO** (p. 257)

8.260.3.3 buffer_alignment

```
int buffer_alignment
```

The receive buffer alignment.

Most users will not need to change this alignment.

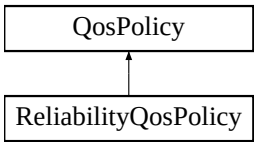
[default] 16

[range] [1,1024] Value must be a power of 2.

8.261 ReliabilityQosPolicy Class Reference

Indicates the level of reliability offered/requested by RTI Connext.

Inheritance diagram for ReliabilityQosPolicy:



Public Attributes

- **ReliabilityQosPolicyKind kind**
  Kind of reliability.
- final **Duration_t max_blocking_time**
  The maximum time a DataWriter can block on a write() call.
- **ReliabilityQosPolicyAcknowledgmentModeKind acknowledgment_kind**
  *<<**extension**>> (p. 155) Kind of reliable acknowledgment*
- **InstanceStateConsistencyKind instance_state_consistency_kind**
  *<<**extension**>> (p. 155) Whether instance state consistency is enabled*

8.261.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Connext.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.↩ DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, com.rti.dds.↩ infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 256) = YES
Changeable (p. 256) = **UNTIL ENABLE** (p. 256) (the instance_state_consistency_kind is **Changeable** (p. 256) = **NO** (p. 256))

8.261.2 Usage

This policy indicates the level of reliability requested by a **com.rti.dds.subscription.DataReader** (p. 450) or offered by a **com.rti.dds.publication.DataWriter** (p. 553).

The reliability of a connection between a DataWriter and DataReader is entirely user configurable. It can be done on a per DataWriter/DataReader connection. A connection may be configured to be "best effort" which means that RTI Connext will not use any resources to monitor or guarantee that the data sent by a DataWriter is received by a DataReader.

For some use cases, such as the periodic update of sensor values to a GUI displaying the value to a person, **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532) delivery is often good enough. It is certainly the fastest, most efficient, and least resource-intensive (CPU and network bandwidth) method of getting the newest/latest value for a topic from DataWriters to DataReaders. But there is no guarantee that the data sent will be received. It may be lost due to a variety of factors, including data loss by the physical transport such as wireless RF or even Ethernet.

However, there are data streams (topics) in which you want an absolute guarantee that all data sent by a DataWriter is received reliably by DataReaders. This means that RTI Connext must check whether or not data was received, and repair any data that was rejected by resending a copy of the data as many times as it takes for the DataReader to receive the data. RTI Connext uses a reliability protocol configured and tuned by these QoS policies: **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144), **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596), **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 501), and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1590).

The Reliability QoS policy is simply a switch to turn on the reliability protocol for a DataWriter/DataReader connection. The level of reliability provided by RTI Connext is determined by the configuration of the aforementioned QoS policies.

You can configure RTI Connext to deliver *all* data in the order they were sent (also known as absolute or strict reliability). Or, as a tradeoff for less memory, CPU, and network usage, you can choose a reduced level of reliability where only the last *N* values are guaranteed to be delivered reliably to DataReaders (where *N* is user-configurable). In the reduced level of reliability, there are no guarantees that the data sent before the last *N* are received. Only the last *N* data packets are monitored and repaired if necessary.

These levels are ordered, **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532) $<$ **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532). A **com.rti.dds.publication.DataWriter** (p. 553) offering one level is implicitly offering all levels below.

Note: To send *large* data reliably, you will also need to set com.rti.dds.infrastructure.PublishModeQosPolicyKind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS. *Large* in this context means that the data cannot be sent as a single packet by the transport (for example, data larger than 63K when using UDP/IP).

The setting of this policy has a dependency on the setting of the **RESOURCE_LIMITS** (p. 259) policy. In case the reliability kind is set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) the write operation on the **com.rti.dds.publication.DataWriter** (p. 553) may block if the modification would cause data to be lost or else cause one of the limits in specified in the **RESOURCE_LIMITS** (p. 259) to be exceeded. Under these circumstances, the **RELIABILITY** (p. 258) `max_blocking_time` configures the maximum duration the write operation may block.

If the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) is set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), data samples originating from a single **com.rti.dds.publication.DataWriter** (p. 553) cannot be made available to the **com.rti.dds.subscription.DataReader** (p. 450) if there are previous data samples that have not been received yet due to a communication error. In other words, RTI Connext will repair the error and resend data samples as needed in order to reconstruct a correct snapshot

of the **com.rti.dds.publication.DataWriter** (p. 553) history before it is accessible by the **com.rti.dds.subscription.↩ DataReader** (p. 450).

If the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) is set to **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), the service will not re-transmit missing data samples. However, for data samples originating from any one DataWriter the service will ensure they are stored in the **com.rti.dds.subscription.DataReader** (p. 450) history in the same order they originated in the **com.rti.dds.↩ publication.DataWriter** (p. 553). In other words, the **com.rti.dds.subscription.DataReader** (p. 450) may miss some data samples, but it will never see the value of a data object change from a newer value to an older value.

See also

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.261.3 Compatibility

The value offered is considered compatible with the value requested if and only if:

- the inequality *offered* `kind` >= *requested* `kind` evaluates to 'TRUE'. For the purposes of this inequality, the values of **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) are considered ordered such that **com.↩ rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532) < **com.rti.↩ dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).

- The offered acknowledgment_kind = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgment↩ ModeKind.PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1530) and requested acknowledgment_kind = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.PROTOCOL_ACKNOWLEDGMENT↩ _MODE** (p. 1530) OR offered acknowledgment_kind != **com.rti.dds.infrastructure.ReliabilityQosPolicy↩ AcknowledgmentModeKind.PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1530).

- the inequality *offered* `instance_state_consistency_kind` >= *requested* `instance_state↩ _consistency_kind` evaluates to 'TRUE'. For the purposes of this inequaility, the values of **com.↩ rti.dds.infrastructure.ReliabilityQosPolicy.instance_state_consistency_kind** (p. 1529) are considered ordered such that **com.rti.dds.infrastructure.InstanceStateConsistencyKind.NO_RECOVER_INSTANCE_↩ STATE_CONSISTENCY** (p. 1159) < **com.rti.dds.infrastructure.InstanceStateConsistencyKind.RECOVER↩ _INSTANCE_STATE_CONSISTENCY** (p. 1159).

8.261.4 Member Data Documentation

8.261.4.1 kind

ReliabilityQosPolicyKind kind

Kind of reliability.

[default] **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532) for **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.infrastructure.↩ ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532) for **com.rti.dds.publication.DataWriter** (p. 553)

8.261.4.2 max_blocking_time

`final` **`Duration_t`** `max_blocking_time`

The maximum time a DataWriter can block on a write() call.

See `Writing Data`, in the Core Libraries User's Manual, for more information about when the write() call can block and what happens when it times out.

Has no meaning for DataReaders.

[default] 100 milliseconds

[range] [0,1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.261.4.3 acknowledgment_kind

`ReliabilityQosPolicyAcknowledgmentModeKind` `acknowledgment_kind`

<<***extension***>> *(p. 155)* Kind of reliable acknowledgment

This setting applies only to the case where **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) = **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).

Sets the kind acknowledgments supported by a **com.rti.dds.publication.DataWriter** (p. 553) and sent by **com.rti.dds.subscription.DataReader** (p. 450).

[default] com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.PROTOCOL_ACKNOWLEDGMENT_MODE (p. 1530)

8.261.4.4 instance_state_consistency_kind

`InstanceStateConsistencyKind` `instance_state_consistency_kind`

<<***extension***>> *(p. 155)* Whether instance state consistency is enabled

A DataReader that determines that the DataWriter is no longer alive will transition instances to NOT_ALIVE_NO_WRITERS if there are no other DataWriters updating that instance. If the DataReader rediscovers the DataWriter, the DataReader does not automatically transition instances back to the state they were in prior to the disconnection until it gets updates (i.e., samples) for them.

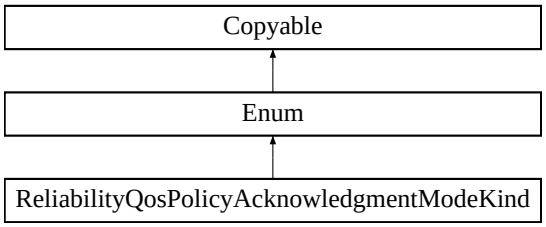
If **`InstanceStateConsistencyKind`** (p. `1158`) is set to `RECOVER_INSTANCE_STATE_CONSISTENCY`, then when the DataReader rediscovers a DataWriter, the DataReader will query the DataWriter for the state of its instances, and restore the instances to their correct state.

[default] com.rti.dds.infrastructure.InstanceStateConsistencyKind.NO_RECOVER_INSTANCE_STATE_CONSISTENCY (p. 1159) for **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.infrastructure.InstanceStateConsistencyKind.RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 1159) for **com.rti.dds.publication.DataWriter** (p. 553)

8.262 ReliabilityQosPolicyAcknowledgmentModeKind Class Reference

<<***extension***>> *(p. 155)* Kinds of acknowledgment.

Inheritance diagram for ReliabilityQosPolicyAcknowledgmentModeKind:



Static Public Attributes

- static final **ReliabilityQosPolicyAcknowledgmentModeKind PROTOCOL_ACKNOWLEDGMENT_MODE**

  Samples are acknowledged by RTPS protocol.
- static final **ReliabilityQosPolicyAcknowledgmentModeKind APPLICATION_AUTO_ACKNOWLEDGMENT↩ _MODE**

  Samples are acknowledged automatically after a subscribing application has accessed them.
- static final **ReliabilityQosPolicyAcknowledgmentModeKind APPLICATION_EXPLICIT_ACKNOWLEDGMENT↩ _MODE**

  Samples are acknowledged after the subscribing application explicitly calls acknowledge on the samples.

Additional Inherited Members

8.262.1 Detailed Description

<<***extension***>> *(p. 155)* Kinds of acknowledgment.

QoS:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

8.262.2 Member Data Documentation

8.262.2.1 PROTOCOL_ACKNOWLEDGMENT_MODE

`final` **`ReliabilityQosPolicyAcknowledgmentModeKind`** `PROTOCOL_ACKNOWLEDGMENT_MODE [static]`

Samples are acknowledged by RTPS protocol.

Samples are acknowledged according to the Real-Time Publish-Subscribe (RTPS) interoperability protocol.

8.262.2.2 APPLICATION_AUTO_ACKNOWLEDGMENT_MODE

```
final ReliabilityQosPolicyAcknowledgmentModeKind APPLICATION_AUTO_ACKNOWLEDGMENT_MODE [static]
```

Samples are acknowledged automatically after a subscribing application has accessed them.

A sample received by a **com.rti.ndds.example.FooDataReader** (p. 1067) is acknowledged after the subscribing application accesses it, either through calling **com.rti.ndds.example.FooDataReader.take** (p. 1071) or **com.rti.ndds.example.FooDataReader.read** (p. 1069) on the DDS sample. If the read or take operation loans the samples, the acknowledgment is done after **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) is called. Otherwise, for read or take operations that make a copy, acknowledgment is done after the read or take operations are executed.

8.262.2.3 APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE

```
final ReliabilityQosPolicyAcknowledgmentModeKind APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE [static]
```

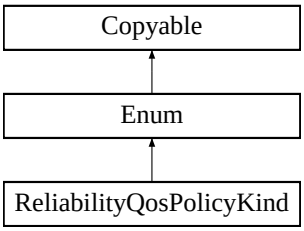
Samples are acknowledged after the subscribing application explicitly calls acknowledge on the samples.

Samples received by a **com.rti.dds.subscription.DataReader** (p. 450) are explicitly acknowledged by the subscribing application, after it calls either **com.rti.dds.subscription.DataReader.acknowledge_all** (p. 471) or **com.rti.dds.subscription.DataReader.acknowledge_sample** (p. 470).

8.263 ReliabilityQosPolicyKind Class Reference

Kinds of reliability.

Inheritance diagram for ReliabilityQosPolicyKind:



Static Public Attributes

- static final **ReliabilityQosPolicyKind BEST_EFFORT_RELIABILITY_QOS**

  Indicates that it is acceptable to not retry propagation of any samples.
- static final **ReliabilityQosPolicyKind RELIABLE_RELIABILITY_QOS**

  Specifies RTI Connext will attempt to deliver all samples in its history. Missed samples may be retried.

Additional Inherited Members

8.263.1 Detailed Description

Kinds of reliability.

QoS:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

8.263.2 Member Data Documentation

8.263.2.1 BEST_EFFORT_RELIABILITY_QOS

`final` **ReliabilityQosPolicyKind** `BEST_EFFORT_RELIABILITY_QOS [static]`

Indicates that it is acceptable to not retry propagation of any samples.

Presumably new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples.

[default] for **com.rti.dds.subscription.DataReader** (p. 450) and **com.rti.dds.topic.Topic** (p. 1807)

8.263.2.2 RELIABLE_RELIABILITY_QOS

`final` **ReliabilityQosPolicyKind** `RELIABLE_RELIABILITY_QOS [static]`

Specifies RTI Connext will attempt to deliver all samples in its history. Missed samples may be retried.

In steady-state (no modifications communicated via the **com.rti.dds.publication.DataWriter** (p. 553)), RTI Connext guarantees that all samples in the **com.rti.dds.publication.DataWriter** (p. 553) history will eventually be delivered to all the **com.rti.dds.subscription.DataReader** (p. 450) objects (subject to timeouts that indicate loss of communication with a particular **com.rti.dds.subscription.Subscriber** (p. 1730)).

Outside steady state, the **HISTORY** (p. 237) and **RESOURCE_LIMITS** (p. 259) policies will determine how samples become part of the history and whether samples can be discarded from it.

[default] for **com.rti.dds.publication.DataWriter** (p. 553)

8.264 ReliableReaderActivityChangedStatus Class Reference

<<***extension***>> *(p. 155)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Inherits Status.

Public Member Functions

- **ReliableReaderActivityChangedStatus** ()

  Construct a new status object with default contents.
- **ReliableReaderActivityChangedStatus** (**ReliableReaderActivityChangedStatus** src)

  Construct a new status identical to the given status.

Public Attributes

- int **active_count**

  The current number of reliable readers currently matched with this reliable writer.
- int **inactive_count**

  The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.
- int **active_count_change**

  The most recent change in the number of active remote reliable readers.
- int **inactive_count_change**

  The most recent change in the number of inactive remote reliable readers.
- final **InstanceHandle_t last_instance_handle**

  The instance handle of the last reliable remote reader to be determined inactive.

8.264.1 Detailed Description

<<***extension***>> *(*p. *155)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

This status is the reciprocal status to the **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1239) on the reader. It is different than the **com.rti.dds.publication.LivelinessLostStatus** (p. 1242) on the writer in that the latter informs the writer about its own liveliness; this status informs the writer about the "liveliness" (activity) of its matched readers.

All counts in this status will remain at zero for best effort writers.

8.264.2 Constructor & Destructor Documentation

8.264.2.1 ReliableReaderActivityChangedStatus() [1/2]

```
ReliableReaderActivityChangedStatus ( )
```

Construct a new status object with default contents.

8.264.2.2 ReliableReaderActivityChangedStatus() [2/2]

```
ReliableReaderActivityChangedStatus (
            ReliableReaderActivityChangedStatus src )
```

Construct a new status identical to the given status.

Exceptions

NullPointerException	if the given status is null.

References **ReliableReaderActivityChangedStatus.active_count**, **ReliableReaderActivityChangedStatus.active_count_change**, **ReliableReaderActivityChangedStatus.inactive_count**, **ReliableReaderActivityChangedStatus.inactive_count_change**, and **ReliableReaderActivityChangedStatus.last_instance_handle**.

8.264.3 Member Data Documentation

8.264.3.1 active_count

```
int active_count
```

The current number of reliable readers currently matched with this reliable writer.

Referenced by **ReliableReaderActivityChangedStatus.ReliableReaderActivityChangedStatus()**.

8.264.3.2 inactive_count

```
int inactive_count
```

The number of reliable readers that have been dropped by this reliable writer because they failed to send acknowledgements in a timely fashion.

A reader is considered to be inactive after is has been sent heartbeats **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_heartbeat_retries** (p. 1611) times, each heartbeat having been separated from the previous by the current heartbeat period.

Referenced by **ReliableReaderActivityChangedStatus.ReliableReaderActivityChangedStatus()**.

8.264.3.3 active_count_change

```
int active_count_change
```

The most recent change in the number of active remote reliable readers.

Referenced by **ReliableReaderActivityChangedStatus.ReliableReaderActivityChangedStatus()**.

8.264.3.4 inactive_count_change

```
int inactive_count_change
```

The most recent change in the number of inactive remote reliable readers.

Referenced by **ReliableReaderActivityChangedStatus.ReliableReaderActivityChangedStatus()**.

8.264.3.5 last_instance_handle

```
final  InstanceHandle_t last_instance_handle
```

The instance handle of the last reliable remote reader to be determined inactive.

Referenced by **ReliableReaderActivityChangedStatus.ReliableReaderActivityChangedStatus()**.

8.265 ReliableWriterCacheChangedStatus Class Reference

<<***extension***>> *(*p. *155)* A summary of the state of a data writer's cache of unacknowledged samples written.

Inherits Status.

Public Member Functions

- **ReliableWriterCacheChangedStatus** ()

  Construct a new status object.
- **ReliableWriterCacheChangedStatus** (**ReliableWriterCacheChangedStatus** src)

  A copy constructor.

Public Attributes

- final **ReliableWriterCacheEventCount empty_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has become empty.
- final **ReliableWriterCacheEventCount full_reliable_writer_cache** = new **ReliableWriterCacheEventCount**()

  The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.
- final **ReliableWriterCacheEventCount low_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.
- final **ReliableWriterCacheEventCount high_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.
- int **unacknowledged_sample_count**

  The current number of unacknowledged samples in the writer's cache.
- int **unacknowledged_sample_count_peak**

  The highest value that unacknowledged_sample_count has reached until now.
- long **replaced_unacknowledged_sample_count**

  The number of unacknowledged samples that have been replaced in the writer's cache.

8.265.1 Detailed Description

<<***extension***>> *(*p. *155)* A summary of the state of a data writer's cache of unacknowledged samples written.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

A written sample is unacknowledged (and therefore accounted for in this status) if the writer is reliable and one or more readers matched with the writer has not yet sent an acknowledgement to the writer declaring that it has received the sample.

If the low watermark is zero and the unacknowledged sample count decreases to zero, both the low watermark and cache empty events are considered to have taken place. A single callback will be dispatched (assuming the user has requested one) that contains both status changes. The same logic applies when the high watermark is set equal to the maximum number of samples and the cache becomes full.

8.265.2 Constructor & Destructor Documentation

8.265.2.1 ReliableWriterCacheChangedStatus() [1/2]

```
ReliableWriterCacheChangedStatus ( )
```

Construct a new status object.

8.265.2.2 ReliableWriterCacheChangedStatus() [2/2]

```
ReliableWriterCacheChangedStatus (
            ReliableWriterCacheChangedStatus src )
```

A copy constructor.

Parameters

src	Source to copy from.

Exceptions

NullPointerException	if the source object is null.

References **ReliableWriterCacheChangedStatus.empty_reliable_writer_cache**, **ReliableWriterCacheChanged↩ Status.full_reliable_writer_cache**, **ReliableWriterCacheChangedStatus.high_watermark_reliable_writer_cache**, **ReliableWriterCacheChangedStatus.low_watermark_reliable_writer_cache**, **ReliableWriterCacheChanged↩ Status.replaced_unacknowledged_sample_count**, **ReliableWriterCacheChangedStatus.unacknowledged_↩ sample_count**, and **ReliableWriterCacheChangedStatus.unacknowledged_sample_count_peak**.

8.265.3 Member Data Documentation

8.265.3.1 empty_reliable_writer_cache

```
final  ReliableWriterCacheEventCount empty_reliable_writer_cache
```

The number of times the reliable writer's cache of unacknowledged samples has become empty.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.2 full_reliable_writer_cache

```
final ReliableWriterCacheEventCount full_reliable_writer_cache = new ReliableWriterCacheEvent↩
Count()
```

The number of times the reliable writer's cache, or send window, of unacknowledged samples has become full.

Applies to writer's cache when the send window is enabled (when both **com.rti.dds.infrastructure.RtpsReliable↩ WriterProtocol_t.min_send_window_size** (p. 1616) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol↩ _t.max_send_window_size** (p. 1617) are not **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259)).

Otherwise, applies when the number of unacknowledged samples has reached the send window limit.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.3 low_watermark_reliable_writer_cache

```
final ReliableWriterCacheEventCount low_watermark_reliable_writer_cache
```

The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.

A low watermark event will only be considered to have taken place when the number of unacknowledged samples in the writer's cache *decreases* to this value. A sample count that increases to this value will not result in a callback or in a change to the total count of low watermark events.

When the writer's send window is enabled, the low watermark is scaled down, if necessary, to fit within the current send window.

Set the low watermark value using the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark** (p. 1607) field of the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605) structure.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.4 high_watermark_reliable_writer_cache

```
final ReliableWriterCacheEventCount high_watermark_reliable_writer_cache
```

The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.

A high watermark event will only be considered to have taken place when the number of unacknowledged sampled *increases* to this value. A sample count that was above this value and then decreases back to it will not trigger an event.

When the writer's send window is enabled, the high watermark is scaled down, if necessary, to fit within the current send window.

Set the high watermark value using the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1607) field of the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t** (p. 1605) structure.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.5 unacknowledged_sample_count

```
int unacknowledged_sample_count
```

The current number of unacknowledged samples in the writer's cache.

A sample is considered unacknowledged if the writer has failed to receive an acknowledgement from one or more reliable readers matched to it.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.6 unacknowledged_sample_count_peak

```
int unacknowledged_sample_count_peak
```

The highest value that unacknowledged_sample_count has reached until now.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.265.3.7 replaced_unacknowledged_sample_count

```
long replaced_unacknowledged_sample_count
```

The number of unacknowledged samples that have been replaced in the writer's cache.

Total number of unacknowledged samples that have been replaced by a **DataWriter** (p. 553) after applying com.rti.↩ dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS policy.

Referenced by **ReliableWriterCacheChangedStatus.ReliableWriterCacheChangedStatus()**.

8.266 ReliableWriterCacheEventCount Class Reference

<<***extension***>> *(p. 155)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

Inherits Struct.

Public Attributes

- int **total_count**

  The total number of times the event has occurred.
- int **total_count_change**

  The incremental number of times the event has occurred since the listener was last invoked or the status read.

8.266.1 Detailed Description

<<***extension***>> *(p. 155)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

The threshold interpretation depends on the usage of this data type in **com.rti.dds.publication.ReliableWriterCache↩ ChangedStatus** (p. 1535).

See also

com.rti.dds.publication.ReliableWriterCacheChangedStatus (p. 1535)

8.266.2 Member Data Documentation

8.266.2.1 total_count

```
int total_count
```

The total number of times the event has occurred.

8.266.2.2 total_count_change

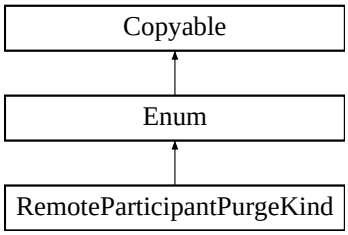
```
int total_count_change
```

The incremental number of times the event has occurred since the listener was last invoked or the status read.

8.267 RemoteParticipantPurgeKind Class Reference

Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

Inheritance diagram for RemoteParticipantPurgeKind:



Static Public Attributes

- static final **RemoteParticipantPurgeKind LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE**

  [default] *Maintain knowledge of the remote participant for as long as it maintains its liveliness contract.*
- static final **RemoteParticipantPurgeKind NO_REMOTE_PARTICIPANT_PURGE**

  Never "forget" a remote participant with which discovery communication has been lost.

Additional Inherited Members

8.267.1 Detailed Description

Available behaviors for halting communication with remote participants (and their contained entities) with which discovery communication has been lost.

When discovery communication with a remote participant has been lost, the local participant must make a decision about whether to continue attempting to communicate with that participant and its contained entities. This "kind" is used to select the desired behavior.

This "kind" does not pertain to the situation in which a remote participant has been gracefully deleted and notification of that deletion have been successfully received by its peers. In that case, the local participant will immediately stop attempting to communicate with those entities and will remove the associated remote entity records from its internal database.

See also

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.remote_participant_purge_kind (p. 649)

8.267.2 Member Data Documentation

8.267.2.1 LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE

```
final  RemoteParticipantPurgeKind LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE  [static]
```

Initial value:

```
= new RemoteParticipantPurgeKind(
                "LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE", 0)
```

[default] Maintain knowledge of the remote participant for as long as it maintains its liveliness contract.

A participant will continue attempting communication with its peers, even if discovery communication with them is lost, as long as the remote participants maintain their liveliness. If both discovery communication and participant liveliness are lost, however, the local participant will remove all records of the remote participant and its contained endpoints, and no further data communication with them will occur until and unless they are rediscovered.

The liveliness contract a participant promises to its peers – its "liveliness lease duration" – is specified in its **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_lease_duration** (p. 649) QoS field. It maintains that contract by writing data to those other participants with a writer that has a **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247) of com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS or com.rti.dds.infrastructure.LivelinessQosPolicyKind.LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS and by asserting itself (at the **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_liveliness_assert_period** (p. 649)) over the Simple Discovery Protocol.

8.267.2.2 NO_REMOTE_PARTICIPANT_PURGE

```
final  RemoteParticipantPurgeKind NO_REMOTE_PARTICIPANT_PURGE  [static]
```

Initial value:

```
= new RemoteParticipantPurgeKind(
                "NO_REMOTE_PARTICIPANT_PURGE", 1)
```

Never "forget" a remote participant with which discovery communication has been lost.

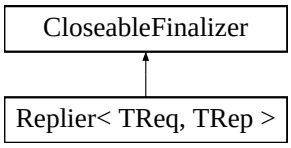
If a participant with this behavior loses discovery communication with a remote participant, it will nevertheless remember that remote participant and its endpoints and continue attempting to communicate with them indefinitely.

This value has consequences for a participant's resource usage. If discovery communication with a remote participant is lost, but the same participant is later rediscovered, any relevant records that remain in the database will be reused. However, if it is not rediscovered, the records will continue to take up space in the database for as long as the local participant remains in existence.

8.268 Replier< TReq, TRep > Class Template Reference

Allows receiving requests and sending replies.

Inheritance diagram for Replier< TReq, TRep >:



Public Member Functions

- **Replier** (**ReplierParams**< TReq, TRep > params)

  *Creates a **Replier** (p. 1542) with parameters.*
- **Replier** (**DomainParticipant** participant, String serviceName, **TypeSupport** requestTypeSupport, **Type↩ Support** replyTypeSupport)

  *Creates a **Replier** (p. 1542) with the minimum set of parameters.*
- void **close** ()

  *Releases the internal entities created by this **Replier** (p. 1542).*
- void **sendReply** (TRep reply, **SampleIdentity_t** relatedRequestId)

  Sends a reply for a previous request.
- void **sendReply** (**WriteSample**< TRep > reply, **SampleIdentity_t** relatedRequestId)

  Sends a reply for a previous request.
- boolean **receiveRequest** (**Sample**< TReq > request, **Duration_t** maxWait)

  Waits for a request and copies its contents into a Sample.
- List< **Sample**< TReq > > **receiveRequests** (List< **Sample**< TReq > > replies, int maxCount, **Duration_t** maxWait)

  Waits for multiple requests and copies them into a list.

- List< **Sample**< TReq > > **receiveRequests** (List< **Sample**< TReq > > replies, int minCount, int maxCount, **Duration_t** maxWait)

  Waits for multiple requests and copies them into a list.
- Sample.Iterator< TReq > **receiveRequests** (**Duration_t** maxWait)

  Waits for multiple requests and provides a loaned iterator to access them.
- Sample.Iterator< TReq > **receiveRequests** (int minCount, int maxCount, **Duration_t** maxWait)

  Waits for multiple requests and provides a loaned iterator to access them.
- boolean **takeRequest** (**Sample**< TReq > request)

  Copies the contents of a request into a Sample.
- Sample.Iterator< TReq > **takeRequests** ()

  Provides a loaned iterator to access the existing requests.
- Sample.Iterator< TReq > **takeRequests** (int maxCount)

  Provides a loaned iterator to access the existing requests.
- List< **Sample**< TReq > > **takeRequests** (List< **Sample**< TReq > > requests, int maxCount)

  Copies existing requests into a list.
- boolean **readRequest** (**Sample**< TReq > request)

  Copies the contents of a request into a Sample.
- Sample.Iterator< TReq > **readRequests** ()

  Provides a loaned iterator to access the existing requests.
- Sample.Iterator< TReq > **readRequests** (int maxCount)

  Provides a loaned iterator to access the existing requests.
- List< **Sample**< TReq > > **readRequests** (List< **Sample**< TReq > > requests, int maxCount)

  Copies existing requests into a list.
- boolean **waitForRequests** (**Duration_t** maxWait)

  Waits for requests.
- boolean **waitForRequests** (int minCount, **Duration_t** maxWait)

  Waits for requests.
- **DataWriter getReplyDataWriter** ()

  *Retrieves the underlying **com.rti.dds.publication.DataWriter** (p. 553).*
- **DataReader getRequestDataReader** ()

  *Retrieves the underlying **com.rti.dds.subscription.DataReader** (p. 450).*
- **WriteSample**< TRep > **createReplySample** ()

  Creates a WriteSample for sending replies.
- **WriteSample**< TRep > **createReplySample** (TRep data)

  Creates a WriteSample for sending replies.
- **Sample**< TReq > **createRequestSample** ()

  Creates a Sample for receiving replies.
- **Sample**< TReq > **createRequestSample** (TReq data)

  Creates a Sample for receiving replies.

8.268.1 Detailed Description

Allows receiving requests and sending replies.

A **Replier** (p. 1542) is an entity with two associated **topics** (p. 54): a request topic and a reply topic. It can receive requests by subscribing to the request topic and can send replies to those requests by publishing the reply topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 61), and **com.rti.dds.dynamicdata.DynamicData** (p. 847). See **Creating a Replier** (p. 151).

A **Replier** (p. 1542) has four main types of operations:

- Waiting for requests to be received from the middleware
- Getting those requests
- Receiving requests (a convenience operation that is a combination of waiting and getting in a single operation)
- Sending a reply for a previously received request (i.e., publishing a reply sample on the reply topic with special meta-data so that the orignal **Requester** (p. 1563) can identify it)

For multi-reply scenarios in which a com.rti.connext.requestreply.Replier<TReq,TRep> generates more than one reply for a request, the com.rti.connext.requestreply.Replier<TReq,TRep> should mark all the intermediate replies (all but the last) using the **com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1631) flag in **com.rti.dds.infrastructure.WriteParams_t.flag** (p. 1999).

Much like a **Requester** (p. 1563), a **Replier** (p. 1542) has an associated **com.rti.dds.domain.DomainParticipant** (p. 670), which can be shared with other Repliers or RTI Connext routines. All the other entities required for the request-reply interaction, including a **com.rti.dds.publication.DataWriter** (p. 553) for writing replies and a **com.rti.↩ dds.subscription.DataReader** (p. 450) for reading requests, are automatically created when the **Replier** (p. 1542) is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **com.rti.connext.↩ requestreply.RequesterParams.setQosProfile(String,String)** (p. 1588)). By default, they are created with **com.rti.↩ dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 153)

There are several ways to use a **Replier** (p. 1542):

- A thread **receives** (p. 1549) requests and then dispatches them. If the computation of a reply is a simple operation, consider using a com.rti.connext.requestreply.SimpleReplier<TReq,TRep> instead of a **Replier** (p. 1542).
- Polling without waiting, using **takeRequests(int)** (p. 1550) directly.
- Using a com.rti.connext.requestreply.ReplierListener<TReq,TRep> to get notified and **get** (p. 1550) the requests within the callback.

Template Parameters

TReq	The data type for the request topic
TRep	The data type for the reply topic

See also

com.rti.connext.requestreply.Requester<TReq,TRep>

Request-Reply Examples (p. 146)

Basic Replier example (p. 152)

8.268.2 Constructor & Destructor Documentation

8.268.2.1 Replier() [1/2]

```
Replier (
            ReplierParams< TReq, TRep > params )
```

Creates a **Replier** (p. 1542) with parameters.

Parameters

params	All the parameters that configure this **Replier** (p. 1542)

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.connext.requestreply.ReplierParams<TReq,TRep>

Creating a Replier (p. 151)

8.268.2.2 Replier() [2/2]

```
Replier (
             DomainParticipant participant,
            String serviceName,
             TypeSupport requestTypeSupport,
             TypeSupport replyTypeSupport )
```

Creates a **Replier** (p. 1542) with the minimum set of parameters.

Parameters

participant	The DomainParticipant that this **Replier** (p. 1542) uses to join a domain.
serviceName	The service name. See **com.rti.connext.requestreply.ReplierParams<TReq,TRep>.setServiceName(String)** (p. 1557)
requestTypeSupport	The type support for type TReq
replyTypeSupport	The type support for type TReq

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **ReplierParams< TReq, TRep >.setServiceName()**.

8.268.3 Member Function Documentation

8.268.3.1 close()

```
void close ( )
```

Releases the internal entities created by this **Replier** (p. 1542).

Among other internal resources, it deletes the **Replier** (p. 1542)'s underlying DataReader and DataWriter.

See also

com.rti.dds.subscription.Subscriber.delete_datareader (p. 1739)

com.rti.dds.publication.Publisher.delete_datawriter (p. 1474)

Referenced by **SimpleReplier< TReq, TRep >.close()**.

8.268.3.2 sendReply() [1/2]

```
void sendReply (
             TRep reply,
              SampleIdentity_t relatedRequestId )
```

Sends a reply for a previous request.

The related request identity can be retrieved from an existing request sample (com.rti.connext.infrastructure.Sample< T>).

Parameters

reply	The reply to be sent.
relatedRequestId	The identity of a previously received request

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.connext.infrastructure.SampleData<T>.getIdentity() (p. 1630)

receiveRequest(Sample<TReq>,Duration_t) (p. 1547)

receiveRequests(int,int,Duration_t) (p. 1549)

takeRequest(Sample<TReq>) (p. 1549)

takeRequests(int) (p. 1550)

Basic Replier example (p. 152)

8.268.3.3 sendReply() [2/2]

```
void sendReply (
            WriteSample< TRep > reply,
            SampleIdentity_t relatedRequestId )
```

Sends a reply for a previous request.

Allows you to set custom parameters for writing a reply.

Contrary to the com.rti.connext.requestreply.Requester<TReq,TRep>, where retrieving the sample identity for correlation is common, on the **Replier** (p. 1542) side using a WriteSample is only necessary when the default write parameters need to be overridden, and **sendReply(TRep, SampleIdentity_t)** (p. 1546) may be used if that is not necessary.

One reason to override the default write parameters is a multi-reply scenario in which a com.rti.connext.requestreply.↩ Replier<TReq,TRep> generates more than one reply for a request. In this case, all the intermediate replies (all but the last) should be marked with the **com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_REPLY_↩ SEQUENCE_SAMPLE** (p. 1631) flag in **com.rti.dds.infrastructure.WriteParams_t.flag** (p. 1999) within **com.rti.↩ connext.infrastructure.WriteSample<T>.getInfo()** (p. 2009).

A com.rti.connext.requestreply.Requester<TReq,TRep> can detect if a reply is the last reply of a sequence of replies by checking that the flag **com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1631) is not set in **com.rti.dds.subscription.SampleInfo.flag** (p. 1645) within **com.rti.connext.infrastructure.↩ Sample<T>.getInfo()** (p. 1628).

See also

sendReply(TRep, SampleIdentity_t) (p. 1546)

com.rti.connext.infrastructure.WriteSample<T>

References **SampleData< T >.getData()**, and **WriteSample< T >.getInfo()**.

8.268.3.4 receiveRequest()

```
boolean receiveRequest (
              Sample< TReq > request,
              Duration_t maxWait )
```

Waits for a request and copies its contents into a Sample.

Equivalent to using **waitForRequests(int,Duration_t)** (p. 1552) and **takeRequest(Sample<TReq>)** (p. 1549)

See also

com.rti.connext.infrastructure.Sample<T>

waitForRequests(int,Duration_t) (p. 1552)

takeRequest(Sample<TReq>) (p. 1549)

Basic Replier example (p. 152)

8.268.3.5 receiveRequests() [1/4]

```
List<  Sample< TReq > > receiveRequests (
             List<  Sample< TReq > > replies,
             int maxCount,
              Duration_t maxWait )
```

Waits for multiple requests and copies them into a list.

Equivalent to **receiveRequests(List<Sample<TReq>>,int,int,Duration_t)** (p. 1548) with `min_count = 0`

See also

com.rti.connext.infrastructure.Sample<T>

receiveRequests(List<Sample<TReq>>,int,Duration_t) (p. 1548)

References **Replier< TReq, TRep >.receiveRequests()**.

Referenced by **Replier< TReq, TRep >.receiveRequests()**.

8.268.3.6 receiveRequests() [2/4]

```
List<  Sample< TReq > > receiveRequests (
            List<  Sample< TReq > > replies,
            int minCount,
            int maxCount,
             Duration_t maxWait )
```

Waits for multiple requests and copies them into a list.

Equivalent to using **waitForRequests(int,Duration_t)** (p. 1552) and **takeRequests(List<Sample<TReq>>,int)** (p. 1550)

See also

com.rti.connext.infrastructure.Sample<T>

waitForRequests(int,Duration_t) (p. 1552)

takeRequests(List<Sample<TReq>>,int) (p. 1550)

8.268.3.7 receiveRequests() [3/4]

```
Sample.Iterator< TReq > receiveRequests (
             Duration_t maxWait )
```

Waits for multiple requests and provides a loaned iterator to access them.

Equivalent to using **receiveRequests(int,int,Duration_t)** (p. 1549) with `min_count=1` and `max_count=`**`com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED`** (p. 259)

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

receiveRequests(int,int,Duration_t) (p. 1549)

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**, and **Replier< TReq, TRep >.receiveRequests()**.

8.268.3.8 receiveRequests() [4/4]

```
Sample.Iterator< TReq > receiveRequests (
            int minCount,
            int maxCount,
             Duration_t maxWait )
```

Waits for multiple requests and provides a loaned iterator to access them.

Equivalent to using **waitForRequests(int,Duration_t)** (p. 1552) and **takeRequests(int)** (p. 1550)

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

waitForRequests(int,Duration_t) (p. 1552)

takeRequests(int) (p. 1550)

8.268.3.9 takeRequest()

```
boolean takeRequest (
              Sample< TReq > request )
```

Copies the contents of a request into a Sample.

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReply(Sample<TRep>) (p. 1572)

8.268.3.10 takeRequests() [1/3]

```
Sample.Iterator< TReq > takeRequests ( )
```

Provides a loaned iterator to access the existing requests.

Equivalent to using **takeRequests(int)** (p. 1550) with max_count=**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies() (p. 1572)

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**.

8.268.3.11 takeRequests() [2/3]

```
Sample.Iterator< TReq > takeRequests (
             int maxCount )
```

Provides a loaned iterator to access the existing requests.

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(int) (p. 1573)

8.268.3.12 takeRequests() [3/3]

```
List<  Sample< TReq > > takeRequests (
            List<  Sample< TReq > > requests,
            int maxCount )
```

Copies existing requests into a list.

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(List<Sample<TRep>>,int) (p. 1574)

8.268.3.13 readRequest()

```
boolean readRequest (
             Sample< TReq > request )
```

Copies the contents of a request into a Sample.

This operation is equivalent to **com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequest(Sample<TReq>)** (p. 1549) except the request remains in the **Replier** (p. 1542) and can be read or taken again.

8.268.3.14 readRequests() [1/3]

```
Sample.Iterator< TReq > readRequests ( )
```

Provides a loaned iterator to access the existing requests.

This operation is equivalent to **com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequests()** (p. 1550) except the requests remain in the **Replier** (p. 1542) and can be read or taken again.

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**.

8.268.3.15 readRequests() [2/3]

```
Sample.Iterator< TReq > readRequests (
            int maxCount )
```

Provides a loaned iterator to access the existing requests.

This operation is equivalent to **com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequests(int)** (p. 1550) except the requests remain in the **Replier** (p. 1542) and can be read or taken again.

8.268.3.16 readRequests() [3/3]

```
List< Sample< TReq > > readRequests (
            List< Sample< TReq > > requests,
            int maxCount )
```

Copies existing requests into a list.

This operation is equivalent to **com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequests(List<↩ Sample<TReq>>,int)** (p. 1550) except the requests remain in the **Replier** (p. 1542) and can be read or taken again.

8.268.3.17 waitForRequests() [1/2]

```
boolean waitForRequests (
             Duration_t maxWait )
```

Waits for requests.

Equivalent to **waitForRequests(int,Duration_t)** (p. 1552) with `min_count=1`

See also

waitForRequests(int,Duration_t) (p. 1552)

References **Replier< TReq, TRep >.waitForRequests()**.

Referenced by **Replier< TReq, TRep >.waitForRequests()**.

8.268.3.18 waitForRequests() [2/2]

```
boolean waitForRequests (
            int minCount,
             Duration_t maxWait )
```

Waits for requests.

This operation waits for minCount requests to be available. It will wait up to maxWait .

This operation is similar to **com.rti.connext.requestreply.Requester<TReq,TRep>.waitForReplies(int,Duration_t)** (p. 1580).

Parameters

minCount	Minimum number of requests that need to be available for this operation to unblock.
maxWait	Maximum waiting time after which this operation unblocks regardless of how many requests are available.

Returns

true if at least minCount requests were available before maxWait elapsed, or false otherwise.

See also

takeRequests(int) (p. 1550)

com.rti.connext.requestreply.Requester<TReq,TRep>.waitForReplies(int,Duration_t) (p. 1580)

8.268.3.19 getReplyDataWriter()

```
DataWriter getReplyDataWriter ( )
```

Retrieves the underlying **com.rti.dds.publication.DataWriter** (p. 553).

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.getRequestDataWriter() (p. 1582)

8.268.3.20 getRequestDataReader()

```
DataReader getRequestDataReader ( )
```

Retrieves the underlying **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.getReplyDataReader() (p. 1583)

8.268.3.21 createReplySample() [1/2]

```
WriteSample< TRep > createReplySample ( )
```

Creates a WriteSample for sending replies.

Returns

A new WriteSample with data of type `TReq` initialized with default values and default write parameters.

See also

com.rti.connext.infrastructure.WriteSample<T>

sendReply(WriteSample<TRep>, SampleIdentity_t) (p. 1547)

8.268.3.22 createReplySample() [2/2]

```
WriteSample< TRep > createReplySample (
            TRep data )
```

Creates a WriteSample for sending replies.

Returns

A new WriteSample with the specified data and default write parameters.

See also

createReplySample() (p. 1553)

8.268.3.23 createRequestSample() [1/2]

```
Sample< TReq > createRequestSample ( )
```

Creates a Sample for receiving replies.

Returns

A new Sample with default data of type `TReq` and invalid SampleInfo.

See also

com.rti.connext.infrastructure.Sample<T>
receiveRequest(Sample<TReq>,Duration_t) (p. 1547)
takeRequest(Sample<TReq>) (p. 1549)

8.268.3.24 createRequestSample() [2/2]

```
Sample< TReq > createRequestSample (
            TReq data )
```

Creates a Sample for receiving replies.

Returns

A new Sample with default data of type `TReq` and invalid SampleInfo.

See also

com.rti.connext.infrastructure.Sample<T>
receiveRequest(Sample<TReq>,Duration_t) (p. 1547)
takeRequest(Sample<TReq>) (p. 1549)

8.269 ReplierListener< TReq, TRep > Interface Template Reference

Called when a com.rti.connext.requestreply.Replier<TReq,TRep> has new available requests.

Inherits EventListener.

Public Member Functions

- void **onRequestAvailable** (**Replier**< TReq, TRep > replier)
  User callback.

8.269.1 Detailed Description

Called when a com.rti.connext.requestreply.Replier<TReq,TRep> has new available requests.

A replier listener is a way to implement a callback that will be invoked when requests are available. It is an optional com.rti.connext.requestreply.ReplierParams<TReq,TRep>.

This listener simply notifies when requests are available. The callback implementation can then use **com.rti.connext.requestreply.Replier<TReq,TRep>.takeRequests(int)** (p. 1550) to retrieve them.

A simpler callback mechanism, where one request sample is a parameter and the reply is the callback return value, is implemented by the com.rti.connext.requestreply.SimpleReplier<TReq,TRep>.

See also

com.rti.connext.requestreply.Replier<TReq,TRep>.Replier(ReplierParams)

8.269.2 Member Function Documentation

8.269.2.1 onRequestAvailable()

```
void onRequestAvailable (
              Replier< TReq, TRep > replier )
```

User callback.

See also

com.rti.dds.subscription.DataReaderListener.on_data_available (p. 500)

8.270 ReplierParams< TReq, TRep > Class Template Reference

Contains the parameters for creating a com.rti.connext.requestreply.Replier<TReq,TRep>.

Inherits EntityParams.

Public Member Functions

- **ReplierParams** (**DomainParticipant** participant, **TypeSupport** requestTypeSupport, **TypeSupport** reply↩ TypeSupport)

  *Creates a **ReplierParams** (p. 1556) with the parameters that a **Replier** (p. 1542) always needs.*
- **ReplierParams**< TReq, TRep > **setServiceName** (String serviceName)

  *The service name the **Replier** (p. 1542) offers and Requesters use to match.*
- **ReplierParams**< TReq, TRep > **setRequestTopicName** (String requestTopicName)

  Sets a specific request topic name.
- **ReplierParams**< TReq, TRep > **setReplyTopicName** (String replyTopicName)

  Sets a specific reply topic name.
- **ReplierParams**< TReq, TRep > **setDataWriterQos** (**DataWriterQos** dataWriterQos)

  Sets the quality of service of the reply DataWriter.
- **ReplierParams**< TReq, TRep > **setDataReaderQos** (**DataReaderQos** dataReaderQos)

  Sets the quality of service of the request DataReader.
- **ReplierParams**< TReq, TRep > **setQosProfile** (String qosLibraryName, String qosProfileName)

  Sets a QoS profile for the entities in this replier.
- **ReplierParams**< TReq, TRep > **setPublisher** (**Publisher** publisher)

  Sets a specific Publisher.
- **ReplierParams**< TReq, TRep > **setSubscriber** (**Subscriber** subscriber)

  Sets a specific Subscriber.
- **ReplierParams**< TReq, TRep > **setReplierListener** (**ReplierListener**< TReq, TRep > replierListener)

  Sets a listener that is called when requests are available.

8.270.1 Detailed Description

Contains the parameters for creating a com.rti.connext.requestreply.Replier<TReq,TRep>.

See also

com.rti.connext.requestreply.RequesterParams (p. 1586)

8.270.2 Constructor & Destructor Documentation

8.270.2.1 ReplierParams()

```
ReplierParams (
            DomainParticipant participant,
            TypeSupport requestTypeSupport,
            TypeSupport replyTypeSupport )
```

Creates a **ReplierParams** (p. 1556) with the parameters that a **Replier** (p. 1542) always needs.

In addition to the parameters this constructor takes , a **Replier** (p. 1542) needs either:

- A service name (**setServiceName(String)** (p. 1557)), or
- Custom topic names (**setReplyTopicName(String)** (p. 1558) and **setRequestTopicName(String)** (p. 1557)).

The other parameters are optional.

Parameters

participant	The DomainParticipant that this replier uses to join a domain.
requestTypeSupport	The type support for type `TReq`
replyTypeSupport	The type support for type `Trep`

8.270.3 Member Function Documentation

8.270.3.1 setServiceName()

```
ReplierParams< TReq, TRep > setServiceName (
            String serviceName )
```

The service name the **Replier** (p. 1542) offers and Requesters use to match.

See also

com.rti.connext.requestreply.RequesterParams.setServiceName(String) (p. 1587)

Referenced by **Replier< TReq, TRep >.Replier()**.

8.270.3.2 setRequestTopicName()

```
ReplierParams< TReq, TRep > setRequestTopicName (
            String requestTopicName )
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **setService↩ Name(String)** (p. 1557). If that topic already exists, it will be reused. This is useful to have the **Replier** (p. 1542) use a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) to receive only certain requests.

8.270.3.3 setReplyTopicName()

```
ReplierParams< TReq, TRep > setReplyTopicName (
            String replyTopicName )
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on c the **setService↩ Name(String)** (p. 1557). If that topic already exists, it will be reused.

8.270.3.4 setDataWriterQos()

```
ReplierParams< TReq, TRep > setDataWriterQos (
             DataWriterQos dataWriterQos )
```

Sets the quality of service of the reply DataWriter.

See also

setQosProfile(String,String) (p. 1558)

8.270.3.5 setDataReaderQos()

```
ReplierParams< TReq, TRep > setDataReaderQos (
             DataReaderQos dataReaderQos )
```

Sets the quality of service of the request DataReader.

See also

setQosProfile(String,String) (p. 1558)

8.270.3.6 setQosProfile()

```
ReplierParams< TReq, TRep > setQosProfile (
            String qosLibraryName,
            String qosProfileName )
```

Sets a QoS profile for the entities in this replier.

Specifies the XML QoS profile that will be used to configure the quality of service for the **Replier** (p. 1542)'s underlying reply DataWriter and request DataReader.

Parameters

qosLibraryName	The name of the QoS library
qosProfileName	The name of the QoS profile inside the QoS library

See also

com.rti.connext.requestreply.RequesterParams.setQosProfile(String,String) (p. 1588)

Configuring Request-Reply QoS profiles (p. 153)

Configuring QoS Profiles with XML (p. 120)

8.270.3.7 setPublisher()

```
ReplierParams< TReq, TRep > setPublisher (
            Publisher publisher )
```

Sets a specific Publisher.

See also

com.rti.connext.requestreply.RequesterParams.setPublisher(Publisher) (p. 1589)

8.270.3.8 setSubscriber()

```
ReplierParams< TReq, TRep > setSubscriber (
            Subscriber subscriber )
```

Sets a specific Subscriber.

See also

com.rti.connext.requestreply.RequesterParams.setSubscriber(Subscriber) (p. 1589)

8.270.3.9 setReplierListener()

```
ReplierParams< TReq, TRep > setReplierListener (
            ReplierListener< TReq, TRep > replierListener )
```

Sets a listener that is called when requests are available.

See also

com.rti.connext.requestreply.ReplierListener<TReq,TRep>

Referenced by **SimpleReplier< TReq, TRep >.SimpleReplier()**.

8.271 RequestedDeadlineMissedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of the deadlines detected for any instance read by the **com.rti.dds.subscription.DataReader** (p. 450).*
- int **total_count_change**

  The incremental number of deadlines detected since the last time the listener was called or the status was read.
- final **InstanceHandle_t last_instance_handle**

  *Handle to the last instance in the **com.rti.dds.subscription.DataReader** (p. 450) for which a deadline was detected.*

8.271.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS

8.271.2 Member Data Documentation

8.271.2.1 total_count

```
int total_count
```

Total cumulative count of the deadlines detected for any instance read by the **com.rti.dds.subscription.DataReader** (p. 450).

8.271.2.2 total_count_change

```
int total_count_change
```

The incremental number of deadlines detected since the last time the listener was called or the status was read.

8.271.2.3 last_instance_handle

```
final InstanceHandle_t last_instance_handle
```

Handle to the last instance in the **com.rti.dds.subscription.DataReader** (p. 450) for which a deadline was detected.

8.272 RequestedIncompatibleQosStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of how many times the concerned **com.rti.dds.subscription.DataReader** (p. 450) discovered a **com.rti.dds.publication.DataWriter** (p. 553) for the same **com.rti.dds.topic.Topic** (p. 1807) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 450).*
- int **total_count_change**

  The change in `total_count` since the last time the listener was called or the status was read.
- **QosPolicyId_t last_policy_id**

  *The **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504) of one of the policies that was found to be incompatible the last time an incompatibility was detected.*
- final **QosPolicyCountSeq policies**

  *A list containing, for each policy, the total number of times that the concerned **com.rti.dds.subscription.DataReader** (p. 450) discovered a **com.rti.dds.publication.DataWriter** (p. 553) for the same **com.rti.dds.topic.Topic** (p. 1807) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 450).*

8.272.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS

See also

DURABILITY (p. 230)
PRESENTATION (p. 247)
RELIABILITY (p. 258)
OWNERSHIP (p. 244)
LIVELINESS (p. 239)
DEADLINE (p. 217)
LATENCY_BUDGET (p. 238)
DESTINATION_ORDER (p. 218)

8.272.2 Member Data Documentation

8.272.2.1 total_count

```
int total_count
```

Total cumulative count of how many times the concerned **com.rti.dds.subscription.DataReader** (p. 450) discovered a **com.rti.dds.publication.DataWriter** (p. 553) for the same **com.rti.dds.topic.Topic** (p. 1807) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 450).

8.272.2.2 total_count_change

```
int total_count_change
```

The change in `total_count` since the last time the listener was called or the status was read.

8.272.2.3 last_policy_id

```
QosPolicyId_t last_policy_id
```

The **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1504) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.272.2.4 policies

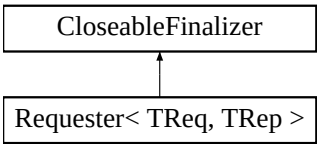
```
final  QosPolicyCountSeq policies
```

A list containing, for each policy, the total number of times that the concerned **com.rti.dds.subscription.DataReader** (p. 450) discovered a **com.rti.dds.publication.DataWriter** (p. 553) for the same **com.rti.dds.topic.Topic** (p. 1807) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 450).

8.273 Requester< TReq, TRep > Class Template Reference

Allows sending requests and receiving replies.

Inheritance diagram for Requester< TReq, TRep >:



Public Member Functions

- **Requester** (**RequesterParams** params)

  *Creates a **Requester** (p. 1563) with parameters.*
- **Requester** (**DomainParticipant** participant, String serviceName, **TypeSupport** requestTypeSupport, **Type↩ Support** replyTypeSupport)

  *Creates a **Requester** (p. 1563) with the minimum set of parameters.*
- void **close** ()

  Releases the internal entities created by this object.
- void **sendRequest** (TReq request)

  Sends a request.
- void **sendRequest** (**WriteSample**< TReq > request)

  Sends a request and gets back information about it that allows correlation with future replies.
- boolean **receiveReply** (**Sample**< TRep > reply, **Duration_t** maxWait)

  Waits for a reply and copies its contents into a Sample.
- List< **Sample**< TRep > > **receiveReplies** (List< **Sample**< TRep > > replies, int maxCount, **Duration_t** maxWait)

  Waits for multiple replies and copies them into a list.
- List< **Sample**< TRep > > **receiveReplies** (List< **Sample**< TRep > > replies, int minCount, int maxCount, **Duration_t** maxWait)

  Waits for multiple replies and copies them into a list.
- Sample.Iterator< TRep > **receiveReplies** (**Duration_t** maxWait)

  Waits for multiple replies and provides a loaned iterator to access them.
- Sample.Iterator< TRep > **receiveReplies** (int minCount, int maxCount, **Duration_t** maxWait)

  Waits for multiple replies and provides a loaned iterator to access them.
- boolean **takeReply** (**Sample**< TRep > reply)

  Copies the contents of a reply into a Sample.
- Sample.Iterator< TRep > **takeReplies** ()

  Provides a loaned iterator to access the existing replies.
- Sample.Iterator< TRep > **takeReplies** (int maxCount)

  Provides a loaned iterator to access the existing replies.
- List< **Sample**< TRep > > **takeReplies** (List< **Sample**< TRep > > replies, int maxCount)

  Copies existing replies into a list.
- boolean **takeReply** (**Sample**< TRep > reply, **SampleIdentity_t** relatedRequestId)

  Copies the contents of a reply for a specific request.

- Sample.Iterator< TRep > **takeReplies** (**SampleIdentity_t** relatedRequestId)

  Provides a loaned iterator to access the existing replies for a specific request.
- Sample.Iterator< TRep > **takeReplies** (int maxCount, **SampleIdentity_t** relatedRequestId)

  Provides a loaned iterator to access the existing replies for a specific request.
- List< **Sample**< TRep > > **takeReplies** (List< **Sample**< TRep > > replies, int maxCount, **SampleIdentity_t** relatedRequestId)

  Copies existing replies for a specific request into a list.
- boolean **readReply** (**Sample**< TRep > reply)

  Copies the contents of a reply into a Sample.
- Sample.Iterator< TRep > **readReplies** ()

  Provides a loaned iterator to access the existing replies.
- Sample.Iterator< TRep > **readReplies** (int maxCount)

  Provides a loaned iterator to access the existing replies.
- List< **Sample**< TRep > > **readReplies** (List< **Sample**< TRep > > replies, int maxCount)

  Copies existing replies into a list.
- boolean **readReply** (**Sample**< TRep > reply, **SampleIdentity_t** relatedRequestId)

  Copies the contents of a reply for a specific request.
- Sample.Iterator< TRep > **readReplies** (**SampleIdentity_t** relatedRequestId)

  Provides a loaned iterator to access the existing replies for a specific request.
- Sample.Iterator< TRep > **readReplies** (int maxCount, **SampleIdentity_t** relatedRequestId)

  Provides a loaned iterator to access the existing replies for a specific request.
- List< **Sample**< TRep > > **readReplies** (List< **Sample**< TRep > > replies, int maxCount, **SampleIdentity_t** relatedRequestId)

  Copies existing replies for a specific request into a list.
- boolean **waitForReplies** (**Duration_t** maxWait)

  Waits for replies to any request.
- boolean **waitForReplies** (int minCount, **Duration_t** maxWait)

  Waits for replies to any request.
- boolean **waitForReplies** (int minCount, **Duration_t** maxWait, **SampleIdentity_t** relatedRequestId)

  Waits for replies to a specific request.
- **DataWriter getRequestDataWriter** ()

  *Retrieves the underlying **com.rti.dds.publication.DataWriter** (p. 553).*
- **DataReader getReplyDataReader** ()

  *Retrieves the underlying **com.rti.dds.subscription.DataReader** (p. 450).*
- **WriteSample**< TReq > **createRequestSample** ()

  Creates a WriteSample for sending requests.
- **WriteSample**< TReq > **createRequestSample** (TReq data)

  Creates a WriteSample for sending requests.
- **Sample**< TRep > **createReplySample** ()

  Creates a Sample for getting replies.
- **Sample**< TRep > **createReplySample** (TRep data)

  Creates a Sample for getting replies.

8.273.1 Detailed Description

Allows sending requests and receiving replies.

A requester is an entity with two associated **topics** (p. 54): a request topic and a reply topic. It can send requests by publishing samples of the request topic and receive replies to those requests by subscribing to the reply topic.

Valid types for these topics (`TReq` and `TRep`) are: those generated by rtiddsgen, the **DDS built-in types** (p. 61), and **com.rti.dds.dynamicdata.DynamicData** (p. 847).

For example:

```
Requester<Foo, Bar> requester;

Requester<DynamicData, Bar> requester;

Requester<Foo, Octets> requester;
```

A **Replier** (p. 1542) and a **Requester** (p. 1563) communicate when they use the same topics for requests and replies (see **com.rti.connext.requestreply.RequesterParams.setServiceName(String)** (p. 1587)) on the same domain ID.

A **Requester** (p. 1563) can send requests and receive one or multiple replies. It does that using the following operations:

- Sending requests (i.e. publishing request samples on the request topic)
- Waiting for replies to be received by the middleware (for any request or for a specific request)
- Getting those replies from the middleware. There are two ways to do this: take (the data samples are removed from the middleware), read (the data samples remain in the middleware and can be read or taken again).
- A convenience operation, receive (which is a combination of wait and take).

In all cases, the middleware guarantees that a requester only receives reply samples that are associated with those requests that it sends.

For multi-reply scenarios, in which a **Requester** (p. 1563) receives multiple replies from a **Replier** (p. 1542) for a given request, the **Requester** (p. 1563) can check if a reply is the last reply of a sequence of replies. To do so, see if the bit **com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1631) is set in **com.rti.dds.subscription.SampleInfo.flag** (p. 1645) after receiving each reply. This indicates it is NOT the last reply.

A requester has an associated **com.rti.dds.domain.DomainParticipant** (p. 670), which can be shared with other requesters or RTI Connext routines. All the other RTI Connext entities required for the request-reply interaction, including a **com.rti.dds.publication.DataWriter** (p. 553) for writing requests and a **com.rti.dds.subscription.DataReader** (p. 450) for reading replies, are automatically created when the requester is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **com.rti.connext.↩ requestreply.RequesterParams.setQosProfile(String,String)** (p. 1588)). By default, they are created with **com.rti.↩ dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 153)

Template Parameters

TReq	The data type for the request topic
TRep	The data type for the reply topic

See also

com.rti.connext.requestreply.Replier<TReq,TRep>

Request-Reply Examples (p. 146)

Basic Requester example (p. 148)

8.273.2 Constructor & Destructor Documentation

8.273.2.1 Requester() [1/2]

```
Requester (
            RequesterParams params )
```

Creates a **Requester** (p. 1563) with parameters.

Parameters

params	All the parameters that configure this requester. See **com.rti.connext.requestreply.RequesterParams** (p. 1586) for the list of mandatory parameters.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.connext.requestreply.RequesterParams (p. 1586)

Creating a Requester (p. 147)

8.273.2.2 Requester() [2/2]

```
Requester (
            DomainParticipant participant,
           String serviceName,
            TypeSupport requestTypeSupport,
            TypeSupport replyTypeSupport )
```

Creates a **Requester** (p. 1563) with the minimum set of parameters.

Parameters

participant	The DomainParticipant this requester uses to join a DDS domain
serviceName	The service name. See **com.rti.connext.requestreply.RequesterParams.setServiceName(String)** (p. 1587)
requestTypeSupport	The type support for type `TReq`
replyTypeSupport	The type support for type `TReq`

Exceptions

One	of the **Standard Return Codes** (p. 261)

References **RequesterParams.setServiceName()**.

8.273.3 Member Function Documentation

8.273.3.1 close()

```
void close ( )
```

Releases the internal entities created by this object.

Among other internal resources, it deletes the underlying DataReader and DataWriter.

See also

com.rti.dds.subscription.Subscriber.delete_datareader (p. 1739)

com.rti.dds.publication.Publisher.delete_datawriter (p. 1474)

8.273.3.2 sendRequest() [1/2]

```
void sendRequest (
            TReq request )
```

Sends a request.

If a future reply needs to be correlated to exactly this request, use **com.rti.connext.requestreply.Requester<TReq,↩ TRep>.sendRequest(WriteSample<TReq>)** (p. 1568).

Parameters

request	The request to be sent

Exceptions

One	of the **Standard Return Codes** (p. 261)

MT Safety:

SAFE

See also

sendRequest(WriteSample<TReq>) (p. 1568)

8.273.3.3 sendRequest() [2/2]

```
void sendRequest (
             WriteSample< TReq > request )
```

Sends a request and gets back information about it that allows correlation with future replies.

After calling this operation, the sample contains a valid identity that can be used for correlation with future replies.

See example code in **Correlating requests and replies** (p. 150).

Parameters

request	<<***inout***>> *(p. 156)* Contains the request and optional write parameters. When this call ends successfully, com.rti.connext.infrastructure.WriteSample<T> contains a valid identity that can be used for correlation with future replies.

Exceptions

One	of the **Standard Return Codes** (p. 261) ; **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) may be reported in the same conditions as in **com.rti.ndds.example.FooDataWriter.write** (p. 1105).

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.SampleData<T>.getIdentity() (p. 1630)

createRequestSample() (p. 1583)

waitForReplies(int,Duration_t,SampleIdentity_t) (p. 1581)

takeReplies(int,SampleIdentity_t) (p. 1576)

Correlating requests and replies (p. 150)

References **SampleIdentity_t.AUTO_SAMPLE_IDENTITY**, **SampleData< T >.getData()**, **WriteSample< T >.getInfo()**, and **WriteParams_t.identity**.

8.273.3.4 receiveReply()

```
boolean receiveReply (
              Sample< TRep > reply,
              Duration_t maxWait )
```

Waits for a reply and copies its contents into a Sample.

This operation is equivalent to using **waitForReplies(int,Duration_t)** (p. 1580) and **takeReply(Sample<TRep>)** (p. 1572).

MT Safety:

Same restrictions as **waitForReplies(int,Duration_t)** (p. 1580)

See also

com.rti.connext.infrastructure.Sample<T>

waitForReplies(int,Duration_t) (p. 1580)

takeReply(Sample<TRep>) (p. 1572)

Basic Requester example (p. 148)

8.273.3.5 receiveReplies() [1/4]

```
List< Sample< TRep > > receiveReplies (
            List< Sample< TRep > > replies,
            int maxCount,
             Duration_t maxWait )
```

Waits for multiple replies and copies them into a list.

This operation is equivalent to using **receiveReplies(List<Sample<TRep>>,int,int,Duration_t)** (p. 1570) with `min_count = 0`

MT Safety:

See **waitForReplies(int,Duration_t)** (p. 1580)

See also

com.rti.connext.infrastructure.Sample<T>

receiveReplies(List<Sample<TRep>>,int,Duration_t) (p. 1569)

References **Requester< TReq, TRep >.receiveReplies()**.

Referenced by **Requester< TReq, TRep >.receiveReplies()**.

8.273.3.6 receiveReplies() [2/4]

```
List< Sample< TRep > > receiveReplies (
            List< Sample< TRep > > replies,
            int minCount,
            int maxCount,
             Duration_t maxWait )
```

Waits for multiple replies and copies them into a list.

This operation is equivalent to using **waitForReplies(int,Duration_t)** (p. 1580) and **takeReplies(List<Sample<TRep>>,int)** (p. 1574).

MT Safety:

See **waitForReplies(int,Duration_t)** (p. 1580)

See also

com.rti.connext.infrastructure.Sample<T>

waitForReplies(int,Duration_t) (p. 1580)

takeReplies(List<Sample<TRep>>,int) (p. 1574)

8.273.3.7 receiveReplies() [3/4]

```
Sample.Iterator< TRep > receiveReplies (
            Duration_t maxWait )
```

Waits for multiple replies and provides a loaned iterator to access them.

This operation is equivalent to using **receiveReplies(int,int,Duration_t)** (p. 1571) with `min_count=1` and `max_count=`**`com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED`** (p. 259)

MT Safety:

See **waitForReplies(int,Duration_t)** (p. 1580)

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

receiveReplies(int,int,Duration_t) (p. 1571)

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**, and **Requester< TReq, TRep >.receiveReplies()**.

8.273.3.8 receiveReplies() [4/4]

```
Sample.Iterator< TRep > receiveReplies (
            int minCount,
            int maxCount,
             Duration_t maxWait )
```

Waits for multiple replies and provides a loaned iterator to access them.

This operation is equivalent to using **waitForReplies(int,Duration_t)** (p. 1580) and **takeReplies(int)** (p. 1573).

MT Safety:

See **waitForReplies(int,Duration_t)** (p. 1580)

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

waitForReplies(int,Duration_t) (p. 1580)

takeReplies(int) (p. 1573)

8.273.3.9 takeReply() [1/2]

```
boolean takeReply (
              Sample< TRep > reply )
```

Copies the contents of a reply into a Sample.

Takes a reply sample from the **Requester** (p. 1563) and makes a copy.

This operation may be used after a call to **waitForReplies(int,Duration_t)** (p. 1580).

To avoid copies, you can use **takeReplies(int)** (p. 1573).

Parameters

reply	The sample where the reply data and the related SampleInfo are copied

Returns

true if there was a reply to get. If this operation returns false, the contents of `reply` remain unchanged, except **com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643) becomes false.

Exceptions

One	of the **Standard Return Codes** (p. 261)

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

waitForReplies(int,Duration_t) (p. 1580)

8.273.3.10 takeReplies() [1/6]

```
Sample.Iterator< TRep > takeReplies ( )
```

Provides a loaned iterator to access the existing replies.

Equivalent to using **takeReplies(int)** (p. 1573) with `max_count=`**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

takeReplies(int) (p. 1573)

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**.

Referenced by **Requester< TReq, TRep >.takeReplies()**.

8.273.3.11 takeReplies() [2/6]

```
Sample.Iterator< TRep > takeReplies (
            int maxCount )
```

Provides a loaned iterator to access the existing replies.

Takes all the existing replies up to `maxCount` and provides a loaned iterator to access them.

This operation does not make a copy of the data.

The loan is returned with **com.rti.connext.infrastructure.Sample<T>.Iterator<T>.close()** (p. 1171)

Since `Sample.Iterator<T>` implements `java.io.Closeable`, the loan is automatically returned when the iterator is enclosed in a `try-with-resources` block (since Java 7)

This operation may be used after a call to **waitForReplies(int,Duration_t)** (p. 1580)

See an **example** here: **Taking loaned samples** (p. 149)

Parameters

maxCount	The maximum number of samples that are taken at a time. The special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) may be used. This value will read up to the limit specified by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_read** (p. 534)

Returns

A iterator with up to maxCount elements. May be empty if there were no replies to get.

Exceptions

One	of the **Standard Return Codes** (p. 261)

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

com.rti.connext.infrastructure.Sample<T>.Iterator<T>.close() (p. 1171)

waitForReplies(int,Duration_t) (p. 1580)

com.rti.ndds.example.FooDataReader.take (p. 1071) (for a more detailed description on how QoS and other parameters affect the underlying DataReader)

Taking loaned samples (p. 149)

Taking samples by copy (p. 150)

8.273.3.12 takeReplies() [3/6]

```
List<  Sample< TRep > > takeReplies (
            List<  Sample< TRep > > replies,
            int maxCount )
```

Copies existing replies into a list.

Takes all the existing replies up to `maxCount` and copies them into a list.

If the first argument is null, a new list containing the copies is created and returned.

If a non-null list is passed as the first argument, the copies are inserted into that list. The list may already contain some elements. Any existing elements are overwritten. If the list contains less elements, samples are inserted at the end; if it contains more elements, the excess samples are removed from the end.

This operation may be used after a call to **waitForReplies(int,Duration_t)** (p. 1580).

Parameters

replies	The list where the samples are copied or null to create a new list.
maxCount	The maximum number of samples that are taken at a time. The special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) may be used.

Returns

Returns `replies` if it's not `null`. Otherwise it returns a new list.

Exceptions

One	of the **Standard Return Codes** (p. 261)

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

waitForReplies(int,Duration_t) (p. 1580)

takeReplies(int) (p. 1573)

Taking loaned samples (p. 149)

8.273.3.13 takeReply() [2/2]

```
boolean takeReply (
               Sample< TRep > reply,
               SampleIdentity_t relatedRequestId )
```

Copies the contents of a reply for a specific request.

This operation is analogous to **takeReply(Sample<TRep>)** (p. 1572) except the reply corresponds to a specific request.

Parameters

reply	The sample where a reply is copied into
relatedRequestId	The identity of a request previously sent by this **Requester** (p. 1563)

Returns

true if there was a reply to get. If this operation returns false, the contents of `reply` remain unchanged, except **com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643) becomes false.

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

takeReply(Sample<TRep>) (p. 1572)

sendRequest(WriteSample<TReq>) (p. 1568)

Correlating requests and replies (p. 150)

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.14 takeReplies() [4/6]

```
Sample.Iterator< TRep > takeReplies (
             SampleIdentity_t relatedRequestId )
```

Provides a loaned iterator to access the existing replies for a specific request.

Equivalent to using **takeReplies(int,SampleIdentity_t)** (p. 1576) with `max_count=`**`com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED`** (p. 259)

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

takeReplies(int,SampleIdentity_t) (p. 1576)

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**, and **Requester< TReq, TRep >.takeReplies()**.

8.273.3.15 takeReplies() [5/6]

```
Sample.Iterator< TRep > takeReplies (
            int maxCount,
             SampleIdentity_t relatedRequestId )
```

Provides a loaned iterator to access the existing replies for a specific request.

This operation is analogous to **takeReplies(int)** (p. 1573) except the replies this operation provides correspond to a specific request.

Parameters

maxCount	The maximum number of samples that are taken at a time. The special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) may be used.
relatedRequestId	The identity of a request previously sent by this **Requester** (p. 1563)

MT Safety:

SAFE

See also

takeReplies(int) (p. 1573)

com.rti.connext.infrastructure.Sample<T>.Iterator<T>

Taking samples by copy (p. 150)

Correlating requests and replies (p. 150)

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.16 takeReplies() [6/6]

```
List<  Sample< TRep > > takeReplies (
             List<  Sample< TRep > > replies,
             int maxCount,
              SampleIdentity_t relatedRequestId )
```

Copies existing replies for a specific request into a list.

This operation is analogous to **takeReplies(List<Sample<TRep>>,int)** (p. 1574) except the replies correspond to a specific request.

Parameters

replies	The list where the samples are copied or null to create a new List.
maxCount	The maximum number of samples that are taken at a time. The special value **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) may be used.
relatedRequestId	The identity of a request previously sent by this **Requester** (p. 1563)

Returns

`replies` if it is not `null`. Otherwise it returns a new `List`.

MT Safety:

SAFE

See also

waitForReplies(int,Duration_t) (p. 1580)

takeReplies(List<Sample<TRep>>,int) (p. 1574)

Taking loaned samples (p. 149)

Correlating requests and replies (p. 150)

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.17 readReply() [1/2]

```
boolean readReply (
              Sample< TRep > reply )
```

Copies the contents of a reply into a Sample.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReply(Sample<TRep>)** (p. 1572) except the reply remains in the **Requester** (p. 1563) and can be read or taken again.

8.273.3.18 readReplies() [1/6]

```
Sample.Iterator< TRep > readReplies ( )
```

Provides a loaned iterator to access the existing replies.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies()** (p. 1572) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**.

Referenced by **Requester< TReq, TRep >.readReplies()**.

8.273.3.19 readReplies() [2/6]

```
Sample.Iterator< TRep > readReplies (
            int maxCount )
```

Provides a loaned iterator to access the existing replies.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(int)** (p. 1573) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

8.273.3.20 readReplies() [3/6]

```
List<  Sample< TRep > > readReplies (
            List<  Sample< TRep > > replies,
            int maxCount )
```

Copies existing replies into a list.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(List<↩ Sample<TRep>>,int)** (p. 1574) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

8.273.3.21 readReply() [2/2]

```
boolean readReply (
             Sample< TRep > reply,
             SampleIdentity_t relatedRequestId )
```

Copies the contents of a reply for a specific request.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReply(Sample<↩ TRep>,SampleIdentity_t)** (p. 1575) except the reply remains in the **Requester** (p. 1563) and can be read or taken again.

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.22 readReplies() [4/6]

```
Sample.Iterator< TRep > readReplies (
             SampleIdentity_t relatedRequestId )
```

Provides a loaned iterator to access the existing replies for a specific request.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(Sample↩ Identity_t)** (p. 1576) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

References **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**, and **Requester< TReq, TRep >.readReplies()**.

8.273.3.23 readReplies() [5/6]

```
Sample.Iterator< TRep > readReplies (
            int maxCount,
             SampleIdentity_t relatedRequestId )
```

Provides a loaned iterator to access the existing replies for a specific request.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(int,SampleIdentity_t)** (p. 1576) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.24 readReplies() [6/6]

```
List< Sample< TRep > > readReplies (
            List< Sample< TRep > > replies,
            int maxCount,
             SampleIdentity_t relatedRequestId )
```

Copies existing replies for a specific request into a list.

This operation is equivalent to **com.rti.connext.requestreply.Requester<TReq,TRep>.takeReplies(List<Sample<TRep>>,int,SampleIdentity_t)** (p. 1577) except the replies remain in the **Requester** (p. 1563) and can be read or taken again.

References **SampleStateKind.ANY_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.25 waitForReplies() [1/3]

```
boolean waitForReplies (
             Duration_t maxWait )
```

Waits for replies to any request.

This operation is equivalent to using **waitForReplies(int,Duration_t)** (p. 1580) with `min_count=1`.

See also

waitForReplies(int,Duration_t) (p. 1580)

References **Requester< TReq, TRep >.waitForReplies()**.

Referenced by **Requester< TReq, TRep >.waitForReplies()**.

8.273.3.26 waitForReplies() [2/3]

```
boolean waitForReplies (
             int minCount,
              Duration_t maxWait )
```

Waits for replies to any request.

This operation waits for minCount requests to be available for up to maxWait .

If this operation is called several times but the available replies are not taken (with **takeReplies(int)** (p. 1573)), this operation may return immediately and will not wait for new replies. New replies may replace existing ones if they are not taken, depending on the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) used to configure this **Requester** (p. 1563).

Parameters

minCount	Minimum number of replies that need to be available for this operation to unblock.
maxWait	Maximum waiting time after which this operation unblocks, regardless of how many replies are available.

Returns

true if at least minCount replies were available before maxWait elapsed, or false otherwise.

MT Safety:

Concurrent calls to this operation on the same object are not allowed. However, waiting for replies for specific requests in parallel is supported (see **waitForReplies(int,Duration_t,SampleIdentity_t)** (p. 1581)).

See also

takeReplies(int) (p. 1573)

8.273.3.27 waitForReplies() [3/3]

```
boolean waitForReplies (
             int minCount,
              Duration_t maxWait,
              SampleIdentity_t relatedRequestId )
```

Waits for replies to a specific request.

This operation is analogous to **waitForReplies(int,Duration_t)** (p. 1580) except this operation waits for replies for a specific request.

Parameters

minCount	Minimum number of replies for the related request that need to be available for this operation to unblock.
maxWait	Maximum wait time after which this operation unblocks, regardless of how many replies are available.
related↩ RequestId	The identity of a request previously sent by this **Requester** (p. 1563)

Returns

true if at least minCount replies for the related request were available before maxWait elapsed, or false otherwise.

Exceptions

One	of the **Standard Return Codes** (p. 261)

MT Safety:

SAFE

See also

waitForReplies(int,Duration_t) (p. 1580)

References **SampleStateKind.ANY_SAMPLE_STATE**, **SampleStateKind.NOT_READ_SAMPLE_STATE**, and **SampleIdentity_t.sequence_number**.

8.273.3.28 getRequestDataWriter()

```
DataWriter getRequestDataWriter ( )
```

Retrieves the underlying **com.rti.dds.publication.DataWriter** (p. 553).

Accessing the request DataWriter may be useful for a number of advanced use cases, such as:

- Finding matching subscriptions (e.g., Repliers)
- Setting a DataWriter listener
- Getting DataWriter protocol or cache statuses

MT Safety:

SAFE

See also

com.rti.dds.publication.DataWriter (p. 553)

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.publication.DataWriter.get_matched_subscriptions (p. 566)

com.rti.dds.publication.DataWriter.get_matched_subscription_data (p. 568)

com.rti.dds.publication.DataWriter.set_listener (p. 558)

com.rti.dds.publication.DataWriter.get_datawriter_protocol_status (p. 562)

8.273.3.29 getReplyDataReader()

```
DataReader getReplyDataReader ( )
```

Retrieves the underlying **com.rti.dds.subscription.DataReader** (p. 450).

Accessing the reply DataReader may be useful for a number of advanced use cases, such as:

- Finding matching publications (e.g., Repliers)
- Setting a DataReader listener
- Getting DataReader protocol or cache statuses

MT Safety:

SAFE

See also

com.rti.dds.subscription.DataReader (p. 450)

com.rti.ndds.example.FooDataReader (p. 1067)

com.rti.dds.subscription.DataReader.get_matched_publications (p. 465)

com.rti.dds.subscription.DataReader.get_matched_publication_data (p. 466)

com.rti.dds.subscription.DataReader.set_listener (p. 459)

com.rti.dds.subscription.DataReader.get_datareader_protocol_status (p. 463)

8.273.3.30 createRequestSample() [1/2]

```
WriteSample< TReq > createRequestSample ( )
```

Creates a WriteSample for sending requests.

Returns

A new WriteSample with data of type `TReq` initialized with default values and default write parameters.

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.WriteSample<T>

sendRequest(WriteSample<TReq>) (p. 1568)

8.273.3.31 createRequestSample() [2/2]

```
WriteSample< TReq > createRequestSample (
            TReq data )
```

Creates a WriteSample for sending requests.

Returns

A new WriteSample with the specified data and default write parameters.

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.WriteSample<T>

sendRequest(WriteSample<TReq>) (p. 1568)

8.273.3.32 createReplySample() [1/2]

```
Sample< TRep > createReplySample ( )
```

Creates a Sample for getting replies.

Returns

A new Sample with default data of type `TRep` and invalid SampleInfo.

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

receiveReply(Sample<TRep>,Duration_t) (p. 1569)

takeReply(Sample<TRep>) (p. 1572)

takeReply(Sample<TRep>,SampleIdentity_t) (p. 1575)

8.273.3.33 createReplySample() [2/2]

```
Sample< TRep > createReplySample (
            TRep data )
```

Creates a Sample for getting replies.

Returns

A new Sample with default data of type `TRep` and invalid SampleInfo.

MT Safety:

SAFE

See also

com.rti.connext.infrastructure.Sample<T>

receiveReply(Sample<TRep>,Duration_t) (p. 1569)

takeReply(Sample<TRep>) (p. 1572)

takeReply(Sample<TRep>,SampleIdentity_t) (p. 1575)

8.274 RequesterParams Class Reference

Contains the parameters for creating a com.rti.connext.requestreply.Requester<TReq,TRep>

Inherits EntityParams.

Public Member Functions

- **RequesterParams** (**DomainParticipant** participant, **TypeSupport** requestTypeSupport, **TypeSupport** reply↩ TypeSupport)

  *Creates a **RequesterParams** (p. 1586) with the parameters a **Requester** (p. 1563) always needs.*
- **RequesterParams setServiceName** (String serviceName)

  *The service name that Repliers and **Requester** (p. 1563) use to match and communicate.*
- **RequesterParams setRequestTopicName** (String requestTopicName)

  Sets a specific request topic name.
- **RequesterParams setReplyTopicName** (String replyTopicName)

  Sets a specific reply topic name.
- **RequesterParams setQosProfile** (String qosLibraryName, String qosProfileName)

  Sets a QoS profile for the entities in this requester.
- **RequesterParams setDataWriterQos** (**DataWriterQos** dataWriterQos)

  Sets the quality of service of the request DataWriter.
- **RequesterParams setDataReaderQos** (**DataReaderQos** dataReaderQos)

  Sets the quality of service of the request DataReader.
- **RequesterParams setPublisher** (**Publisher** publisher)

  Sets a specific Publisher.
- **RequesterParams setSubscriber** (**Subscriber** subscriber)

  Sets a specific Subscriber.

8.274.1 Detailed Description

Contains the parameters for creating a com.rti.connext.requestreply.Requester<TReq,TRep>

See also

Creating a Requester with optional parameters (p. 148)

8.274.2 Constructor & Destructor Documentation

8.274.2.1 RequesterParams()

```
RequesterParams (
            DomainParticipant participant,
            TypeSupport requestTypeSupport,
            TypeSupport replyTypeSupport )
```

Creates a **RequesterParams** (p. 1586) with the parameters a **Requester** (p. 1563) always needs.

In addition to the parameters this constructor takes , a **Requester** (p. 1563) needs either:

- A service name (**setServiceName(String)** (p. 1587)),
- Or custom topic names (**setRequestTopicName(String)** (p. 1587) and **setReplyTopicName(String)** (p. 1588))

The rest of the parameters that can be set in a **RequesterParams** (p. 1586) object are optional.

Parameters

participant	The **com.rti.dds.domain.DomainParticipant** (p. 670) this requester uses to join a domain.
requestTypeSupport	The type support for type `TReq`
replyTypeSupport	The type support for type `TReq`

See also

Creating a Requester with optional parameters (p. 148)

8.274.3 Member Function Documentation

8.274.3.1 setServiceName()

```
RequesterParams setServiceName (
            String serviceName )
```

The service name that Repliers and **Requester** (p. 1563) use to match and communicate.

A **Requester** (p. 1563) and a **Replier** (p. 1542) need to be configured with the same topic names in order to match.

The service name is used to generate a request topic and a reply topic that the **Requester** (p. 1563) and **Replier** (p. 1542) will use to communicate. For example, the service name "MyService" will be used to create topics named "MyServiceRequest" and "MyServiceReply".

In some cases, the name of these topics is known beforehand or needs to be customized for another reason. The service name can be overridden by setting specific request and reply topic names using **setRequestTopicName(↩ String)** (p. 1587) and **setReplyTopicName(String)** (p. 1588).

Referenced by **Requester< TReq, TRep >.Requester()**.

8.274.3.2 setRequestTopicName()

```
RequesterParams setRequestTopicName (
            String requestTopicName )
```

Sets a specific request topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **setService↩ Name(String)** (p. 1587). If that topic already exists, it will be reused.

8.274.3.3 setReplyTopicName()

```
RequesterParams setReplyTopicName (
            String replyTopicName )
```

Sets a specific reply topic name.

The specified topic name will be used, instead of allowing a topic name to be generated based on the **setService↩ Name(String)** (p. 1587). If that topic already exists, it will be reused.

8.274.3.4 setQosProfile()

```
RequesterParams setQosProfile (
            String qosLibraryName,
            String qosProfileName )
```

Sets a QoS profile for the entities in this requester.

Specifies an XML QoS profile that will be used to configure the quality of service of a **Requester** (p. 1563)'s underlying request DataWriter and reply DataReader.

Within that profile, the DataWriter QoS in <datawriter_qos> will be used to configure the request DataWriter and the DataReader.

Alternatively, you can set the QoS using the DataWriterQos and DataReaderQos objects (**com.rti.connext.↩ requestreply.RequesterParams.setDataWriterQos(DataWriterQos)** (p. 1588), **com.rti.connext.requestreply.↩ RequesterParams.setDataReaderQos(DataReaderQos)** (p. 1589)).

Parameters

qosLibraryName	The name of the QoS library
qosProfileName	The name of the QoS profile inside the QoS library

See also

Configuring Request-Reply QoS profiles (p. 153)

Configuring QoS Profiles with XML (p. 120)

8.274.3.5 setDataWriterQos()

```
RequesterParams setDataWriterQos (
            DataWriterQos dataWriterQos )
```

Sets the quality of service of the request DataWriter.

See also

setQosProfile(String,String) (p. 1588)

8.274.3.6 setDataReaderQos()

```
RequesterParams setDataReaderQos (
            DataReaderQos dataReaderQos )
```

Sets the quality of service of the request DataReader.

See also

setQosProfile(String,String) (p. 1588)

8.274.3.7 setPublisher()

```
RequesterParams setPublisher (
            Publisher publisher )
```

Sets a specific Publisher.

By default, a **Requester** (p. 1563) uses the DomainParticipant's implicit Publisher. Sometimes a different Publisher may be needed, for example, to use non-default PublisherQos.

8.274.3.8 setSubscriber()

```
RequesterParams setSubscriber (
            Subscriber subscriber )
```

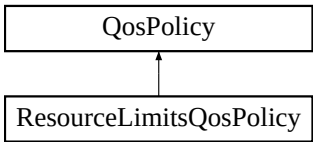
Sets a specific Subscriber.

By default, a **Requester** (p. 1563) uses the DomainParticipant's implicit Subscriber. Sometimes a different Subscriber may be needed, for example, to use non-default SubscriberQos.

8.275 ResourceLimitsQosPolicy Class Reference

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Inheritance diagram for ResourceLimitsQosPolicy:



Public Attributes

- int **max_samples**

  Represents the maximum samples the middleware can store for any one ***com.rti.dds.publication.DataWriter*** *(*p. *553) (or* ***com.rti.dds.subscription.DataReader*** *(*p. *450)).*
- int **max_instances**

  Represents the maximum number of instances a ***com.rti.dds.publication.DataWriter*** *(*p. *553) (or* ***com.rti.dds.subscription.DataReader*** *(*p. *450)) can manage.*
- int **max_samples_per_instance**

  Represents the maximum number of samples of any one instance a ***com.rti.dds.publication.DataWriter*** *(*p. *553) (or* ***com.rti.dds.subscription.DataReader*** *(*p. *450)) can manage.*
- int **initial_samples**

  *<<****extension****>> (*p. *155) Represents the initial samples the middleware will store for any one* ***com.rti.dds.publication.DataWriter*** *(*p. *553) (or* ***com.rti.dds.subscription.DataReader*** *(*p. *450)).*
- int **initial_instances**

  *<<****extension****>> (*p. *155) Represents the initial number of instances a* ***com.rti.dds.publication.DataWriter*** *(*p. *553) (or* ***com.rti.dds.subscription.DataReader*** *(*p. *450)) will manage.*
- int **instance_hash_buckets**

  *<<****extension****>> (*p. *155) Number of hash buckets for instances.*

Static Public Attributes

- static final int **LENGTH_UNLIMITED**

  A special value indicating an unlimited quantity.

8.275.1 Detailed Description

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Entity:

com.rti.dds.topic.Topic (p. 1807), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS, **com.rti.dds.subscription.SampleRejectedStatus** (p. 1657)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.275.2 Usage

This policy controls the resources that RTI Connext can use to meet the requirements imposed by the application and other QoS settings.

For the reliability protocol (and **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 830)), this QoS policy determines the actual maximum queue size when the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) is set to com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS.

In general, this QoS policy is used to limit the amount of system memory that RTI Connext can allocate. For embedded real-time systems and safety-critical systems, pre-determination of maximum memory usage is often required. In addition, dynamic memory allocation could introduce non-deterministic latencies in time-critical paths.

This QoS policy can be set such that an entity does not dynamically allocate any more memory after its initialization phase.

If **com.rti.dds.publication.DataWriter** (p. 553) objects are communicating samples faster than they are ultimately taken by the **com.rti.dds.subscription.DataReader** (p. 450) objects, the middleware will eventually hit against some of the QoS-imposed resource limits. Note that this may occur when just a single **com.rti.dds.subscription.DataReader** (p. 450) cannot keep up with its corresponding **com.rti.dds.publication.DataWriter** (p. 553). The behavior in this case depends on the setting for the **RELIABILITY** (p. 258). If reliability is **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), then RTI Connext is allowed to drop samples. If the reliability is **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), RTI Connext will block the **com.rti.dds.publication.DataWriter** (p. 553) or discard the sample at the **com.rti.dds.subscription.DataReader** (p. 450) in order not to lose existing samples.

The constant **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) may be used to indicate the absence of a particular limit. For example setting **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259) will cause RTI Connext not to enforce this particular limit.

If these resource limits are not set sufficiently, under certain circumstances the **com.rti.dds.publication.DataWriter** (p. 553) may block on a write() call even though the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1144) is com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS. To guarantee the writer does not block for com.rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, make sure the resource limits are set such that:

```
max_samples >= max_instances * max_samples_per_instance
```

See also

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1526)

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

8.275.3 Consistency

The setting of **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) must be consistent with **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593). For these two values to be consistent, it must be true that **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) >= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593). As described above, this limit will not be enforced if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) is set to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

The setting of **RESOURCE_LIMITS** (p. 259) `max_samples_per_instance` must be consistent with the **HISTORY** (p. 237) `depth`. For these two QoS to be consistent, it must be true that *depth* <= *max_samples_per_instance*.

See also

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1144)

8.275.4 Member Data Documentation

8.275.4.1 max_samples

```
int max_samples
```

Represents the maximum samples the middleware can store for any one **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.subscription.DataReader** (p. 450)).

Specifies the maximum number of data samples a **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.subscription.DataReader** (p. 450)) can manage across all the instances associated with it.

For unkeyed types, this value has to be equal to max_samples_per_instance if max_samples_per_instance is not equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

When batching is enabled, the maximum number of data samples a **com.rti.dds.publication.DataWriter** (p. 553) can manage will also be limited by **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_samples, >= max_samples_per_instance, >= **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532) or >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeats_per_max_samples** (p. 1611)

For **com.rti.dds.publication.DataWriterQos** (p. 612) max_samples >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeats_per_max_samples** (p. 1611) in **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600) if batching is disabled.

8.275.4.2 max_instances

```
int max_instances
```

Represents the maximum number of instances a **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.↩ subscription.DataReader** (p. 450)) can manage.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= initial_instances

8.275.4.3 max_samples_per_instance

```
int max_samples_per_instance
```

Represents the maximum number of samples of any one instance a **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.subscription.DataReader** (p. 450)) can manage.

While an unkeyed type is logically considered as a single instance, for unkeyed types this value has to be equal to max_samples or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259).

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), <= max_samples or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), >= **com.↩ rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147)

8.275.4.4 initial_samples

```
int initial_samples
```

<<***extension***>> *(p. 155)* Represents the initial samples the middleware will store for any one **com.rti.dds.↩ publication.DataWriter** (p. 553) (or **com.rti.dds.subscription.DataReader** (p. 450)).

Specifies the initial number of data samples a **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.↩ subscription.DataReader** (p. 450)) will manage across all the instances associated with it.

[default] 32

[range] [1,100 million], <= max_samples

8.275.4.5 initial_instances

```
int initial_instances
```

<<***extension***>> *(p. 155)* Represents the initial number of instances a **com.rti.dds.publication.DataWriter** (p. 553) (or **com.rti.dds.subscription.DataReader** (p. 450)) will manage.

[default] 32

[range] [1,1 million], <= max_instances

8.275.4.6 instance_hash_buckets

```
int instance_hash_buckets
```

<<***extension***>> *(*p. *155)* Number of hash buckets for instances.

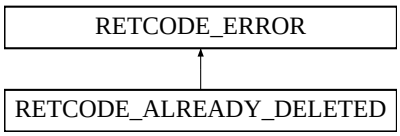
The instance hash table facilitates instance lookup. A higher number of buckets decreases instance lookup time but increases the memory usage.

[default] 1 **[range]** [1,1 million]

8.276 RETCODE_ALREADY_DELETED Class Reference

The object target of this operation has already been deleted.

Inheritance diagram for RETCODE_ALREADY_DELETED:



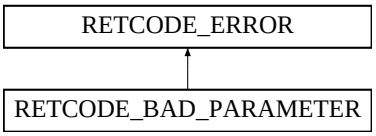
8.276.1 Detailed Description

The object target of this operation has already been deleted.

8.277 RETCODE_BAD_PARAMETER Class Reference

Illegal parameter value.

Inheritance diagram for RETCODE_BAD_PARAMETER:



8.277.1 Detailed Description

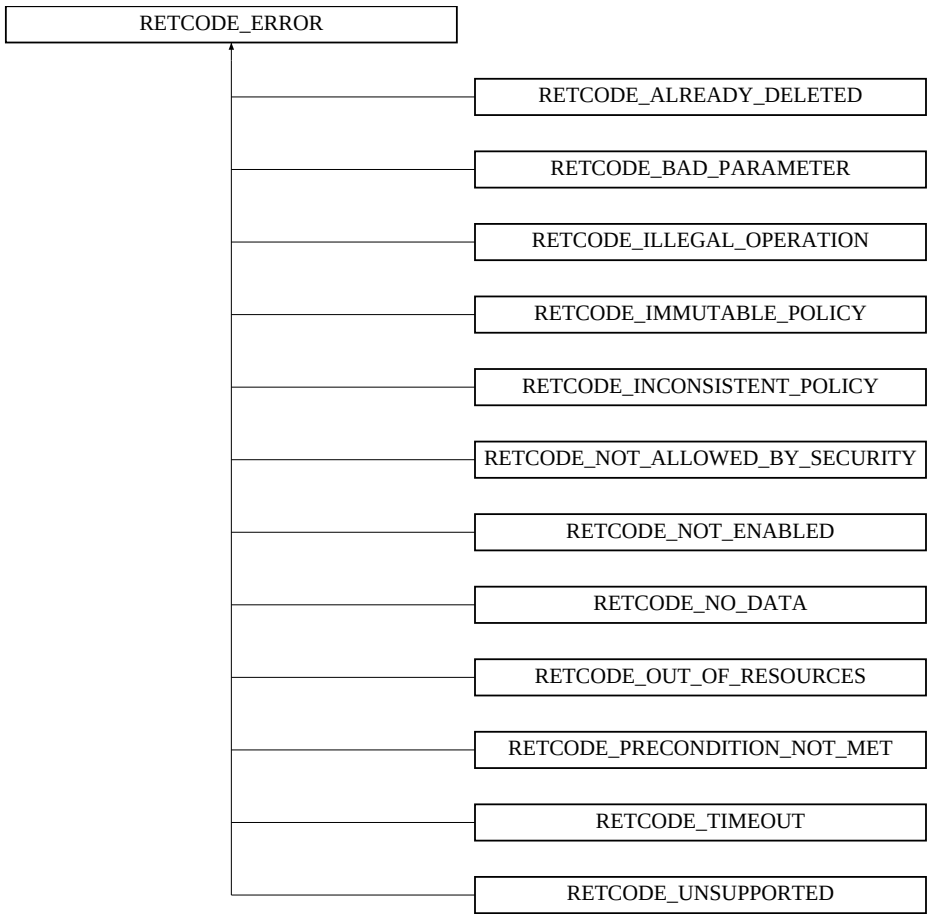
Illegal parameter value.

The value of the parameter that is passed in has llegal value. Things that fall into this category include null parameters and parameter values that are out of range.

8.278 RETCODE_ERROR Class Reference

Generic, unspecified error.

Inheritance diagram for RETCODE_ERROR:



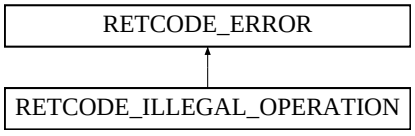
8.278.1 Detailed Description

Generic, unspecified error.

8.279 RETCODE_ILLEGAL_OPERATION Class Reference

The operation was called under improper circumstances.

Inheritance diagram for RETCODE_ILLEGAL_OPERATION:



8.279.1 Detailed Description

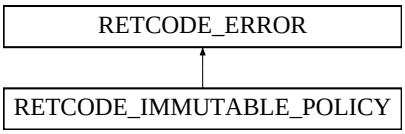
The operation was called under improper circumstances.

An operation was invoked on an inappropriate object or at an inappropriate time. This return code is similar to **com.rti.↩ dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598), except that there is no precondition that could be changed to make the operation succeed.

8.280 RETCODE_IMMUTABLE_POLICY Class Reference

Application attempted to modify an immutable QoS policy.

Inheritance diagram for RETCODE_IMMUTABLE_POLICY:



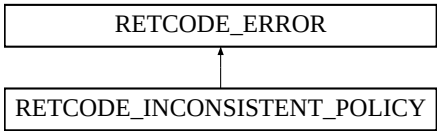
8.280.1 Detailed Description

Application attempted to modify an immutable QoS policy.

8.281 RETCODE_INCONSISTENT_POLICY Class Reference

Application specified a set of QoS policies that are not consistent with each other.

Inheritance diagram for RETCODE_INCONSISTENT_POLICY:



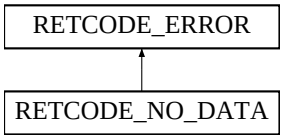
8.281.1 Detailed Description

Application specified a set of QoS policies that are not consistent with each other.

8.282 RETCODE_NO_DATA Class Reference

Indicates a transient situation where the operation did not return any data but there is no inherent error.

Inheritance diagram for RETCODE_NO_DATA:



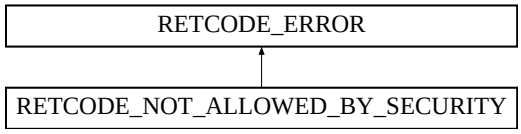
8.282.1 Detailed Description

Indicates a transient situation where the operation did not return any data but there is no inherent error.

8.283 RETCODE_NOT_ALLOWED_BY_SECURITY Class Reference

An operation on the DDS API that fails because the security plugins do not allow it.

Inheritance diagram for RETCODE_NOT_ALLOWED_BY_SECURITY:



8.283.1 Detailed Description

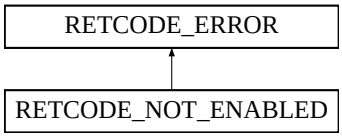
An operation on the DDS API that fails because the security plugins do not allow it.

An operation on the DDS API that fails because the security plugins do not allow it.

8.284 RETCODE_NOT_ENABLED Class Reference

Operation invoked on a **com.rti.dds.infrastructure.Entity** (p. 1029) that is not yet enabled.

Inheritance diagram for RETCODE_NOT_ENABLED:



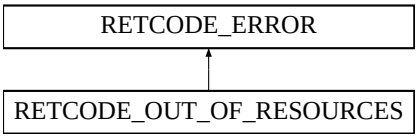
8.284.1 Detailed Description

Operation invoked on a **com.rti.dds.infrastructure.Entity** (p. 1029) that is not yet enabled.

8.285 RETCODE_OUT_OF_RESOURCES Class Reference

RTI Connext ran out of the resources needed to complete the operation.

Inheritance diagram for RETCODE_OUT_OF_RESOURCES:



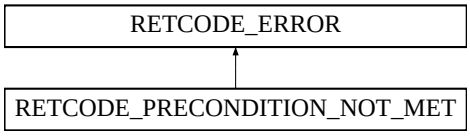
8.285.1 Detailed Description

RTI Connext ran out of the resources needed to complete the operation.

8.286 RETCODE_PRECONDITION_NOT_MET Class Reference

A pre-condition for the operation was not met.

Inheritance diagram for RETCODE_PRECONDITION_NOT_MET:



8.286.1 Detailed Description

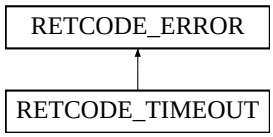
A pre-condition for the operation was not met.

The system is not in the expected state when the function is called, or the parameter itself is not in the expected state when the function is called.

8.287 RETCODE_TIMEOUT Class Reference

The operation timed out.

Inheritance diagram for RETCODE_TIMEOUT:



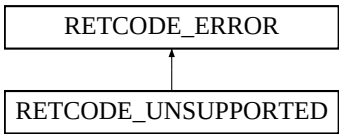
8.287.1 Detailed Description

The operation timed out.

8.288 RETCODE_UNSUPPORTED Class Reference

Unsupported operation. Only returned by operations that are unsupported.

Inheritance diagram for RETCODE_UNSUPPORTED:



8.288.1 Detailed Description

Unsupported operation. Only returned by operations that are unsupported.

8.289 RtpsReliableReaderProtocol_t Class Reference

Qos (p. 1500) related to reliable reader protocol defined in RTPS.

Inherits Struct.

Public Member Functions

- **RtpsReliableReaderProtocol_t** ()

  Constructor with default values.
- **RtpsReliableReaderProtocol_t** (**Duration_t min_heartbeat_response_delay**, **Duration_t max_heartbeat_response_delay**, **Duration_t heartbeat_suppression_duration**, **Duration_t nack_period**, **Duration_t round_trip_time**, **Duration_t app_ack_period**, **Duration_t min_app_ack_response_keep_duration**)

  Constructor with given durations.

Public Attributes

- final **Duration_t min_heartbeat_response_delay**

  The minimum delay to respond to a heartbeat.
- final **Duration_t max_heartbeat_response_delay**

  The maximum delay to respond to a heartbeat.
- final **Duration_t heartbeat_suppression_duration**

  The duration a reader ignores consecutively received heartbeats.
- final **Duration_t nack_period**

  The period at which to send NACKs.
- int **receive_window_size** = 256

  The number of received out-of-order samples a reader can keep at a time.
- final **Duration_t round_trip_time**

  The duration from sending a NACK to receiving a repair of a sample.
- final **Duration_t app_ack_period**

  The period at which application-level acknowledgment messages are sent.
- final **Duration_t min_app_ack_response_keep_duration**

  Minimum duration for which application-level acknowledgment response data is kept.
- int **samples_per_app_ack**

  The minimum number of samples acknowledged by one application-level acknowledgment message.

8.289.1 Detailed Description

Qos (p. 1500) related to reliable reader protocol defined in RTPS.

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

QoS:

com.rti.dds.infrastructure.DataReaderProtocolQosPolicy (p. 501) **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy** (p. 646)

8.289.2 Constructor & Destructor Documentation

8.289.2.1 RtpsReliableReaderProtocol_t() [1/2]

```
RtpsReliableReaderProtocol_t ( )
```

Constructor with default values.

8.289.2.2 RtpsReliableReaderProtocol_t() [2/2]

```
RtpsReliableReaderProtocol_t (
            Duration_t min_heartbeat_response_delay,
            Duration_t max_heartbeat_response_delay,
            Duration_t heartbeat_suppression_duration,
            Duration_t nack_period,
            Duration_t round_trip_time,
            Duration_t app_ack_period,
            Duration_t min_app_ack_response_keep_duration )
```

Constructor with given durations.

References **RtpsReliableReaderProtocol_t.app_ack_period**, **RtpsReliableReaderProtocol_t.heartbeat_↩ suppression_duration**, **RtpsReliableReaderProtocol_t.max_heartbeat_response_delay**, **RtpsReliableReader↩ Protocol_t.min_app_ack_response_keep_duration**, **RtpsReliableReaderProtocol_t.min_heartbeat_response↩ _delay**, **RtpsReliableReaderProtocol_t.nack_period**, and **RtpsReliableReaderProtocol_t.round_trip_time**.

8.289.3 Member Data Documentation

8.289.3.1 min_heartbeat_response_delay

```
final  Duration_t min_heartbeat_response_delay
```

The minimum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of the minimum delay.

[default] 0 seconds

[range] [0, 1 year], <= max_heartbeat_response_delay

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.2 max_heartbeat_response_delay

```
final Duration_t max_heartbeat_response_delay
```

The maximum delay to respond to a heartbeat.

When a reliable reader receives a heartbeat from a remote writer and finds out that it needs to send back an ACK/NACK message, the reader can choose to delay a while. This sets the value of maximum delay.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_reliable_reader** (p. 504): 0.5 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_reader** (p. 654): 0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_reader** (p. 654): 0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_reader** (p. 656): 0 seconds

[range] [0, 1 year], >= min_heartbeat_response_delay

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.3 heartbeat_suppression_duration

```
final Duration_t heartbeat_suppression_duration
```

The duration a reader ignores consecutively received heartbeats.

When a reliable reader receives consecutive heartbeats within a short duration that will trigger redundant NACKs, the reader may ignore the latter heartbeat(s). This sets the duration during which additionally received heartbeats are suppressed.

[default] 0.0625 seconds

[range] [0, 1 year],

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.4 nack_period

```
final  Duration_t nack_period
```

The period at which to send NACKs.

A reliable reader will send periodic NACKs at this rate when it first matches with a reliable writer. The reader will stop sending NACKs when it has received all available historical data from the writer.

[default] 5 seconds

[range] [1 nanosec, 1 year]

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.5 receive_window_size

```
int receive_window_size = 256
```

The number of received out-of-order samples a reader can keep at a time.

A reliable reader stores the out-of-order samples it receives until it can present them to the application in-order. The receive window is the maximum number of out-of-order samples that a reliable reader keeps at a given time. When the receive window is full, subsequently received out-of-order samples are dropped.

[default] 256

[range] [>= 1]

8.289.3.6 round_trip_time

```
final  Duration_t round_trip_time
```

The duration from sending a NACK to receiving a repair of a sample.

This round-trip time is an estimate of the time starting from when the reader sends a NACK for a specific sample to when it receives that sample. For each sample, the reader will not send a subsequent NACK for it until the round-trip time has passed, thus preventing inefficient redundant requests.

[default] 0 seconds

[range] [0 nanosec, 1 year]

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.7 app_ack_period

```
final Duration_t app_ack_period
```

The period at which application-level acknowledgment messages are sent.

A **com.rti.dds.subscription.DataReader** (p. 450) sends application-level acknowledgment messages to a **com.rti.dds.publication.DataWriter** (p. 553) at this periodic rate, and will continue sending until it receives a message from the **com.rti.dds.publication.DataWriter** (p. 553) that it has received and processed the acknowledgment and an AppAckConfirmation has been received by the **com.rti.dds.subscription.DataReader** (p. 450). Note: application-level acknowledgment messages can also be sent non-periodically, as determined by **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.samples_per_app_ack** (p. 1604).

[default] 5 seconds

[range] [1 nanosec, 1 year]

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.8 min_app_ack_response_keep_duration

```
final Duration_t min_app_ack_response_keep_duration
```

Minimum duration for which application-level acknowledgment response data is kept.

The user-specified response data of an explicit application-level acknowledgment (called by **com.rti.dds.subscription.DataReader.acknowledge_sample** (p. 470) or **com.rti.dds.subscription.DataReader.acknowledge_all** (p. 471)) is cached by the **com.rti.dds.subscription.DataReader** (p. 450) for the purpose of reliably resending the data with the acknowledgment message. After this duration has passed from the time of the first acknowledgment, the response data is dropped from the cache and will not be resent with future acknowledgments for the corresponding sample(s).

[default] 0 sec

[range] [0 sec, 1 year]

Referenced by **RtpsReliableReaderProtocol_t.RtpsReliableReaderProtocol_t()**.

8.289.3.9 samples_per_app_ack

```
int samples_per_app_ack
```

The minimum number of samples acknowledged by one application-level acknowledgment message.

This setting applies only when **com.rti.dds.infrastructure.ReliabilityQosPolicy.acknowledgment_kind** (p. 1529) = **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgmentModeKind.APPLICATION_EXPLICIT_↩ ACKNOWLEDGMENT_MODE** (p. 1531) or **com.rti.dds.infrastructure.ReliabilityQosPolicyAcknowledgment↩ ModeKind.APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1530)

A **com.rti.dds.subscription.DataReader** (p. 450) will immediately send an application-level acknowledgment message when it has at least this many samples that have been acknowledged. It will not send an acknowledgment message until it has at least this many samples pending acknowledgment.

For example, calling **com.rti.dds.subscription.DataReader.acknowledge_sample** (p. 470) this many times consecutively will trigger the sending of an acknowledgment message. Calling **com.rti.dds.subscription.DataReader.↩ acknowledge_all** (p. 471) may trigger the sending of an acknowledgment message, if at least this many samples are being acknowledged at once.

This is independent of the **com.rti.dds.infrastructure.RtpsReliableReaderProtocol_t.app_ack_period** (p. 1603), where a **com.rti.dds.subscription.DataReader** (p. 450) will send acknowledgement messages at the periodic rate regardless.

When this is set to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), then acknowledgement messages are sent only periodically, at the rate set by **com.rti.dds.infrastructure.RtpsReliable↩ ReaderProtocol_t.app_ack_period** (p. 1603).

[default] 1

[range] [1, 1000000], or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.290 RtpsReliableWriterProtocol_t Class Reference

QoS related to the reliable writer protocol defined in RTPS.

Inherits Struct.

Public Attributes

- int **low_watermark**

  When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.
- int **high_watermark**

  When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.
- final **Duration_t heartbeat_period**

The period at which to send heartbeats.

- final **Duration_t fast_heartbeat_period**

  An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

- final **Duration_t late_joiner_heartbeat_period**

  An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.

- final **Duration_t virtual_heartbeat_period**

  The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

- int **samples_per_virtual_heartbeat**

  The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

- int **max_heartbeat_retries**

  The maximum number of periodic heartbeat retries before marking a remote reader as inactive.

- boolean **inactivate_nonprogressing_readers**

  Whether to treat remote readers as inactive when their NACKs do not progress.

- int **heartbeats_per_max_samples**

  The number of piggyback heartbeats sent per max send window.

- final **Duration_t min_nack_response_delay**

  The minimum delay to respond to a NACK.

- final **Duration_t max_nack_response_delay**

  The maximum delay to respond to a nack.

- final **Duration_t nack_suppression_duration**

  The duration for ignoring consecutive NACKs that may trigger redundant repairs.

- int **max_bytes_per_nack_response**

  The maximum total message size when resending rejected samples.

- final **Duration_t disable_positive_acks_min_sample_keep_duration**

  The minimum duration a sample is queued for ACK-disabled readers.

- final **Duration_t disable_positive_acks_max_sample_keep_duration**

  The maximum duration a sample is queued for ACK-disabled readers.

- boolean **disable_positive_acks_enable_adaptive_sample_keep_duration**

  Enables dynamic adjustment of sample keep duration in response to congestion.

- int **disable_positive_acks_decrease_sample_keep_duration_factor**

  Controls rate of contraction of dynamic sample keep duration.

- int **disable_positive_acks_increase_sample_keep_duration_factor**

  Controls rate of growth of dynamic sample keep duration.

- int **min_send_window_size**

  Minimum size of send window of unacknowledged samples.

- int **max_send_window_size**

  Maximum size of send window of unacknowledged samples.

- final **Duration_t send_window_update_period**

  Period in which send window may be dynamically changed.

- int **send_window_increase_factor**

  Increases send window size by this percentage when reacting dynamically to network conditions.

- int **send_window_decrease_factor**

  Decreases send window size by this percentage when reacting dynamically to network conditions.

- int **multicast_resend_threshold**

  The minimum number of requesting readers needed to trigger a multicast resend.

- boolean **enable_multicast_periodic_heartbeat**

  Whether periodic heartbeat messages are sent over multicast.
- boolean **disable_repair_piggyback_heartbeat**

  Prevents piggyback heartbeats from being sent with repair samples.

8.290.1 Detailed Description

QoS related to the reliable writer protocol defined in RTPS.

It is used to configure a reliable writer according to RTPS protocol.

The reliability protocol settings are applied to batches instead of individual data samples when batching is enabled.

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

QoS:

com.rti.dds.infrastructure.DataWriterProtocolQosPolicy (p. 596) **com.rti.dds.infrastructure.Discovery↩ ConfigQosPolicy** (p. 646)

8.290.2 Member Data Documentation

8.290.2.1 low_watermark

```
int low_watermark
```

When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be greater than or equal to zero and strictly less than high_watermark.

The high and low watermarks are used for switching between the regular and fast heartbeat rates (**com.rti.dds.↩ infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1608) and **com.rti.dds.infrastructure.Rtps↩ ReliableWriterProtocol_t.fast_heartbeat_period** (p. 1608), respectively). When the number of unacknowledged samples in the queue of a reliable **com.rti.dds.publication.DataWriter** (p. 553) meets or exceeds high_watermark, the com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is changed, and the DataWriter will start heartbeating at **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.fast_heartbeat_↩ period** (p. 1608). When the number of samples meets or falls below low_watermark, com.rti.dds.infrastructure.Status↩ Kind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is changed, and the heartbeat rate will return to the "normal" rate (**com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1608)).

[default] 0

[range] [0, 100 million], $<$ high_watermark

8.290.2.2 high_watermark

```
int high_watermark
```

When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, the com.rti.dds.infrastructure.StatusKind.StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.

This value is measured in units of samples, except with batching configurations where it is measured in units of batches.

The value must be strictly greater than low_watermark and less than or equal to a maximum that depends on the container QoS policy:

In **com.rti.dds.domain.DomainParticipantQos.discovery_config** (p. 801):

For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652)
high_watermark<= **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.local_writer_allocation** (p. 807)

For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653)
high_watermark<= **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.local_reader_allocation** (p. 807)

In **com.rti.dds.publication.DataWriterQos.protocol** (p. 620):

For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600),

high_watermark<=**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) if batching is disabled. Otherwise,
high_watermark<=**com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628) high_watermark<=**com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617)

[default] 1

[range] [1, 100 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), > low_watermark <= *maximum* which depends on the container policy

8.290.2.3 heartbeat_period

```
final Duration_t heartbeat_period
```

The period at which to send heartbeats.

A reliable writer will send periodic heartbeats at this rate.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 1.0 seconds

[range] [1 nanosec,1 year], >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.fast_heartbeat_period** (p. 1608), >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.late_joiner_heartbeat_period** (p. 1609)

8.290.2.4 fast_heartbeat_period

```
final Duration_t fast_heartbeat_period
```

An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

This heartbeat period will be used when the number of unacknowledged samples in the cache of a reliable writer meets or exceeds the writer's high watermark and has not subsequently dropped to the low watermark. The normal period will be used at all other times.

This period must not be slower (i.e. must be of the same or shorter duration) than the normal heartbeat period.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 1.0 seconds

[range] [1 nanosec,1 year], <= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1608)

8.290.2.5 late_joiner_heartbeat_period

```
final Duration_t late_joiner_heartbeat_period
```

An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.

This heartbeat period will be used when a reliable reader joins after a reliable writer with non-volatile durability has begun publishing samples. Once the reliable reader has received all cached samples, it will be serviced at the same rate as other reliable readers.

This period must not be slower (i.e., must be of the same or shorter duration) than the normal heartbeat period.

A reliable writer will use whichever heartbeat period is faster, the current heartbeat period being used for other reliable readers or the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.late_joiner_heartbeat_period** (p. 1609), to service the late joining reader. This means that if the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.↩ fast_heartbeat_period** (p. 1608) is currently being used and is faster than the late_joiner_heartbeat_period, then the fast_heartbeat_period will continue to be used for the late joiner as well.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 3.0 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 1.0 seconds

[range] [1 nanosec,1 year], <= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1608)

8.290.2.6 virtual_heartbeat_period

```
final Duration_t virtual_heartbeat_period
```

The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

A reliable writer will send periodic virtual heartbeats at this rate.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): com.rti.dds.infrastructure.Duration_t.AUTO. If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) is set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS on the DataWriter, this value is set to **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1608). Otherwise, the value is set to **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

[range] > 1 nanosec, **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), or com.rti.dds.infrastructure.Duration_t.AUTO

8.290.2.7 samples_per_virtual_heartbeat

```
int samples_per_virtual_heartbeat
```

The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1,1000000], **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.290.2.8 max_heartbeat_retries

```
int max_heartbeat_retries
```

The maximum number of *periodic* heartbeat retries before marking a remote reader as inactive.

When a remote reader has not acked all the samples the reliable writer has in its queue, and max_heartbeat_retries number of periodic heartbeats has been sent without receiving any ack/nack back, the remote reader will be marked as inactive (not alive) and be ignored until it resumes sending ack/nack.

Note that piggyback heartbeats do NOT count towards this value.

[default] 10

[range] [1, 1 million] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.290.2.9 inactivate_nonprogressing_readers

```
boolean inactivate_nonprogressing_readers
```

Whether to treat remote readers as inactive when their NACKs do not progress.

Nominally, a remote reader is marked inactive when a successive number of periodic heartbeats equal or greater than **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_heartbeat_retries** (p. 1611) have been sent without receiving any ack/nacks back.

By setting this com.rti.dds.infrastructure.true, it changes the conditions of inactivating a remote reader: a reader will be considered inactive when it either does not send any ack/nacks or keeps sending non-progressing nacks for **com.↩ rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_heartbeat_retries** (p. 1611) number of heartbeat periods, where a non-progressing nack is one whose oldest sample requested has not advanced from the oldest sample requested of the previous nack.

[default] com.rti.dds.infrastructure.false

8.290.2.10 heartbeats_per_max_samples

```
int heartbeats_per_max_samples
```

The number of piggyback heartbeats sent per max send window.

When a DataWriter is configured with a fixed send window size (**com.rti.dds.infrastructure.RtpsReliableWriter↩ Protocol_t.min_send_window_size** (p. 1616) is equal to effective max_send_window_size), a piggyback heartbeat is sent every [(effective max send window size/heartbeats_per_max_samples)] number of samples written.

Otherwise, the number of piggyback heartbeats sent is scaled according to the current size of the send window. For example, consider a **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeats_per_max_samples** (p. 1611) of 50. If the current send window size is 100, a piggyback heartbeat will be sent every 2 samples. If the send window size grows to 150, a piggyback heartbeat will be sent every 3 samples, and so on. Additionally, when the send window size grows, a piggyback heartbeat is sent with the next sample. (If it weren't, the sending of that heartbeat could be delayed, since the heartbeat rate scales with the increasing window size.)

The effective max send window is calculated as follows:

Without batching:

min (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592), **com.rti.dds.infrastructure.↩ RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617))

With batching:

min (**com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628), **com.rti.dds.↩ infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1617))

If heartbeats_per_max_samples is set to zero, no piggyback heartbeats will be sent.

If current send window size is **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259), 100 million is assumed as the effective max send window.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 8
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 8
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 8
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 1

[range] [0, 100 million]

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): heartbeats_per_max_samples<= **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.local_writer_allocation** (p. 807)

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): heartbeats_per_max_samples<= **com.rti.dds.infrastructure.AllocationSettings_t.max_count** (p. 338) in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.local_reader_allocation** (p. 807)

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600):

heartbeats_per_max_samples<= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) if batching is disabled. Otherwise:
heartbeats_per_max_samples<= **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 628)

heartbeats_per_max_samples<= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_↩ size** (p. 1617)

8.290.2.11 min_nack_response_delay

```
final Duration_t min_nack_response_delay
```

The minimum delay to respond to a NACK.

When a reliable writer receives a NACK from a remote reader, the writer can choose to delay a while before it sends repair samples or a heartbeat. This sets the value of the minimum delay.

[default] 0 seconds

[range] [0,1 day], <= max_nack_response_delay

8.290.2.12 max_nack_response_delay

```
final Duration_t max_nack_response_delay
```

The maximum delay to respond to a nack.

This set the value of maximum delay between receiving a NACK and sending repair samples or a heartbeat.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 0.2 seconds

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 0 seconds

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 0 seconds

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 0 seconds

[range] [0,1 day], >= min_nack_response_delay

8.290.2.13 nack_suppression_duration

```
final Duration_t nack_suppression_duration
```

The duration for ignoring consecutive NACKs that may trigger redundant repairs.

A reliable writer may receive consecutive NACKs within a short duration from a remote reader that will trigger the sending of redundant repair messages.

This specifies the duration during which consecutive NACKs are ignored to prevent redundant repairs from being sent.

[default] 0 seconds

[range] [0,1 day],

8.290.2.14 max_bytes_per_nack_response

```
int max_bytes_per_nack_response
```

The maximum total message size when resending rejected samples.

As part of the reliable communication protocol, data writers send heartbeat (HB) messages to their data readers. Each HB message contains the sequence number of the most recent sample sent by the data writer.

In response, a data reader sends an acknowledgement (ACK) message, indicating what sequence numbers it did not receive, if any. If the data reader is missing some samples, the data writer will send them again.

max_bytes_per_nack_response determines the maximum size of the message sent by the data writer in response to an ACK. This message may contain multiple samples. The data writer will always send at least one message, even if the size of that message exceeds the max_bytes_per_nack_response value.

If max_bytes_per_nack_response is larger than the maximum message size supported by the underlying transport, RTI Connext will send multiple messages. If the total size of all samples that need to be resent is larger than max_bytes_↩ per_nack_response, the remaining samples will be resent the next time an ACK arrives.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 131072 bytes
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 131072 bytes
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 131072 bytes
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 9216 bytes

[range] [0, 1 GB]

8.290.2.15 disable_positive_acks_min_sample_keep_duration

```
final Duration_t disable_positive_acks_min_sample_keep_duration
```

The minimum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.↩ disable_positive_acks** (p. 598) = com.rti.dds.infrastructure.true) or a data reader (**com.rti.dds.infrastructure.Data↩ ReaderProtocolQosPolicy.disable_positive_acks** (p. 503) = com.rti.dds.infrastructure.true), a sample is available from the data writer's queue for at least this duration, after which the sample may be considered to be acknowledged.

[default] 1 millisecond

[range] [0,1 year], <= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_max_↩ sample_keep_duration** (p. 1615)

8.290.2.16 disable_positive_acks_max_sample_keep_duration

```
final Duration_t disable_positive_acks_max_sample_keep_duration
```

The maximum duration a sample is queued for ACK-disabled readers.

When positive ACKs are disabled for a data writer (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.↩ disable_positive_acks** (p. 598) = com.rti.dds.infrastructure.true) or a data reader (**com.rti.dds.infrastructure.Data↩ ReaderProtocolQosPolicy.disable_positive_acks** (p. 503) = com.rti.dds.infrastructure.true), a sample is available from the data writer's queue for at most this duration, after which the sample is considered to be acknowledged.

[default] 1 second

[range] [0,1 year], >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_min_↩ sample_keep_duration** (p. 1614)

8.290.2.17 disable_positive_acks_enable_adaptive_sample_keep_duration

```
boolean disable_positive_acks_enable_adaptive_sample_keep_duration
```

Enables dynamic adjustment of sample keep duration in response to congestion.

For dynamic networks where a static minimum sample keep duration may not provide sufficient performance or reliability, setting **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_enable_adaptive↩ _sample_keep_duration** (p. 1615) = com.rti.dds.infrastructure.true, enables the sample keep duration to be dynamically adjusted to adapt to network conditions. The keep duration changes according to the detected level of congestion, which is determined to be proportional to the rate of NACKs received. An adaptive algorithm automatically controls the keep duration to optimize throughput and reliability.

To relieve high congestion, the keep duration is increased to effectively decrease the send rate; this lengthening of the keep duration is controlled by **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_↩ acks_increase_sample_keep_duration_factor** (p. 1616). Alternatively, when congestion is low, the keep duration

is decreased to effectively increase send rate; this shortening of the keep duration is controlled by **com.rti.dds.↩ infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_decrease_sample_keep_duration_factor** (p. 1616).

The lower and upper bounds of the dynamic sample keep duration are set by **com.rti.dds.infrastructure.↩ RtpsReliableWriterProtocol_t.disable_positive_acks_min_sample_keep_duration** (p. 1614) and **com.rti.dds.↩ infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_max_sample_keep_duration** (p. 1615), respectively.

When **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_enable_adaptive_↩ sample_keep_duration** (p. 1615) = com.rti.dds.infrastructure.false, the sample keep duration is set to **com.rti.↩ dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_min_sample_keep_duration** (p. 1614) .

[default] com.rti.dds.infrastructure.true

8.290.2.18 disable_positive_acks_decrease_sample_keep_duration_factor

```
int disable_positive_acks_decrease_sample_keep_duration_factor
```

Controls rate of contraction of dynamic sample keep duration.

Used when **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_enable_adaptive_↩ sample_keep_duration** (p. 1615) = com.rti.dds.infrastructure.true.

When the adaptive algorithm determines that the keep duration should be decreased, this factor (a percentage) is multiplied with the current keep duration to get the new shorter keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 95% would result in a new keep duration of 19 milliseconds.

[default] 95

[range] <= 100

8.290.2.19 disable_positive_acks_increase_sample_keep_duration_factor

```
int disable_positive_acks_increase_sample_keep_duration_factor
```

Controls rate of growth of dynamic sample keep duration.

Used when **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_enable_adaptive_↩ sample_keep_duration** (p. 1615) = com.rti.dds.infrastructure.true.

When the adaptive algorithm determines that the keep duration should be increased, this factor (a percentage) is multiplied with the current keep duration to get the new longer keep duration. For example, if the current keep duration is 20 milliseconds, using the default factor of 150% would result in a new keep duration of 30 milliseconds.

[default] 150

[range] >= 100

8.290.2.20 min_send_window_size

```
int min_send_window_size
```

Minimum size of send window of unacknowledged samples.

A **com.rti.dds.publication.DataWriter** (p. 553) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (this field) and a maximum size (max_send_window_↩ size). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] $>$ 0, $<=$ max_send_window_size, or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259)

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size (p. 1617)

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1607)

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark (p. 1607)

com.rti.dds.publication.ReliableWriterCacheChangedStatus.full_reliable_writer_cache (p. 1537)

8.290.2.21 max_send_window_size

```
int max_send_window_size
```

Maximum size of send window of unacknowledged samples.

A **com.rti.dds.publication.DataWriter** (p. 553) has a limit on the number of unacknowledged samples in-flight at a time. This send window can be configured to have a minimum size (min_send_window_size) and a maximum size (this field). The send window can dynamically change, between the min and max sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When a variable sized send window is used (i.e., when min_send_window_size and max_send_window_size are not set to the same value) the send window is initialized to min_send_window_size.

When both min_send_window_size and max_send_window_size are **com.rti.dds.infrastructure.ResourceLimits↩ QosPolicy.LENGTH_UNLIMITED** (p. 259), then either **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max↩ _samples** (p. 1592) (for non-batching) or **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_↩ batches** (p. 628) (for batching) serves as the effective max_send_window_size. When **com.rti.dds.infrastructure.↩ ResourceLimitsQosPolicy.max_samples** (p. 1592) (for non-batching) or **com.rti.dds.infrastructure.DataWriter↩ ResourceLimitsQosPolicy.max_batches** (p. 628) (for batching) is less than max_send_window_size, then it serves as the effective max_send_window_size. If it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max_samples or max_batches.

In addition, the low and high watermarks are scaled down linearly to stay within the current send window size, and the full reliable queue status is set when the send window is full.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] $>$ 0, $>=$ min_send_window_size, or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_↩ UNLIMITED** (p. 259)

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size (p. 1616)

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1607)

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark (p. 1607)

com.rti.dds.publication.ReliableWriterCacheChangedStatus.full_reliable_writer_cache (p. 1537)

8.290.2.22 send_window_update_period

```
final Duration_t send_window_update_period
```

Period in which send window may be dynamically changed.

The **com.rti.dds.publication.DataWriter** (p. 553)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

The change in send window size happens at this update period, whereupon the send window is either increased or decreased in size according to the increase or decrease factors, respectively.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 3 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 3 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 3 seconds
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 1 second

[range] $>$ [0,1 year]

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_increase_factor (p. 1618), **com.↩ rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_decrease_factor** (p. 1619)

8.290.2.23 send_window_increase_factor

```
int send_window_increase_factor
```

Increases send window size by this percentage when reacting dynamically to network conditions.

The **com.rti.dds.publication.DataWriter** (p. 553)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

After an update period during which no negative acknowledgements were received, the send window will be increased by this factor. The factor is treated as a percentage, where a factor of 150 would increase the send window by 150%. The increased send window size will not exceed the max_send_window_size.

[default] 105

[range] $>$ 100

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_update_period (p. 1618), **com.rti.↩ dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_decrease_factor** (p. 1619)

8.290.2.24 send_window_decrease_factor

```
int send_window_decrease_factor
```

Decreases send window size by this percentage when reacting dynamically to network conditions.

The **com.rti.dds.publication.DataWriter** (p. 553)'s send window will dynamically change, between the min and max send window sizes, to throttle the effective send rate in response to changing network congestion, as measured by negative acknowledgements received.

When increased network congestion causes a negative acknowledgement to be received by a writer, the send window will be decreased by this factor to throttle the effective send rate. The factor is treated as a percentage, where a factor of 80 would decrease the send window to 80% of its previous size. The decreased send window size will not be less than the min_send_window_size.

[default] The default value depends on the container policy:

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600): 70
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 652): 50
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 653): 50
- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_message_writer** (p. 656): 50

[range] [0, 100]

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_update_period (p. 1618), **com.rti.↩ dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_increase_factor** (p. 1618)

8.290.2.25 multicast_resend_threshold

```
int multicast_resend_threshold
```

The minimum number of requesting readers needed to trigger a multicast resend.

Given readers with multicast destinations, when a reader NACKs for samples to be resent, the writer can either resend them over unicast or multicast. In order for the writer to resend over multicast, this threshold is the minimum number of readers of the same multicast group that the writer must receive NACKs from within a single response-delay. This allows the writer to coalesce near-simultaneous unicast resends into a multicast resend. Note that a threshold of 1 means that all resends will be sent over multicast, if available.

[default] 2

[range] [>= 1]

8.290.2.26 enable_multicast_periodic_heartbeat

```
boolean enable_multicast_periodic_heartbeat
```

Whether periodic heartbeat messages are sent over multicast.

When enabled, if a reader has a multicast destination, then the writer will send its periodic HEARTBEAT messages to that destination. Otherwise, if not enabled or the reader does not have a multicast destination, the writer will send its periodic HEARTBEATs over unicast.

[default] com.rti.dds.infrastructure.false

8.290.2.27 disable_repair_piggyback_heartbeat

```
boolean disable_repair_piggyback_heartbeat
```

Prevents piggyback heartbeats from being sent with repair samples.

When samples are repaired, the **com.rti.dds.publication.DataWriter** (p. 553) resends **com.rti.dds.infrastructure.↩ RtpsReliableWriterProtocol_t.max_bytes_per_nack_response** (p. 1614) bytes and a piggyback heartbeat with each message. You can configure the **com.rti.dds.publication.DataWriter** (p. 553) to not send the piggyback heartbeat and instead rely on the **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.late_joiner_heartbeat_↩ period** (p. 1609) to control the throughput used to repair samples. This field is mutable only for **com.rti.dds.↩ infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 600). **[default]** com.rti.dds.infrastructure.false

8.291 RtpsReservedPortKind Class Reference

RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.

Static Public Attributes

- static final int **BUILTIN_UNICAST** = 0x0001 << 0

  *Select the **metatraffic** unicast port.*
- static final int **BUILTIN_MULTICAST** = 0x0001 << 1

  *Select the **metatraffic** multicast port.*
- static final int **USER_UNICAST** = 0x0001 << 2

  *Select the **usertraffic** unicast port.*
- static final int **USER_MULTICAST** = 0x0001 << 3

  *Select the **usertraffic** multicast port.*
- static final int **MASK_DEFAULT** = **BUILTIN_UNICAST** | **BUILTIN_MULTICAST** | **USER_UNICAST**

  *The default value of **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask** (p. 1992).*
- static final int **MASK_NONE**

  No bits are set.
- static final int **MASK_ALL**

  All bits are set.

8.291.1 Detailed Description

RTPS reserved port kind, used to identify the types of ports that can be reserved on domain participant enable.

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask (p. 1992)

8.291.2 Member Data Documentation

8.291.2.1 BUILTIN_UNICAST

```
final int BUILTIN_UNICAST = 0x0001 << 0  [static]
```

Select the **metatraffic** unicast port.

8.291.2.2 BUILTIN_MULTICAST

```
final int BUILTIN_MULTICAST = 0x0001 << 1  [static]
```

Select the **metatraffic** multicast port.

8.291.2.3 USER_UNICAST

```
final int USER_UNICAST = 0x0001 << 2  [static]
```

Select the **usertraffic** unicast port.

8.291.2.4 USER_MULTICAST

```
final int USER_MULTICAST = 0x0001 << 3  [static]
```

Select the **usertraffic** multicast port.

8.292 RtpsWellKnownPorts_t Class Reference

RTPS well-known port mapping configuration.

Inherits Struct.

Public Attributes

- int **port_base**
  The base port offset.
- int **domain_id_gain**
  Tunable domain gain parameter.
- int **participant_id_gain**
  Tunable participant gain parameter.
- int **builtin_multicast_port_offset**
  *Additional offset for **metatraffic** multicast port.*
- int **builtin_unicast_port_offset**
  *Additional offset for **metatraffic** unicast port.*
- int **user_multicast_port_offset**
  *Additional offset for **usertraffic** multicast port.*
- int **user_unicast_port_offset**
  *Additional offset for **usertraffic** unicast port.*

Static Public Attributes

- static final **RtpsWellKnownPorts_t RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS**
  Assign to use well-known port mappings which are compatible with previous versions of the RTI Connext middleware.
- static final **RtpsWellKnownPorts_t INTEROPERABLE_RTPS_WELL_KNOWN_PORTS**
  Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

8.292.1 Detailed Description

RTPS well-known port mapping configuration.

RTI Connext uses the RTPS wire protocol. The discovery protocols defined by RTPS rely on well-known ports to initiate discovery. These well-known ports define the multicast and unicast ports on which a Participant will listen for discovery **metatraffic** from other Participants. The discovery metatraffic contains all the information required to establish the presence of remote DDS entities in the network.

The well-known ports are defined by RTPS in terms of port mapping expressions with several tunable parameters, which allow you to customize what network ports are used by RTI Connext. These parameters are exposed in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622). In order for all Participants in a system to correctly discover each other, it is important that they all use the same port mapping expressions.

The actual port mapping expressions, as defined by the RTPS specification, can be found below. In addition to the parameters listed in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622), the port numbers depend on:

- `domain_id`, as specified in **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 765)
- `participant_id`, as specified using **com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id** (p. 1990)

The `domain_id` parameter ensures no port conflicts exist between Participants belonging to different domains. This also means that discovery metatraffic in one domain is not visible to Participants in a different domain. The `participant_id` parameter ensures that unique unicast port numbers are assigned to Participants belonging to the same domain on a given host.

The *metatraffic_unicast_port* is used to exchange discovery metatraffic using unicast.

```
metatraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + b
```

The *metatraffic_multicast_port* is used to exchange discovery metatraffic using multicast. The corresponding multicast group addresses are specified via **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 667) on a **com.rti.dds.domain.DomainParticipant** (p. 670) entity.

```
metatraffic_multicast_port = port_base + (domain_id_gain * domain_id) + builtin_multicast_port_offset
```

RTPS also defines the *default* multicast and unicast ports on which DataReaders and DataWriters receive **usertraffic**. These default ports can be overridden using the **com.rti.dds.subscription.DataReaderQos.multicast** (p. 524), **com.rti.dds.subscription.DataReaderQos.unicast** (p. 524), or by the **com.rti.dds.publication.DataWriterQos.unicast** (p. 620) QoS policies.

The *usertraffic_unicast_port* is used to exchange user data using unicast.

```
usertraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + u
```

The *usertraffic_multicast_port* is used to exchange user data using multicast. The corresponding multicast group addresses can be configured using **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1853).

```
usertraffic_multicast_port = port_base + (domain_id_gain * domain_id) + user_multicast_port_offset
```

By default, the port mapping parameters are configured to compliant with OMG's DDS Interoperability Wire Protocol (see also **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_↩ PORTS** (p. 283)).

The OMG's DDS Interoperability Wire Protocol compliant port mapping parameters are *not* backwards compatible with previous versions of the RTI Connext middleware.

When modifying the port mapping parameters, care must be taken to avoid port aliasing. This would result in undefined discovery behavior. The chosen parameter values will also determine the maximum possible number of domains in the system and the maximum number of participants per domain. Additionally, any resulting mapped port number must be within the range imposed by the underlying transport. For example, for UDPv4, this range typically equals [1024 - 65535].

Note: On Windows, you should avoid using ports 49152 through 65535 for inbound traffic. RTI Connext's ephemeral ports (see "Ports Used for Communication" in the `User's Manual`) may be within that range (see `https↩ ://msdn.microsoft.com/en-us/library/windows/desktop/ms737550(v=vs.85).aspx`). With the default RtpsWellKnownPorts settings, port 49152 corresponds to domain ID 167, so using domain IDs 168 through 232 on Windows introduces the risk of a port collision and failure to create the Domain Participant when using multicast discovery. You may see this error:

RTIOsapiSocket_bindWithIP:OS bind() failure, error 0X271D: An attempt was made to access a socket in a way forbidden by its access permissions.

QoS:

com.rti.dds.infrastructure.WireProtocolQosPolicy (p. 1986)

8.292.2 Member Data Documentation

8.292.2.1 port_base

```
int port_base
```

The base port offset.

All mapped well-known ports are offset by this value.

[default] 7400

[range] [>= 1], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.2 domain_id_gain

```
int domain_id_gain
```

Tunable domain gain parameter.

Multiplier of the domain_id. Together with participant_id_gain, it determines the highest domain_id and participant_id allowed on this network.

In general, there are two ways to setup domain_id_gain and participant_id_gain parameters.

If domain_id_gain > participant_id_gain, it results in a port mapping layout where all **com.rti.dds.domain.DomainParticipant** (p. 670) instances within a single domain occupy a consecutive range of domain_id_gain ports. Precisely, all ports occupied by the domain fall within:

```
(port_base + (domain_id_gain * domain_id))
```

and:

```
(port_base + (domain_id_gain * (domain_id + 1)) - 1)
```

Under such a case, the highest domain_id is limited only by the underlying transport's maximum port. The highest participant_id, however, must satisfy:

```
max_participant_id < (domain_id_gain / participant_id_gain)
```

On the contrary, if domain_id_gain <= participant_id_gain, it results in a port mapping layout where a given domain's **com.rti.dds.domain.DomainParticipant** (p. 670) instances occupy ports spanned across the entire valid port range allowed by the underlying transport. For instance, it results in the following potential mapping:

Mapped Port	Domain Id	Participant ID
higher port number	Domain Id = 1	Participant ID = 2
	Domain Id = 0	Participant ID = 2
	Domain Id = 1	Participant ID = 1
	Domain Id = 0	Participant ID = 1
	Domain Id = 1	Participant ID = 0
lower port number	Domain Id = 0	Participant ID = 0

Under this case, the highest participant_id is limited only by the underlying transport's maximum port. The highest domain_id, however, must satisfy:

```
max_domain_id < (participant_id_gain / domain_id_gain)
```

Additionally, domain_id_gain also determines the range of the port-specific offsets.

```
domain_id_gain > abs(builtin_multicast_port_offset - user_multicast_port_offset)
```

```
domain_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

Violating this may result in port aliasing and undefined discovery behavior.

[default] 250

[range] [> 0], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.3 participant_id_gain

```
int participant_id_gain
```

Tunable participant gain parameter.

Multiplier of the `participant_id`. See **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.domain_id_gain** (p. 1624) for its implications on the highest `domain_id` and `participant_id` allowed on this network.

Additionally, `participant_id_gain` also determines the range of `builtin_unicast_port_offset` and `user_unicast_port_offset`.

```
participant_id_gain > abs(builtin_unicast_port_offset - user_unicast_port_offset)
```

[default] 2

[range] [> 0], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.4 builtin_multicast_port_offset

```
int builtin_multicast_port_offset
```

Additional offset for **metatraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 0

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.5 builtin_unicast_port_offset

```
int builtin_unicast_port_offset
```

Additional offset for **metatraffic** unicast port.

It must be unique from other port-specific offsets.

[default] 10

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.6 user_multicast_port_offset

```
int user_multicast_port_offset
```

Additional offset for **usertraffic** multicast port.

It must be unique from other port-specific offsets.

[default] 1

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.292.2.7 user_unicast_port_offset

```
int user_unicast_port_offset
```

Additional offset for **usertraffic** unicast port.

It must be unique from other port-specific offsets.

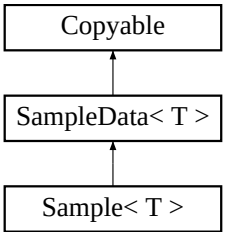
[default] 11

[range] [>= 0], but resulting ports must be within the range imposed by the underlying transport.

8.293 Sample< T > Interface Template Reference

A data sample and related info received from the middleware.

Inheritance diagram for Sample< T >:



Classes

- interface **Iterator**

  Provides access to a collection of middleware-loaned samples.

Public Member Functions

- **SampleInfo getInfo** ()

  Gets the sample information.
- **SampleIdentity_t getRelatedIdentity** ()

  Gets the identity of a sample that is related to this one.

8.293.1 Detailed Description

A data sample and related info received from the middleware.

Samples contain data received from the middleware and information associated to the data.

Template Parameters

T	The data type that this sample contains

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.receiveReply(Sample<TRep>,Duration_t) (p. 1569)

com.rti.connext.requestreply.Requester<TReq,TRep>.takeReply(Sample<TRep>) (p. 1572)

com.rti.connext.requestreply.Replier<TReq,TRep>.receiveRequest(Sample<TReq>,Duration_t) (p. 1547)

8.293.2 Member Function Documentation

8.293.2.1 getInfo()

```
SampleInfo getInfo ( )
```

Gets the sample information.

See also

com.rti.dds.subscription.SampleInfo (p. 1634)

Basic Requester example (p. 148)

8.293.2.2 getRelatedIdentity()

```
SampleIdentity_t getRelatedIdentity ( )
```

Gets the identity of a sample that is related to this one.

When a com.rti.connext.requestreply.Requester<TReq,TRep> receives a reply, the reply **Sample** (p. 1627) contains the identity of the related request.

Returns

The identity of another sample that is related to this one, or com.rti.dds.infrastructure.SampleIdentity_t.Sample↩ Identity_t.UNKNOWN_SAMPLE_IDENTITY if there is not a related sample.

See also

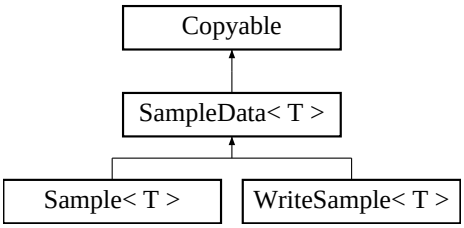
com.rti.connext.requestreply.Requester<TReq,TRep>.receiveReply(Sample<TRep>,Duration_t) (p. 1569)

Correlating requests and replies (p. 150)

8.294 SampleData< T > Interface Template Reference

Sample (p. 1627) base type that contains data and has an identity.

Inheritance diagram for SampleData< T >:



Public Member Functions

- T **getData** ()

  Gets the data this sample contains.
- **SampleIdentity_t getIdentity** ()

  Gets the identity of this sample.

8.294.1 Detailed Description

Sample (p. 1627) base type that contains data and has an identity.

This is the base interface for samples received by the middleware (com.rti.connext.infrastructure.Sample<T>) and samples used to write (com.rti.connext.infrastructure.WriteSample<T>)

Template Parameters

T	The data type that this sample contains

8.294.2 Member Function Documentation

8.294.2.1 getData()

```
T getData ( )
```

Gets the data this sample contains.

See also

Basic Requester example (p. 148)

Referenced by **Replier< TReq, TRep >.sendReply()**, and **Requester< TReq, TRep >.sendRequest()**.

8.294.2.2 getIdentity()

```
SampleIdentity_t getIdentity ( )
```

Gets the identity of this sample.

The identity is assigned by the middleware upon reception.

See also

Correlating requests and replies (p. 150)

8.295 SampleFlagBits Class Reference

Type to identify the sample flags reserved by RTI.

Static Public Attributes

- static final int **REDELIVERED_SAMPLE**

  Indicates that a sample has been redelivered by RTI Queuing Service.
- static final int **INTERMEDIATE_REPLY_SEQUENCE_SAMPLE**

  Indicates that a response sample is not the last response sample for a given request. This bit is usually set by Connext Repliers sending multiple responses for a request.
- static final int **REPLICATE_SAMPLE**

  Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.
- static final int **LAST_SHARED_READER_QUEUE_SAMPLE**

  Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer DataReader.
- static final int **INTERMEDIATE_TOPIC_QUERY_SAMPLE**

  Indicates that a sample for a TopicQuery will be followed by more samples.
- static final int **WRITER_REMOVED_BATCH_SAMPLE** = (0x00000001) << 5

  This flag will be set if the sample was accepted into the DataReader queue even though it was marked by the DataWriter as removed.
- static final int **DISCOVERY_SERVICE_SAMPLE** = (0x00000001) << 6

  This flag will be set if the sample was sent by Cloud Discovery Service.

8.295.1 Detailed Description

Type to identify the sample flags reserved by RTI.

8.295.2 Member Data Documentation

8.295.2.1 REDELIVERED_SAMPLE

```
final int REDELIVERED_SAMPLE  [static]
```

Indicates that a sample has been redelivered by RTI Queuing Service.

8.295.2.2 INTERMEDIATE_REPLY_SEQUENCE_SAMPLE

```
final int INTERMEDIATE_REPLY_SEQUENCE_SAMPLE  [static]
```

Indicates that a response sample is not the last response sample for a given request. This bit is usually set by Connext Repliers sending multiple responses for a request.

8.295.2.3 REPLICATE_SAMPLE

```
final int REPLICATE_SAMPLE  [static]
```

Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.

8.295.2.4 LAST_SHARED_READER_QUEUE_SAMPLE

```
final int LAST_SHARED_READER_QUEUE_SAMPLE  [static]
```

Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer DataReader.

8.295.2.5 INTERMEDIATE_TOPIC_QUERY_SAMPLE

```
final int INTERMEDIATE_TOPIC_QUERY_SAMPLE  [static]
```

Indicates that a sample for a TopicQuery will be followed by more samples.

This flag only applies to samples that have been published as a response to a **com.rti.dds.subscription.TopicQuery** (p. 1830).

When this bit is not set and **com.rti.dds.subscription.SampleInfo.topic_query_guid** (p. 1646) is different from **com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134), this sample is the last sample for that TopicQuery coming from the DataWriter identified by **com.rti.dds.subscription.SampleInfo.original_publication_virtual_guid** (p. 1644).

8.295.2.6 WRITER_REMOVED_BATCH_SAMPLE

```
final int WRITER_REMOVED_BATCH_SAMPLE = (0x00000001) << 5  [static]
```

This flag will be set if the sample was accepted into the DataReader queue even though it was marked by the DataWriter as removed.

A sample can be marked as removed by the DataWriter in a batch when it is replaced due to the com.↩ rti.dds.infrastructure.HistoryQosPolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS **com.rti.dds.↩ infrastructure.HistoryQosPolicy** (p. 1144) QoS or because the duration in **com.rti.dds.infrastructure.Lifespan↩ QosPolicy** (p. 1234) was reached.

If the DataReader sets the property "dds.data_reader.accept_writer_removed_batch_samples" to true, the removed sample will be accepted into the DataReader queue and this flag will be set.

8.295.2.7 DISCOVERY_SERVICE_SAMPLE

```
final int DISCOVERY_SERVICE_SAMPLE = (0x00000001) << 6  [static]
```

This flag will be set if the sample was sent by Cloud Discovery Service.

The samples sent by Cloud Discovery Service are participant announcement samples on the ParticipantBuiltinTopic.

8.296 SampleIdentity_t Class Reference

Type definition for a Sample Identity.

Inherits Struct.

Public Member Functions

- **SampleIdentity_t** (**SampleIdentity_t** other)

  Copy constructor.

Public Attributes

- final **GUID_t writer_guid** = new **GUID_t**(**GUID_t.GUID_AUTO**)

  16-byte identifier identifying the virtual GUID.
- final **SequenceNumber_t sequence_number**

  monotonically increasing 64-bit integer that identifies the sample in the data source.

Static Public Attributes

- static final **SampleIdentity_t AUTO_SAMPLE_IDENTITY**

  The AUTO sample identity.
- static final **SampleIdentity_t UNKNOWN_SAMPLE_IDENTITY**

  An invalid or unknown sample identity.

8.296.1 Detailed Description

Type definition for a Sample Identity.

A SampleIdentity defines a pair (Virtual Writer GUID, Sequence Number) that uniquely identifies a sample within a DDS domain and a Topic.

8.296.2 Constructor & Destructor Documentation

8.296.2.1 SampleIdentity_t()

```
SampleIdentity_t (
            SampleIdentity_t other )
```

Copy constructor.

8.296.3 Member Data Documentation

8.296.3.1 AUTO_SAMPLE_IDENTITY

```
final SampleIdentity_t AUTO_SAMPLE_IDENTITY [static]
```

The AUTO sample identity.

Special com.rti.dds.infrastructure.SampleIdentity_t.SampleIdentity_t.AUTO_SAMPLE_IDENTITY value {**com.rti.dds.infrastructure.GUID_t.GUID_AUTO** (p. 1134), com.rti.dds.infrastructure.SequenceNumber_t.AUTO}

Referenced by **Requester< TReq, TRep >.sendRequest()**.

8.296.3.2 UNKNOWN_SAMPLE_IDENTITY

```
final SampleIdentity_t UNKNOWN_SAMPLE_IDENTITY [static]
```

An invalid or unknown sample identity.

Special com.rti.dds.infrastructure.SampleIdentity_t.SampleIdentity_t.UNKNOWN_SAMPLE_IDENTITY value {**com.↩ rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134), com.rti.dds.infrastructure.SequenceNumber_t.UNKNOWN}

8.296.3.3 writer_guid

```
final GUID_t writer_guid = new GUID_t( GUID_t.GUID_AUTO)
```

16-byte identifier identifying the virtual GUID.

8.296.3.4 sequence_number

```
final SequenceNumber_t sequence_number
```

Initial value:

```
=
        new SequenceNumber_t(SequenceNumber_t.AUTO_SEQUENCE_NUMBER)
```

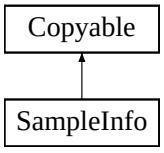
monotonically increasing 64-bit integer that identifies the sample in the data source.

Referenced by **Requester< TReq, TRep >.readReplies()**, **Requester< TReq, TRep >.readReply()**, **Requester< TReq, TRep >.takeReplies()**, **Requester< TReq, TRep >.takeReply()**, and **Requester< TReq, TRep >.waitFor↩ Replies()**.

8.297 SampleInfo Class Reference

Information that accompanies each sample that is `read` or `taken`.

Inheritance diagram for SampleInfo:



Public Member Functions

- Object **copy_from** (Object other)

Public Attributes

- int **sample_state**

  The sample state of the sample.
- int **view_state**

  The view state of the instance.
- int **instance_state**

  The instance state of the instance.
- final **Time_t source_timestamp**

  The timestamp when the sample was written by a DataWriter.
- final **InstanceHandle_t instance_handle**

  Identifies locally the corresponding instance.
- final **InstanceHandle_t publication_handle**

  Identifies locally the DataWriter that modified the instance.
- int **disposed_generation_count**

  The disposed generation count of the instance at the time of sample reception.
- int **no_writers_generation_count**

  The no writers generation count of the instance at the time of sample reception.
- int **sample_rank**

  The sample rank of the sample.
- int **generation_rank**

  The generation rank of the sample.
- int **absolute_generation_rank**

  The absolute generation rank of the sample.
- boolean **valid_data**

  Indicates whether the `DataSample` *contains data or else it is only used to communicate a change in the* `instance_state` *of the instance.*
- final **Time_t reception_timestamp**

  *<<**extension**>> (p. 155) The timestamp when the sample was committed by a **DataReader** (p. 450).*
- final **SequenceNumber_t publication_sequence_number**

  *<<**extension**>> (p. 155) The publication sequence number.*
- final **SequenceNumber_t reception_sequence_number**

  *<<**extension**>> (p. 155) The reception sequence number when sample was committed by a **DataReader** (p. 450)*
- final **GUID_t original_publication_virtual_guid**

  *<<**extension**>> (p. 155) The original publication virtual GUID.*
- final **SequenceNumber_t original_publication_virtual_sequence_number**

  *<<**extension**>> (p. 155) The original publication virtual sequence number.*
- final **GUID_t related_original_publication_virtual_guid**

  *<<**extension**>> (p. 155) The original publication virtual GUID of a related sample.*
- final **SequenceNumber_t related_original_publication_virtual_sequence_number**

  *<<**extension**>> (p. 155) The original publication virtual sequence number of a related sample.*
- int **flag**

  *<<**extension**>> (p. 155) Flags associated with the sample.*
- **GUID_t source_guid**

  *<<**extension**>> (p. 155) The application logical data source associated with the sample.*
- **GUID_t related_source_guid**

  *<<**extension**>> (p. 155) The application logical data source that is related to the sample.*

- **GUID_t related_subscription_guid**

  <<***extension***>> *(*p. *155) The related_reader_guid associated with the sample.*
- final **GUID_t topic_query_guid**

  <<***extension***>> *(*p. *155) The GUID of the* ***com.rti.dds.subscription.TopicQuery*** *(*p. *1830) that is related to the sample.*
- **CoherentSetInfo_t coherent_set_info**

  <<***extension***>> *(*p. *155) The information about the coherent set that this sample is a part of.*

8.297.1 Detailed Description

Information that accompanies each sample that is `read` or `taken`.

8.297.2 Interpretation of the SampleInfo

The **com.rti.dds.subscription.SampleInfo** (p. 1634) contains information pertaining to the associated `Data` instance sample including:

- the `sample_state` of the `Data` value (i.e., if it has already been read or not)
- the `view_state` of the related instance (i.e., if the instance is new or not)
- the `instance_state` of the related instance (i.e., if the instance is alive or not)
- **com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643) flag. This flag indicates whether there is data associated with the sample. Some samples do not contain data indicating only a change on the `instance_state` of the corresponding instance.
- The values of `disposed_generation_count` and `no_writers_generation_count` for the related instance at the time the sample was received. These counters indicate the number of times the instance had become ALIVE (with `instance_state`= **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161)) at the time the sample was received.
- The `sample_rank` and `generation_rank` of the sample within the returned sequence. These ranks provide a preview of the samples that follow within the sequence returned by the `read` or `take` operations.
- The `absolute_generation_rank` of the sample within the **com.rti.dds.subscription.DataReader** (p. 450). This rank provides a preview of what is available within the **com.rti.dds.subscription.DataReader** (p. 450).
- The `source_timestamp` of the sample. This is the timestamp provided by the **com.rti.dds.publication.DataWriter** (p. 553) at the time the sample was produced.

8.297.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count

For each instance, RTI Connext internally maintains two counts, the **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1641) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1641), relative to each **DataReader** (p. 450):

- The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1641) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1641) are initialized to zero when the **com.rti.dds.subscription.DataReader** (p. 450) first detects the presence of a never-seen-before instance.

- The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1641) is incremented each time the instance_state of the corresponding instance changes from **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) to **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161).

- The **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1641) is incremented each time the instance_state of the corresponding instance changes from **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) to **com.rti.dds.subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161).

- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1641) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1641) available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) capture a snapshot of the corresponding counters at the time the sample was received.

8.297.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **com.rti.dds.subscription.SampleInfo.sample_rank** (p. 1642) and **com.rti.dds.subscription.SampleInfo.generation_rank** (p. 1642) available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) are computed based solely on the actual samples in the ordered collection returned by read or take.

- The **com.rti.dds.subscription.SampleInfo.sample_rank** (p. 1642) indicates the number of samples of the same instance that follow the current one in the collection.

- The **com.rti.dds.subscription.SampleInfo.generation_rank** (p. 1642) available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that appears in the returned Collection (MRSIC). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the reception of MRSIC.

- These 'generation ranks' are reset to zero when the instance resource is reclaimed.

The **com.rti.dds.subscription.SampleInfo.generation_rank** (p. 1642) is computed using the formula:

```
generation_rank = (MRSIC.disposed_generation_count
                       + MRSIC.no_writers_generation_count)
                  - (S.disposed_generation_count
                       + S.no_writers_generation_count)
```

The **com.rti.dds.subscription.SampleInfo.absolute_generation_rank** (p. 1642) available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that the middleware has received (MRS). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the time when the read or take was called.

```
absolute_generation_rank = (MRS.disposed_generation_count
                                 + MRS.no_writers_generation_count)
                           - (S.disposed_generation_count
                                 + S.no_writers_generation_count)
```

8.297.5 Interpretation of the SampleInfo counters and ranks

These counters and ranks allow the application to distinguish samples belonging to different "generations" of the instance. Note that it is possible for an instance to transition from not-alive to alive (and back) several times before the application accesses the data by means of read or take. In this case, the returned collection may contain samples that cross generations (i.e. some samples were received before the instance became not-alive, other after the instance re-appeared again). Using the information in the **com.rti.dds.subscription.SampleInfo** (p. 1634), the application can anticipate what other information regarding the same instance appears in the returned collection, as well as in the infrastructure and thus make appropriate decisions.

For example, an application desiring to only consider the most current sample for each instance would only look at samples with sample_rank == 0. Similarly, an application desiring to only consider samples that correspond to the latest generation in the collection will only look at samples with generation_rank == 0. An application desiring only samples pertaining to the latest generation available will ignore samples for which absolute_generation_rank != 0. Other application-defined criteria may also be used.

See also

com.rti.dds.subscription.SampleStateKind (p. 1663), **com.rti.dds.subscription.InstanceStateKind** (p. 1160), **com.rti.dds.subscription.ViewStateKind** (p. 1970), **com.rti.dds.subscription.SampleInfo.valid_data** (p. 1643)

"Statechart of the \p instance_state and \p view_state of a single instance"

8.297.6 Member Function Documentation

8.297.6.1 copy_from()

```
Object copy_from (
             Object other )
```

Implementation of the `Copyable` interface.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

References **SampleInfo.absolute_generation_rank**, **SampleInfo.coherent_set_info**, **InstanceHandle_t.copy_from()**, **SampleInfo.disposed_generation_count**, **SampleInfo.flag**, **SampleInfo.generation_rank**, **SampleInfo.instance_handle**, **SampleInfo.instance_state**, **Time_t.nanosec**, **SampleInfo.no_writers_generation_count**, **SampleInfo.original_publication_virtual_guid**, **SampleInfo.original_publication_virtual_sequence_number**, **SampleInfo.publication_handle**, **SampleInfo.publication_sequence_number**, **SampleInfo.reception_sequence_number**, **SampleInfo.reception_timestamp**, **SampleInfo.related_original_publication_virtual_guid**, **SampleInfo.related_original_publication_virtual_sequence_number**, **SampleInfo.related_source_guid**, **SampleInfo.related_subscription_guid**, **SampleInfo.sample_rank**, **SampleInfo.sample_state**, **Time_t.sec**, **SampleInfo.source_guid**, **SampleInfo.source_timestamp**, **SampleInfo.topic_query_guid**, **SampleInfo.valid_data**, and **SampleInfo.view_state**.

8.297.7 Member Data Documentation

8.297.7.1 sample_state

```
int sample_state
```

The sample state of the sample.

Indicates whether or not the corresponding data sample has already been read.

See also

com.rti.dds.subscription.SampleStateKind (p. 1663)

Referenced by **SampleInfo.copy_from()**.

8.297.7.2 view_state

```
int view_state
```

The view state of the instance.

Indicates whether the **com.rti.dds.subscription.DataReader** (p. 450) has already seen samples for the most-current generation of the related instance.

See also

com.rti.dds.subscription.ViewStateKind (p. 1970)

Referenced by **SampleInfo.copy_from()**.

8.297.7.3 instance_state

```
int instance_state
```

The instance state of the instance.

Indicates whether the instance is currently in existence or, if it has been disposed, the reason why it was disposed.

See also

com.rti.dds.subscription.InstanceStateKind (p. 1160)

Referenced by **SampleInfo.copy_from()**.

8.297.7.4 source_timestamp

```
final  Time_t source_timestamp
```

The timestamp when the sample was written by a DataWriter.

Referenced by **SampleInfo.copy_from()**.

8.297.7.5 instance_handle

```
final InstanceHandle_t instance_handle
```

Identifies locally the corresponding instance.

The handle is equal to **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156) for unkeyed topics.

Referenced by **SampleInfo.copy_from()**.

8.297.7.6 publication_handle

```
final InstanceHandle_t publication_handle
```

Identifies locally the DataWriter that modified the instance.

The `publication_handle` is the same **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1152) that is returned by the operation **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 465) and can also be used as a parameter to the operation **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466).

Referenced by **SampleInfo.copy_from()**.

8.297.7.7 disposed_generation_count

```
int disposed_generation_count
```

The disposed generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **com.rti.↩ dds.subscription.InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1161) to **com.rti.dds.↩ subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161). The counter is reset when the instance resource is reclaimed (removed from the **DataReader** (p. 450) cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1637)
Interpretation of the SampleInfo counters and ranks (p. 1638)

Referenced by **SampleInfo.copy_from()**.

8.297.7.8 no_writers_generation_count

```
int no_writers_generation_count
```

The no writers generation count of the instance at the time of sample reception.

Indicates how many times the `instance_state` of the corresponding instance changed from **com.rti.dds.↩ subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161) to **com.rti.dds.↩ subscription.InstanceStateKind.ALIVE_INSTANCE_STATE** (p. 1161). The counter is reset when the instance resource is reclaimed (removed from the **DataReader** (p. 450) cache).

See also

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1637)
Interpretation of the SampleInfo counters and ranks (p. 1638)

Referenced by **SampleInfo.copy_from()**.

8.297.7.9 sample_rank

```
int sample_rank
```

The sample rank of the sample.

Indicates the number of samples related to the same instance that follow in the collection returned by `read` or `take`.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1637)
Interpretation of the SampleInfo counters and ranks (p. 1638)

Referenced by **SampleInfo.copy_from()**.

8.297.7.10 generation_rank

```
int generation_rank
```

The generation rank of the sample.

Indicates the generation difference (number of times the instance was NOT_ALIVE and become alive again) between the time the sample was received and the time the most recent sample in the collection related to the same instance was received.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1637)
Interpretation of the SampleInfo counters and ranks (p. 1638)

Referenced by **SampleInfo.copy_from()**.

8.297.7.11 absolute_generation_rank

```
int absolute_generation_rank
```

The absolute generation rank of the sample.

Indicates the generation difference (number of times the instance was disposed and become alive again) between the time the sample was received, and the time the most recent sample (which may not be in the returned collection) related to the same instance was received.

See also

Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank (p. 1637)
Interpretation of the SampleInfo counters and ranks (p. 1638)

Referenced by **SampleInfo.copy_from()**.

8.297.7.12 valid_data

```
boolean valid_data
```

Indicates whether the `DataSample` contains data or else it is only used to communicate a change in the `instance_state` of the instance.

Normally each `DataSample` contains both a **com.rti.dds.subscription.SampleInfo** (p. 1634) and some Data. However there are situations where a `DataSample` contains only the **com.rti.dds.subscription.SampleInfo** (p. 1634) and does not have any associated data. This occurs when the RTI Connext notifies the application of a change of state for an instance that was caused by some internal mechanism (such as a timeout) for which there is no associated data. An example of this situation is when the RTI Connext detects that an instance has no writers and changes the corresponding instance_state to **com.rti.dds.subscription.InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1161).

The application can distinguish whether a particular `DataSample` has data by examining the value of the **`com.rti.dds.subscription.SampleInfo.valid_data`** (p. 1643). If this flag is set to com.rti.dds.infrastructure.true, then the `DataSample` contains valid Data. If the flag is set to com.rti.dds.infrastructure.false, the `DataSample` contains no Data.

To ensure correctness and portability, the `valid_data` flag must be examined by the application prior to accessing the Data associated with the `DataSample` and if the flag is set to com.rti.dds.infrastructure.false, the application should not access the Data associated with the `DataSample`, that is, the application should access only the **com.rti.dds.subscription.SampleInfo** (p. 1634).

Referenced by **SampleInfo.copy_from()**.

8.297.7.13 reception_timestamp

```
final  Time_t reception_timestamp
```

<<***extension***>> *(*p. *155)* The timestamp when the sample was committed by a **DataReader** (p. 450).

Referenced by **SampleInfo.copy_from()**.

8.297.7.14 publication_sequence_number

```
final  SequenceNumber_t publication_sequence_number
```

<<***extension***>> *(*p. *155)* The publication sequence number.

Referenced by **SampleInfo.copy_from()**.

8.297.7.15 reception_sequence_number

```
final  SequenceNumber_t reception_sequence_number
```

<<***extension***>> *(*p. *155)* The reception sequence number when sample was committed by a **DataReader** (p. 450)

Referenced by **SampleInfo.copy_from()**.

8.297.7.16 original_publication_virtual_guid

```
final  GUID_t original_publication_virtual_guid
```

<<***extension***>> *(*p. *155)* The original publication virtual GUID.

If the **com.rti.dds.infrastructure.PresentationQosPolicy::access_scope** (p. 1382) of the **com.rti.dds.publication.↩ Publisher** (p. 1466) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicy↩ AccessScopeKind.GROUP_PRESENTATION_QOS, this field contains the **com.rti.dds.publication.Publisher** (p. 1466) virtual GUID that uniquely identifies the DataWriter group.

See also

com.rti.connext.infrastructure.Sample<T>.getIdentity() (p. 1630)

Referenced by **SampleInfo.copy_from()**.

8.297.7.17 original_publication_virtual_sequence_number

```
final SequenceNumber_t original_publication_virtual_sequence_number
```

<<***extension***>> *(*p. *155)* The original publication virtual sequence number.

If the **com.rti.dds.infrastructure.PresentationQosPolicy::access_scope** (p. 1382) of the **com.rti.dds.publication.Publisher** (p. 1466) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS, this field contains the **com.rti.dds.publication.Publisher** (p. 1466) virtual sequence number that uniquely identifies a sample within the DataWriter group.

See also

com.rti.connext.infrastructure.Sample<T>.getIdentity() (p. 1630)

Referenced by **SampleInfo.copy_from()**.

8.297.7.18 related_original_publication_virtual_guid

```
final GUID_t related_original_publication_virtual_guid
```

<<***extension***>> *(*p. *155)* The original publication virtual GUID of a related sample.

See also

com.rti.connext.infrastructure.Sample<T>.getRelatedIdentity() (p. 1628)

Referenced by **SampleInfo.copy_from()**.

8.297.7.19 related_original_publication_virtual_sequence_number

```
final SequenceNumber_t related_original_publication_virtual_sequence_number
```

<<***extension***>> *(*p. *155)* The original publication virtual sequence number of a related sample.

See also

com.rti.connext.infrastructure.Sample<T>.getRelatedIdentity() (p. 1628)

Referenced by **SampleInfo.copy_from()**.

8.297.7.20 flag

```
int flag
```

<<***extension***>> *(*p. *155)* Flags associated with the sample.

The flags can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.flag** (p. 1999) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

Referenced by **SampleInfo.copy_from()**.

8.297.7.21 source_guid

```
GUID_t source_guid
```

<<***extension***>> *(*p. *155)* The application logical data source associated with the sample.

The source_guid can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.source_guid** (p. 2000) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

Referenced by **SampleInfo.copy_from()**.

8.297.7.22 related_source_guid

```
GUID_t related_source_guid
```

<<***extension***>> *(*p. *155)* The application logical data source that is related to the sample.

The related_source_guid can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.related_source↩ _guid** (p. 2000) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

Referenced by **SampleInfo.copy_from()**.

8.297.7.23 related_subscription_guid

```
GUID_t related_subscription_guid
```

<<***extension***>> *(*p. *155)* The related_reader_guid associated with the sample.

The related_reader_guid can be set by using the field **com.rti.dds.infrastructure.WriteParams_t.related_reader↩ _guid** (p. 2001) when writing a sample using the method **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110).

Referenced by **SampleInfo.copy_from()**.

8.297.7.24 topic_query_guid

```
final  GUID_t topic_query_guid
```

<<***extension***>> *(p. 155)* The GUID of the **com.rti.dds.subscription.TopicQuery** (p. 1830) that is related to the sample.

This GUID indicates whether a sample is part of the response to a **com.rti.dds.subscription.TopicQuery** (p. 1830) or a regular ("live") sample:

- If the sample was written for the **TopicQuery** (p. 1830) stream, this field contains the GUID of the target **Topic↩ Query** (p. 1830).
- If the sample was written for the live stream, this field will be set to **com.rti.dds.infrastructure.GUID_t.GUID_↩ UNKNOWN** (p. 1134).

Referenced by **SampleInfo.copy_from()**.

8.297.7.25 coherent_set_info

```
CoherentSetInfo_t coherent_set_info
```

<<***extension***>> *(p. 155)* The information about the coherent set that this sample is a part of.

This field is set for all samples that are part of a coherent set. Coherent sets are initiated using the operation **com.↩ rti.dds.publication.Publisher.begin_coherent_changes** (p. 1483) and finalized using the operation **com.rti.dds.↩ publication.Publisher.end_coherent_changes** (p. 1483).

See also

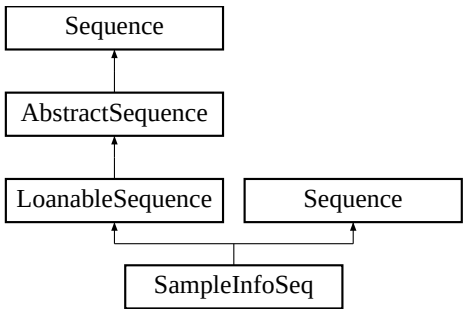
com.rti.dds.publication.Publisher.begin_coherent_changes (p. 1483) for additional information on coherent sets.

Referenced by **SampleInfo.copy_from()**.

8.298 SampleInfoSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.SampleInfo** (p. 1634) > .

Inheritance diagram for SampleInfoSeq:



Public Member Functions

- **SampleInfo get** (int index)

  Returns the element at the specified position in this sequence.

8.298.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.SampleInfo** (p. 1634) > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.298.2 Member Function Documentation

8.298.2.1 get()

```
SampleInfo get (
            int index )
```

Returns the element at the specified position in this sequence.

See also

java.util.List::get(int)

Reimplemented from **LoanableSequence** (p. 1252).

8.299 SampleLostStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of all samples lost across all instances of data published under the **com.rti.dds.topic.Topic** (p. 1807).*
- int **total_count_change**

  The incremental number of samples lost since the last time the listener was called or the status was read.
- **SampleLostStatusKind last_reason**

  Reason for rejecting the last sample rejected.

8.299.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS

8.299.2 Member Data Documentation

8.299.2.1 total_count

```
int total_count
```

Total cumulative count of all samples lost across all instances of data published under the **com.rti.dds.topic.Topic** (p. 1807).

8.299.2.2 total_count_change

```
int total_count_change
```

The incremental number of samples lost since the last time the listener was called or the status was read.

8.299.2.3 last_reason

```
SampleLostStatusKind last_reason
```

Reason for rejecting the last sample rejected.

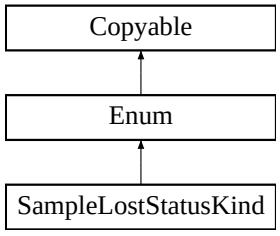
See also

com.rti.dds.subscription.SampleRejectedStatusKind (p. 1659)

8.300 SampleLostStatusKind Class Reference

<<***extension***>> *(p. 155)* Kinds of reasons why a sample was lost.

Inheritance diagram for SampleLostStatusKind:



Static Public Attributes

- static final **SampleLostStatusKind NOT_LOST**

  The sample was not lost.
- static final **SampleLostStatusKind LOST_BY_WRITER**

  *A **com.rti.dds.publication.DataWriter** (p. 553) removed the sample before being received by the **com.rti.dds.subscription.DataReader** (p. 450).*
- static final **SampleLostStatusKind LOST_BY_INSTANCES_LIMIT**

  *A resource limit on the number of instances (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)) was reached.*
- static final **SampleLostStatusKind LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT**

  *A resource limit on the number of remote writers for a single instance from which a **com.rti.dds.subscription.DataReader** (p. 450) may read (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_writers_per_instance** (p. 531)) was reached.*
- static final **SampleLostStatusKind LOST_BY_INCOMPLETE_COHERENT_SET**

  A sample is lost because it is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples are missing.
- static final **SampleLostStatusKind LOST_BY_LARGE_COHERENT_SET**

  *A sample is lost because it is part of a large coherent set. A large coherent set is a coherent set that cannot fit all at once into the **com.rti.dds.subscription.DataReader** (p. 450) queue because resource limits are exceeded.*
- static final **SampleLostStatusKind LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): a resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 450) may store (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532) will trigger a rejection, not a loss, with reason **com.rti.dds.subscription.SampleRejectedStatusKind.REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT** (p. 1661).*
- static final **SampleLostStatusKind LOST_BY_VIRTUAL_WRITERS_LIMIT**

  *A resource limit on the number of virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_virtual_writers** (p. 537)) was reached.*
- static final **SampleLostStatusKind LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT**

  *A resource limit on the number of remote writers per sample (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_writers_per_sample** (p. 538)) was reached.*
- static final **SampleLostStatusKind LOST_BY_AVAILABILITY_WAITING_TIME**

  ***com.rti.dds.infrastructure.AvailabilityQosPolicy.max_data_availability_waiting_time** (p. 345) expired.*
- static final **SampleLostStatusKind LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT**

  *A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **com.rti.dds.subscription.DataReader** (p. 450) may store was reached. (This field is currently not used.)*
- static final **SampleLostStatusKind LOST_BY_OUT_OF_MEMORY**

  A sample was lost because there was not enough memory to store the sample.
- static final **SampleLostStatusKind LOST_BY_UNKNOWN_INSTANCE**

  A received sample was lost because it doesn't contain enough information for the reader to know what instance it is associated with.
- static final **SampleLostStatusKind LOST_BY_DESERIALIZATION_FAILURE**

  A received sample was lost because it could not be deserialized.
- static final **SampleLostStatusKind LOST_BY_DECODE_FAILURE**

*When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): A received sample was lost because it could not be decoded. When using **com.rti.dds.infrastructure.ReliabilityQos↩ PolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), the sample will be rejected, not lost, with reason **com.rti.dds.↩ subscription.SampleRejectedStatusKind.REJECTED_BY_DECODE_FAILURE** (p. 1662).*

- static final **SampleLostStatusKind LOST_BY_SAMPLES_PER_INSTANCE_LIMIT**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples per instance (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_↩ samples_per_instance** (p. 1593)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.↩ RELIABLE_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_↩ samples_per_instance** (p. 1593) will trigger a rejection, not a loss, with reason **com.rti.dds.subscription.Sample↩ RejectedStatusKind.REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT** (p. 1661).*

- static final **SampleLostStatusKind LOST_BY_SAMPLES_LIMIT**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532)↩ : A resource limit on the number of samples (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_↩ samples** (p. 1592)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_↩ RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) will trigger a rejection, not a loss, with reason **com.rti.dds.subscription.SampleRejectedStatusKind.REJECTED_↩ BY_SAMPLES_LIMIT** (p. 1660).*

Additional Inherited Members

8.300.1 Detailed Description

<<***extension***>> *(p. 155)* Kinds of reasons why a sample was lost.

8.300.2 Member Data Documentation

8.300.2.1 NOT_LOST

```
final  SampleLostStatusKind NOT_LOST  [static]
```

Initial value:

```
= new com.rti.dds.subscription.SampleLostStatusKind(
            "NOT_LOST", 0)
```

The sample was not lost.

This constant is an extension to the DDS standard. MONITOR-273 We assign an integer to the names in order to be compatible with the DDSMonitoring types. If new values are added, update DataReaderTest.testGetAndSetSample↩ LostStatus

8.300.2.2 LOST_BY_WRITER

```
final  SampleLostStatusKind LOST_BY_WRITER  [static]
```

Initial value:

```
= new com.rti.dds.subscription.SampleLostStatusKind(
            "LOST_BY_WRITER", 1)
```

A **com.rti.dds.publication.DataWriter** (p. 553) removed the sample before being received by the **com.rti.dds.↩ subscription.DataReader** (p. 450).

This constant is an extension to the DDS standard.

8.300.2.3 LOST_BY_INSTANCES_LIMIT

```
final  SampleLostStatusKind LOST_BY_INSTANCES_LIMIT  [static]
```

Initial value:

```
= new com.rti.dds.subscription.SampleLostStatusKind(
            "LOST_BY_INSTANCES_LIMIT", 2)
```

A resource limit on the number of instances (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)) was reached.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.300.2.4 LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT

```
final  SampleLostStatusKind LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT  [static]
```

Initial value:

```
= new SampleLostStatusKind(
               "LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT", 3)
```

A resource limit on the number of remote writers for a single instance from which a **com.rti.dds.subscription.↩ DataReader** (p. 450) may read (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_↩ writers_per_instance** (p. 531)) was reached.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.5 LOST_BY_INCOMPLETE_COHERENT_SET

```
final  SampleLostStatusKind LOST_BY_INCOMPLETE_COHERENT_SET  [static]
```

Initial value:

```
= new SampleLostStatusKind(
                "LOST_BY_INCOMPLETE_COHERENT_SET", 4)
```

A sample is lost because it is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples are missing.

For example, consider a **com.rti.dds.publication.DataWriter** (p. 553) using com.rti.dds.infrastructure.HistoryQos↩ PolicyKind.HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS with a depth of 1. The DataWriter publishes two samples of the same instance as part of a coherent set 'CS1'; the first sample of 'CS1' is replaced by a new sample before it can be successfully delivered to the **com.rti.dds.subscription.DataReader** (p. 450). In this case, the coherent set containing the two samples is considered incomplete. The new sample, by default, will not be provided to the application, and will be reported as LOST_BY_INCOMPLETE_COHERENT_SET. (You can change this default behavior by setting **com.rti.dds.infrastructure.PresentationQosPolicy.drop_incomplete_coherent_set** (p. 1383) to FALSE. If you do, the new sample will be provided to the application, but it will be marked as part of an incomplete coherent set in the **com.rti.dds.subscription.SampleInfo** (p. 1634) structure.)

This constant is an extension to the DDS standard.

8.300.2.6 LOST_BY_LARGE_COHERENT_SET

```
final  SampleLostStatusKind LOST_BY_LARGE_COHERENT_SET  [static]
```

Initial value:

```
= new SampleLostStatusKind(
                "LOST_BY_LARGE_COHERENT_SET", 5)
```

A sample is lost because it is part of a large coherent set. A large coherent set is a coherent set that cannot fit all at once into the **com.rti.dds.subscription.DataReader** (p. 450) queue because resource limits are exceeded.

For example, if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) on the **DataReader** (p. 450) is 10 and the coherent set has 15 samples for a given instance, the coherent set is a large coherent set that will be considered incomplete.

The resource limits that can lead to large coherent sets are: **com.rti.dds.infrastructure.ResourceLimits↩ QosPolicy.max_samples** (p. 1592), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per↩ _instance** (p. 1593), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592), and **com.↩ rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532).

This constant is an extension to the DDS standard.

8.300.2.7 LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT

```
final  SampleLostStatusKind LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT  [static]
```

Initial value:

```
= new SampleLostStatusKind(
                "LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT", 6)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532)↩ : a resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.**↩ **DataReader** (p. 450) may store (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples**↩ **_per_remote_writer** (p. 532)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.**↩ **RELIABLE_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.DataReaderResourceLimitsQos**↩ **Policy.max_samples_per_remote_writer** (p. 532) will trigger a rejection, not a loss, with reason **com.rti.dds.**↩ **subscription.SampleRejectedStatusKind.REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT** (p. 1661).

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.8 LOST_BY_VIRTUAL_WRITERS_LIMIT

```
final  SampleLostStatusKind LOST_BY_VIRTUAL_WRITERS_LIMIT  [static]
```

Initial value:

```
= new SampleLostStatusKind(
                "DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT", 7)
```

A resource limit on the number of virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 450) may read (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_virtual_writers** (p. 537)) was reached.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.9 LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT

```
final  SampleLostStatusKind LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT  [static]
```

Initial value:
```
= new SampleLostStatusKind(
                "LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT", 8)
```

A resource limit on the number of remote writers per sample (**com.rti.dds.infrastructure.DataReaderResource↩ LimitsQosPolicy.max_remote_writers_per_sample** (p. 538)) was reached.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.10 LOST_BY_AVAILABILITY_WAITING_TIME

```
final  SampleLostStatusKind LOST_BY_AVAILABILITY_WAITING_TIME  [static]
```

Initial value:
```
= new SampleLostStatusKind(
                "LOST_BY_AVAILABILITY_WAITING_TIME", 9)
```

com.rti.dds.infrastructure.AvailabilityQosPolicy.max_data_availability_waiting_time (p. 345) expired.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.AvailabilityQosPolicy (p. 343)

8.300.2.11 LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT

```
final  SampleLostStatusKind LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT  [static]
```

Initial value:
```
= new SampleLostStatusKind(
                "LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT", 10)
```

A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **com.rti.↩ dds.subscription.DataReader** (p. 450) may store was reached. (This field is currently not used.)

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.12 LOST_BY_OUT_OF_MEMORY

```
final SampleLostStatusKind LOST_BY_OUT_OF_MEMORY [static]
```

Initial value:

```
= new SampleLostStatusKind(
                "LOST_BY_OUT_OF_MEMORY", 11)
```

A sample was lost because there was not enough memory to store the sample.

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.300.2.13 LOST_BY_UNKNOWN_INSTANCE

```
final SampleLostStatusKind LOST_BY_UNKNOWN_INSTANCE [static]
```

Initial value:

```
=
        new SampleLostStatusKind("LOST_BY_UNKNOWN_INSTANCE", 12)
```

A received sample was lost because it doesn't contain enough information for the reader to know what instance it is associated with.

This constant is an extension to the DDS standard.

8.300.2.14 LOST_BY_DESERIALIZATION_FAILURE

```
final SampleLostStatusKind LOST_BY_DESERIALIZATION_FAILURE [static]
```

Initial value:

```
=
        new SampleLostStatusKind("LOST_BY_DESERIALIZATION_FAILURE", 13)
```

A received sample was lost because it could not be deserialized.

This constant is an extension to the DDS standard.

8.300.2.15 LOST_BY_DECODE_FAILURE

```
final SampleLostStatusKind LOST_BY_DECODE_FAILURE [static]
```

Initial value:

```
=
        new SampleLostStatusKind("LOST_BY_DECODE_FAILURE", 14)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): A received sample was lost because it could not be decoded. When using **com.rti.dds.infrastructure.ReliabilityQos↩ PolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), the sample will be rejected, not lost, with reason **com.rti.dds.↩ subscription.SampleRejectedStatusKind.REJECTED_BY_DECODE_FAILURE** (p. 1662).

This constant is an extension to the DDS standard.

8.300.2.16 LOST_BY_SAMPLES_PER_INSTANCE_LIMIT

```
final SampleLostStatusKind LOST_BY_SAMPLES_PER_INSTANCE_LIMIT [static]
```

Initial value:

```
=
        new SampleLostStatusKind("LOST_BY_SAMPLES_PER_INSTANCE_LIMIT", 15)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples per instance (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) will trigger a rejection, not a loss, with reason **com.rti.dds.subscription.SampleRejectedStatusKind.REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT** (p. 1661).

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.300.2.17 LOST_BY_SAMPLES_LIMIT

```
final SampleLostStatusKind LOST_BY_SAMPLES_LIMIT [static]
```

Initial value:

```
=
        new SampleLostStatusKind("LOST_BY_SAMPLES_LIMIT", 16)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) will trigger a rejection, not a loss, with reason **com.rti.dds.subscription.SampleRejectedStatusKind.REJECTED_BY_SAMPLES_LIMIT** (p. 1660).

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.301 SampleRejectedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of samples rejected by the **com.rti.dds.subscription.DataReader** (p. 450).*
- int **total_count_change**

  The incremental number of samples rejected since the last time the listener was called or the status was read.
- **SampleRejectedStatusKind last_reason**

  Reason for rejecting the last sample rejected.
- final **InstanceHandle_t last_instance_handle**

  Handle to the instance being updated by the last sample that was rejected.

8.301.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS

8.301.2 Member Data Documentation

8.301.2.1 total_count

```
int total_count
```

Total cumulative count of samples rejected by the **com.rti.dds.subscription.DataReader** (p. 450).

8.301.2.2 total_count_change

```
int total_count_change
```

The incremental number of samples rejected since the last time the listener was called or the status was read.

8.301.2.3 last_reason

```
SampleRejectedStatusKind last_reason
```

Reason for rejecting the last sample rejected.

See also

com.rti.dds.subscription.SampleRejectedStatusKind (p. 1659)

8.301.2.4 last_instance_handle

```
final  InstanceHandle_t last_instance_handle
```

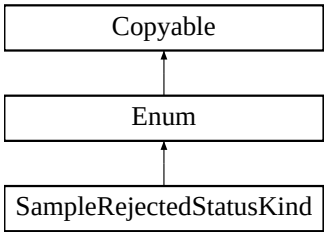
Handle to the instance being updated by the last sample that was rejected.

If the sample was rejected because of **com.rti.dds.subscription.SampleRejectedStatusKind.REJECTED_BY_↩ DECODE_FAILURE** (p. 1662) and the **com.rti.dds.publication.DataWriter** (p. 553) set **com.rti.dds.infrastructure.↩ DataWriterProtocolQosPolicy.disable_inline_keyhash** (p. 599) to com.rti.dds.infrastructure.true, then the last_↩ instance_handle may not be correct if the sample was encrypted.

8.302 SampleRejectedStatusKind Class Reference

Kinds of reasons for rejecting a sample.

Inheritance diagram for SampleRejectedStatusKind:



Static Public Attributes

- static final **SampleRejectedStatusKind NOT_REJECTED**

  The sample was not rejected.
- static final **SampleRejectedStatusKind REJECTED_BY_INSTANCES_LIMIT**

  *Connext DDS does not reject samples based on instance limits (**com.rti.dds.infrastructure.ResourceLimitsQos↩ Policy.max_instances** (p. 1592)), so this value will never be used.*
- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_LIMIT**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_SAMPLES_LIMIT** (p. 1657).*
- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples per instance (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_↩ samples_per_instance** (p. 1593)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.↩ BEST_EFFORT_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.↩ max_samples_per_instance** (p. 1593) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.↩ SampleLostStatusKind.LOST_BY_SAMPLES_PER_INSTANCE_LIMIT** (p. 1656).*
- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT**

*When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): a resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 450) may store (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT** (p. 1653).*

- static final **SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT**

  *A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **com.rti.dds.subscription.DataReader** (p. 450) may store was reached. (This field is currently not used.)*

- static final **SampleRejectedStatusKind REJECTED_BY_DECODE_FAILURE**

  *When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A received sample was rejected because it could not be decoded. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), the sample will be lost, not rejected, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_DECODE_FAILURE** (p. 1656).*

Additional Inherited Members

8.302.1 Detailed Description

Kinds of reasons for rejecting a sample.

8.302.2 Member Data Documentation

8.302.2.1 NOT_REJECTED

```
final SampleRejectedStatusKind NOT_REJECTED [static]
```

Initial value:

```
= new com.rti.dds.subscription.SampleRejectedStatusKind(
            "NOT_REJECTED", 0)
```

The sample was not rejected.

MONITOR-273 We assign an integer to the names in order to be compatible with the DDSMonitoring types. If new values are added, update DataReaderTest.testGetAndSetSampleRejectedStatus

8.302.2.2 REJECTED_BY_INSTANCES_LIMIT

```
final SampleRejectedStatusKind REJECTED_BY_INSTANCES_LIMIT [static]
```

Initial value:

```
= new com.rti.dds.subscription.SampleRejectedStatusKind(
            "REJECTED_BY_INSTANCES_LIMIT", 1)
```

Connext DDS does not reject samples based on instance limits (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592)), so this value will never be used.

See also

com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_INSTANCES_LIMIT<P> (p. 1652)

8.302.2.3 REJECTED_BY_SAMPLES_LIMIT

```
final SampleRejectedStatusKind REJECTED_BY_SAMPLES_LIMIT [static]
```

Initial value:

```
= new SampleRejectedStatusKind(
            "REJECTED_BY_SAMPLES_LIMIT", 2)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1592) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_SAMPLES_LIMIT** (p. 1657).

See also

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1590)

8.302.2.4 REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT

```
final SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT [static]
```

Initial value:

```
= new SampleRejectedStatusKind(
                "REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT", 3)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A resource limit on the number of samples per instance (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1593) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_SAMPLES_PER_INSTANCE_LIMIT** (p. 1656).

See also

ResourceLimitsQosPolicy

8.302.2.5 REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT

```
final SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT [static]
```

Initial value:

```
= new SampleRejectedStatusKind(
                "REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT", 6)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): a resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 450) may store (**com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 532)) was reached. When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.BEST_EFFORT_↩ RELIABILITY_QOS** (p. 1532), reaching **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max↩ _samples_per_remote_writer** (p. 532) will trigger a loss, not a rejection, with reason **com.rti.dds.subscription.↩ SampleLostStatusKind.LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT** (p. 1653).

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.302.2.6 REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT

```
final SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT [static]
```

Initial value:

```
= new SampleRejectedStatusKind(
                "REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT", 9)
```

A resource limit on the number of samples published by a remote writer on behalf of a virtual writer that a **com.rti.↩ dds.subscription.DataReader** (p. 450) may store was reached. (This field is currently not used.)

This constant is an extension to the DDS standard.

See also

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 529)

8.302.2.7 REJECTED_BY_DECODE_FAILURE

```
final  SampleRejectedStatusKind REJECTED_BY_DECODE_FAILURE  [static]
```

Initial value:

```
=
            new SampleRejectedStatusKind("REJECTED_BY_DECODE_FAILURE", 10)
```

When using **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532): A received sample was rejected because it could not be decoded. When using **com.rti.dds.infrastructure.Reliability↩ QosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1532), the sample will be lost, not rejected, with reason **com.rti.dds.subscription.SampleLostStatusKind.LOST_BY_DECODE_FAILURE** (p. 1656).

If a sample was rejected for this reason and the **com.rti.dds.publication.DataWriter** (p. 553) set **com.rti.dds.↩ infrastructure.DataWriterProtocolQosPolicy.disable_inline_keyhash** (p. 599) to com.rti.dds.infrastructure.true, then **com.rti.dds.subscription.SampleRejectedStatus.last_instance_handle** (p. 1658) may not be correct if the sample was encrypted.

This constant is an extension to the DDS standard.

8.303 SampleStateKind Class Reference

Indicates whether or not a sample has ever been read.

Static Public Attributes

- static final int **READ_SAMPLE_STATE** = 0x0001 << 0

  Sample has been read.
- static final int **NOT_READ_SAMPLE_STATE** = 0x0001 << 1

  Sample has not been read.
- static final int **ANY_SAMPLE_STATE** = 0xffff

  Any sample state com.rti.dds.subscription.SampleStateKind.SampleStateKind.READ_SAMPLE_STATE | com.rti.dds.↩ subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE.

8.303.1 Detailed Description

Indicates whether or not a sample has ever been read.

For each sample received, the middleware internally maintains a sample_state relative to each **com.rti.dds.↩ subscription.DataReader** (p. 450). The sample state can be either:

- com.rti.dds.subscription.SampleStateKind.SampleStateKind.READ_SAMPLE_STATE indicates that the **com.↩ rti.dds.subscription.DataReader** (p. 450) has already accessed that sample by means of a read or take operation.
- com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_STATE indicates that the **com.rti.dds.subscription.DataReader** (p. 450) has not accessed that sample before.

The sample state will, in general, be different for each sample in the collection returned by read or take.

8.303.2 Member Data Documentation

8.303.2.1 READ_SAMPLE_STATE

```
final int READ_SAMPLE_STATE = 0x0001 << 0  [static]
```

Sample has been read.

8.303.2.2 NOT_READ_SAMPLE_STATE

```
final int NOT_READ_SAMPLE_STATE = 0x0001 << 1  [static]
```

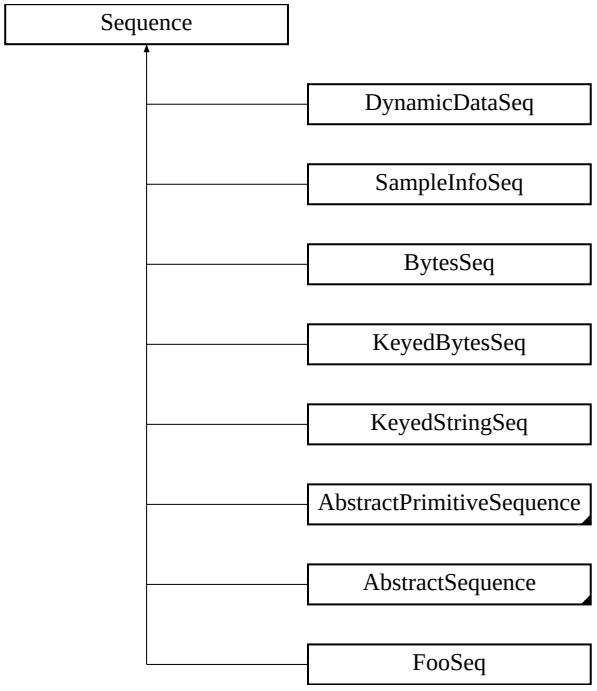
Sample has not been read.

Referenced by **Requester< TReq, TRep >.waitForReplies()**.

8.304 Sequence Interface Reference

*<<**interface**>> (p. 156) <<**generic**>> (p. 156)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **com.rti.ndds.example.Foo** (p. 1066).

Inheritance diagram for Sequence:



Public Member Functions

- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- void **setMaximum** (int new_max)

  Resize this sequence to a new desired maximum.
- Class **getElementType** ()

8.304.1 Detailed Description

<<***interface***>> *(*p. *156)* <<***generic***>> *(*p. *156)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **`com.rti.ndds.example.Foo`** (p. 1066).

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo>`.

For any user-data type `Foo` that an application defines for the purpose of data-distribution with RTI Connext, a `FooSeq` is generated. We refer to an IDL `sequence<Foo> as FooSeq`.

A sequence is a type-safe List that makes a distinction between its allocated size and its logical size (much like the ArrayList class). The Collection.size() method returns the logical size.

A new sequence is created for elements of a particular Class, which does not change throughout the lifetime of a sequence instance.

To add an element to a sequence, use the `add()` method inherited from the standard interface `java.util.List`; this will implicitly increase the sequence's size. Or, to pre-allocate space for several elements at once, use com.rti.↩ dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

An attempt to add an element to a sequence that is not of the correct element type will result in a ClassCastException. (Note that null is considered to belong to any type.)

See also

com.rti.ndds.example.FooDataWriter (p. 1097), **com.rti.ndds.example.FooDataReader** (p. 1067), **com.rti.↩ ndds.example.FooTypeSupport** (p. 1118), the `Code Generator User's Manual`

8.304.2 Member Function Documentation

8.304.2.1 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with `add()`, or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implemented in **BooleanSeq** (p. 364), **ByteSeq** (p. 400), **CharSeq** (p. 421), **ConditionSeq** (p. 432), **DoubleSeq** (p. 828), **FloatSeq** (p. 1053), **IntSeq** (p. 1167), **LongSeq** (p. 1293), **ShortSeq** (p. 1691), **PublisherSeq** (p. 1495), **DataReaderSeq** (p. 544), **SubscriberSeq** (p. 1762), and **LoanableSequence** (p. 1250).

Referenced by **AbstractPrimitiveSequence.loan()**, and **AbstractPrimitiveSequence.setSize()**.

8.304.2.2 setMaximum()

```
void setMaximum (
             int new_max )
```

Resize this sequence to a new desired maximum.

This operation does nothing if the new desired maximum matches the current maximum.

Note: If you add an element with `add()`, the sequence's size is increased implicitly.

Postcondition

length == MINIMUM(original length, new_max)

Parameters

new_max	Must be >= 0.

Implemented in **AbstractSequence** (p. 328), and **LoanableSequence** (p. 1251).

8.304.2.3 getElementType()

```
Class getElementType ( )
```

Returns

a common supertype for all elements in this sequence.

Implemented in **AbstractPrimitiveSequence** (p. 323), and **AbstractSequence** (p. 329).

8.305 SequenceNumber_t Class Reference

Type for *sequence* number representation.

Inherits Struct.

Public Member Functions

- **SequenceNumber_t** ()
  Constructor.
- **SequenceNumber_t** (**SequenceNumber_t** sn)
  Copy constructor.
- **SequenceNumber_t** (int **high**, long **low**)
  Constructor.
- int **compare** (**SequenceNumber_t** sn)
  Compares two sequence numbers.
- void **plusplus** ()
  Increases the value of this by one
- void **minusminus** ()
  Decreases the value of this by one
- **SequenceNumber_t** **add** (**SequenceNumber_t** val)
  Returns a sequence number whose value is (this + val)
- **SequenceNumber_t** **subtract** (**SequenceNumber_t** val)
  Returns a sequence number whose value is (this - val)

Public Attributes

- int **high**
  The most significant part of the sequence number.
- long **low**
  The least significant part of the sequence number.

Static Public Attributes

- static final **SequenceNumber_t SEQUENCE_NUMBER_UNKNOWN**

  Unknown sequence number.
- static final **SequenceNumber_t AUTO_SEQUENCE_NUMBER**

  The sequence number is internally determined by RTI Connext.
- static final **SequenceNumber_t SEQUENCE_NUMBER_ZERO**

  Zero value for the sequence number.
- static final **SequenceNumber_t SEQUENCE_NUMBER_MAX**

  Highest, most positive value for the sequence number.

8.305.1 Detailed Description

Type for *sequence* number representation.

Represents a 64-bit sequence number.

8.305.2 Constructor & Destructor Documentation

8.305.2.1 SequenceNumber_t() [1/3]

```
SequenceNumber_t ( )
```

Constructor.

References **SequenceNumber_t.SEQUENCE_NUMBER_UNKNOWN**.

Referenced by **SequenceNumber_t.add()**, and **SequenceNumber_t.subtract()**.

8.305.2.2 SequenceNumber_t() [2/3]

```
SequenceNumber_t (
            SequenceNumber_t sn )
```

Copy constructor.

Parameters

sn	The sequence number instance to copy. It must not be null.

References **SequenceNumber_t.high**, and **SequenceNumber_t.low**.

8.305.2.3 SequenceNumber_t() [3/3]

```
SequenceNumber_t (
            int high,
            long low )
```

Constructor.

Parameters

high	must be in the interval [0,0xffffffff]
low	must be in the interval [0,0x00000000ffffffff]

Exceptions

RETCODE_BAD_PARAMETER *(p. 1594)*	

References **SequenceNumber_t.high**, and **SequenceNumber_t.low**.

8.305.3 Member Function Documentation

8.305.3.1 compare()

```
int compare (
             SequenceNumber_t sn )
```

Compares two sequence numbers.

Parameters

sn	<<***in***>> *(p. 156)* Sequence number to compare. Cannot be null.

Returns

If the two sequence numbers are equal, the function returns 0. If sn1 is greater than sn2 the function returns a positive number; otherwise, it returns a negative number.

References **SequenceNumber_t.high**, and **SequenceNumber_t.low**.

8.305.3.2 plusplus()

```
void plusplus ( )
```

Increases the value of this by one

References **SequenceNumber_t.high**, and **SequenceNumber_t.low**.

8.305.3.3 minusminus()

```
void minusminus ( )
```

Decreases the value of this by one

References **SequenceNumber_t.high**, and **SequenceNumber_t.low**.

8.305.3.4 add()

```
SequenceNumber_t add (
            SequenceNumber_t val )
```

Returns a sequence number whose value is (this + val)

Returns

(this+val)

References **SequenceNumber_t.high**, **SequenceNumber_t.low**, and **SequenceNumber_t.SequenceNumber_t()**.

8.305.3.5 subtract()

```
SequenceNumber_t subtract (
            SequenceNumber_t val )
```

Returns a sequence number whose value is (this - val)

Returns

(this-val)

References **SequenceNumber_t.high**, **SequenceNumber_t.low**, and **SequenceNumber_t.SequenceNumber_t()**.

8.305.4 Member Data Documentation

8.305.4.1 SEQUENCE_NUMBER_UNKNOWN

```
final SequenceNumber_t SEQUENCE_NUMBER_UNKNOWN [static]
```

Unknown sequence number.

Referenced by **SequenceNumber_t.SequenceNumber_t()**.

8.305.4.2 AUTO_SEQUENCE_NUMBER

```
final SequenceNumber_t AUTO_SEQUENCE_NUMBER [static]
```

The sequence number is internally determined by RTI Connext.

8.305.4.3 SEQUENCE_NUMBER_ZERO

```
final SequenceNumber_t SEQUENCE_NUMBER_ZERO [static]
```

Zero value for the sequence number.

8.305.4.4 SEQUENCE_NUMBER_MAX

```
final SequenceNumber_t SEQUENCE_NUMBER_MAX [static]
```

Highest, most positive value for the sequence number.

8.305.4.5 high

```
int high
```

The most significant part of the sequence number.

Referenced by **SequenceNumber_t.add()**, **SequenceNumber_t.compare()**, **SequenceNumber_t.minusminus()**, **SequenceNumber_t.plusplus()**, **SequenceNumber_t.SequenceNumber_t()**, and **SequenceNumber_t.subtract()**.

8.305.4.6 low

```
long low
```

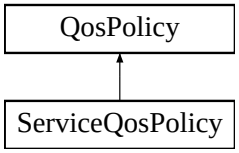
The least significant part of the sequence number.

Referenced by **SequenceNumber_t.add()**, **SequenceNumber_t.compare()**, **SequenceNumber_t.minusminus()**, **SequenceNumber_t.plusplus()**, **SequenceNumber_t.SequenceNumber_t()**, and **SequenceNumber_t.subtract()**.

8.306 ServiceQosPolicy Class Reference

Service associated with a DDS entity.

Inheritance diagram for ServiceQosPolicy:



Public Attributes

- **ServiceQosPolicyKind kind**
  The kind of service.

8.306.1 Detailed Description

Service associated with a DDS entity.

This QoS policy is intended to be used by RTI infrastructure services.

User applications should not modify its value.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.306.2 Member Data Documentation

8.306.2.1 kind

```
ServiceQosPolicyKind kind
```

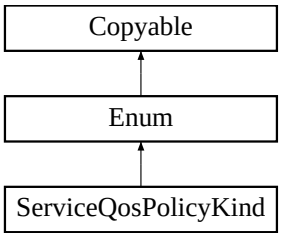
The kind of service.

[default] com.rti.dds.infrastructure.ServiceQosPolicyKind.NO_SERVICE_QOS (p. 1674)

8.307 ServiceQosPolicyKind Class Reference

Kinds of service.

Inheritance diagram for ServiceQosPolicyKind:



Static Public Attributes

- static final **ServiceQosPolicyKind NO_SERVICE_QOS**
  There is no service associated with the entity.
- static final **ServiceQosPolicyKind PERSISTENCE_SERVICE_QOS**
  The entity is an entity created by RTI Persistence Service.
- static final **ServiceQosPolicyKind QUEUING_SERVICE_QOS**
  The entity is an entity created by RTI Queuing Service.
- static final **ServiceQosPolicyKind ROUTING_SERVICE_QOS**
  The entity is an entity created by RTI Routing Service.
- static final **ServiceQosPolicyKind RECORDING_SERVICE_QOS**
  The entity is an entity created by RTI Recording Service.
- static final **ServiceQosPolicyKind REPLAY_SERVICE_QOS**
  The entity is an entity created by RTI Replay Service.
- static final **ServiceQosPolicyKind DATABASE_INTEGRATION_SERVICE_QOS**
  The entity is an entity created by RTI Database Integration Service.
- static final **ServiceQosPolicyKind WEB_INTEGRATION_SERVICE_QOS**
  The entity is an entity created by RTI Web Integration Service.
- static final **ServiceQosPolicyKind OBSERVABILITY_COLLECTOR_SERVICE_QOS**
  The entity is an entity created by RTI Observability Collector Service.

Additional Inherited Members

8.307.1 Detailed Description

Kinds of service.

QoS:

com.rti.dds.infrastructure.ServiceQosPolicy (p. 1672)

8.307.2 Member Data Documentation

8.307.2.1 NO_SERVICE_QOS

```
final ServiceQosPolicyKind NO_SERVICE_QOS [static]
```

There is no service associated with the entity.

8.307.2.2 PERSISTENCE_SERVICE_QOS

```
final ServiceQosPolicyKind PERSISTENCE_SERVICE_QOS [static]
```

The entity is an entity created by RTI Persistence Service.

8.307.2.3 QUEUING_SERVICE_QOS

```
final ServiceQosPolicyKind QUEUING_SERVICE_QOS [static]
```

The entity is an entity created by RTI Queuing Service.

8.307.2.4 ROUTING_SERVICE_QOS

```
final ServiceQosPolicyKind ROUTING_SERVICE_QOS [static]
```

The entity is an entity created by RTI Routing Service.

8.307.2.5 RECORDING_SERVICE_QOS

```
final ServiceQosPolicyKind RECORDING_SERVICE_QOS [static]
```

The entity is an entity created by RTI Recording Service.

8.307.2.6 REPLAY_SERVICE_QOS

```
final ServiceQosPolicyKind REPLAY_SERVICE_QOS [static]
```

The entity is an entity created by RTI Replay Service.

8.307.2.7 DATABASE_INTEGRATION_SERVICE_QOS

```
final ServiceQosPolicyKind DATABASE_INTEGRATION_SERVICE_QOS [static]
```

The entity is an entity created by RTI Database Integration Service.

8.307.2.8 WEB_INTEGRATION_SERVICE_QOS

```
final ServiceQosPolicyKind WEB_INTEGRATION_SERVICE_QOS [static]
```

The entity is an entity created by RTI Web Integration Service.

8.307.2.9 OBSERVABILITY_COLLECTOR_SERVICE_QOS

```
final ServiceQosPolicyKind OBSERVABILITY_COLLECTOR_SERVICE_QOS [static]
```

The entity is an entity created by RTI Observability Collector Service.

8.308 ServiceRequest Class Reference

A request coming from one of the built-in services.

Inherits AbstractBuiltinTopicData.

Public Attributes

- int **service_id**

  The id of the service that the request was sent on.
- final **GUID_t instance_id**

  *Each **ServiceRequest** (p. 1675) is keyed on the instance_id.*
- final **ByteSeq request_body**

  Service-specific information.

Static Public Attributes

- static final int **UNKNOWN_SERVICE_ID**

  An invalid Service Id.
- static final int **TOPIC_QUERY_SERVICE_ID**

  *Service Id for the **com.rti.dds.subscription.TopicQuery** (p. 1830) Service.*
- static final int **LOCATOR_REACHABILITY_SERVICE_ID**

  Service Id for the Locator Reachability Service.
- static final int **INSTANCE_STATE_SERVICE_ID**

  Service Id for the Instance State Request service.
- static final int **MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Command Service Request.
- static final int **MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID**

  Service Id for RTI Monitoring Library 2.0 Reply Service Requests.

8.308.1 Detailed Description

A request coming from one of the built-in services.

Data associated with the built-in topic **com.rti.dds.topic.builtin.ServiceRequestTypeSupport.SERVICE_↩ REQUEST_TOPIC_NAME** (p. 168). It contains service-specific information.

See also

com.rti.dds.topic.builtin.ServiceRequestTypeSupport.SERVICE_REQUEST_TOPIC_NAME (p. 168)

builtin.ParticipantBuiltinTopicDataDataReader

8.308.2 Member Data Documentation

8.308.2.1 service_id

```
int service_id
```

The id of the service that the request was sent on.

There can be multiple services that use the built-in **ServiceRequest** (p. 1675) topic. The service_id identifies which service a specific request was sent from.

See also

com.rti.dds.topic.builtin.ServiceRequest.UNKNOWN_SERVICE_ID (p. 167)

com.rti.dds.topic.builtin.ServiceRequest.TOPIC_QUERY_SERVICE_ID (p. 167)

8.308.2.2 instance_id

```
final GUID_t instance_id
```

Each **ServiceRequest** (p. 1675) is keyed on the instance_id.

The instance_id provides a way for users to differentiate between different requests coming from the same service.

8.308.2.3 request_body

```
final ByteSeq request_body
```

Service-specific information.

Each service uses the request_body field to send information specific to that service in the form of an opaque sequence of bytes. Each service provides a helper function that will deserialize the information from the request body.

See also

com.rti.dds.subscription.TopicQueryHelper.topic_query_data_from_service_request (p. 1835)

Referenced by **TopicQueryHelper.topic_query_data_from_service_request()**.

8.309 ServiceRequestAcceptedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS (p. 1709)

Inherits Status.

Public Attributes

- int **total_count**

  *The total cumulative number of ServiceRequests that have been accepted by a **com.rti.dds.publication.DataWriter** (p. 553).*
- int **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.
- int **current_count**

  *The current number of ServiceRequests that have been accepted by this **com.rti.dds.publication.DataWriter** (p. 553).*
- int **current_count_change**

  The change in current_count since the last time the listener was called or the status was read.
- final **InstanceHandle_t last_request_handle**

  *A handle to the last **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) that caused the **com.rti.dds.publication.DataWriter** (p. 553)'s status to change.*
- int **service_id**

  ID of the service to which the accepted Request belongs.

8.309.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.SERVICE_REQUEST_ACCEPTED_STATUS (p. 1709)

Currently, the only service that causes the **ServiceRequestAcceptedStatus** (p. 1677) to be triggered is the **com.rti.dds.subscription.TopicQuery** (p. 1830) service. A **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) is accepted when a **com.rti.dds.publication.DataWriter** (p. 553) matches with a **com.rti.dds.subscription.DataReader** (p. 450) that has created a **com.rti.dds.subscription.TopicQuery** (p. 1830).

This status is also changed (and the listener, if any, called) when a ServiceRequest has been cancelled, or deleted. This will happen when a **com.rti.dds.subscription.DataReader** (p. 450) deletes a TopicQuery using **com.rti.dds.subscription.DataReader.delete_topic_query** (p. 473).

8.309.2 Member Data Documentation

8.309.2.1 total_count

```
int total_count
```

The total cumulative number of ServiceRequests that have been accepted by a **com.rti.dds.publication.DataWriter** (p. 553).

This number increases whenever a new request is accepted. It does not change when a request is cancelled.

8.309.2.2 total_count_change

```
int total_count_change
```

The incremental changes in total_count since the last time the listener was called or the status was read.

8.309.2.3 current_count

```
int current_count
```

The current number of ServiceRequests that have been accepted by this **com.rti.dds.publication.DataWriter** (p. 553).

This number increases when a new request is accepted and decreases when an existing request is cancelled.

8.309.2.4 current_count_change

```
int current_count_change
```

The change in current_count since the last time the listener was called or the status was read.

8.309.2.5 last_request_handle

```
final  InstanceHandle_t last_request_handle
```

A handle to the last **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) that caused the **com.rti.dds.publication.↩ DataWriter** (p. 553)'s status to change.

8.309.2.6 service_id

```
int service_id
```

ID of the service to which the accepted Request belongs.

See also

com.rti.dds.topic.builtin.ServiceRequest.TOPIC_QUERY_SERVICE_ID (p. 167)

8.310 ServiceRequestDataReader Class Reference

Instantiates `DataReader` < **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) > .

Inherits DataReaderImpl.

8.310.1 Detailed Description

Instantiates `DataReader` < **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) > .

com.rti.dds.subscription.DataReader (p. 450) of topic **com.rti.dds.topic.builtin.ServiceRequestTypeSupport.↩ SERVICE_REQUEST_TOPIC_NAME** (p. 168) used for accessing **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) samples.

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooDataReader** (p. 1067)

See also

com.rti.dds.topic.builtin.ServiceRequest (p. 1675)

com.rti.dds.topic.builtin.ServiceRequestTypeSupport.SERVICE_REQUEST_TOPIC_NAME (p. 168)

A reader for the **ServiceRequest** (p. 1675) user type.

8.311 ServiceRequestSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.topic.↩ builtin.ServiceRequest** (p. 1675) > .

Inherits AbstractBuiltinTopicDataSeq.

8.311.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **com.rti.dds.topic.↩ builtin.ServiceRequest** (p. 1675) > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

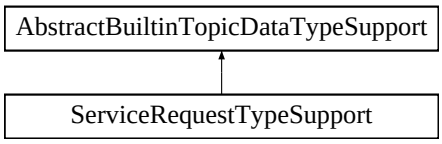
See also

com.rti.dds.topic.builtin.ServiceRequest (p. 1675)

8.312 ServiceRequestTypeSupport Class Reference

Instantiates **TypeSupport** (p. 1941) < **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) > .

Inheritance diagram for ServiceRequestTypeSupport:



Static Public Attributes

- static final String **SERVICE_REQUEST_TOPIC_NAME** = DDS_SERVICE_REQUEST_TOPIC_NAME()
  Service Request topic name.

Additional Inherited Members

8.312.1 Detailed Description

Instantiates **TypeSupport** (p. 1941) < **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) > .

Instantiates:

<<***generic***>> *(p. 156)* **com.rti.ndds.example.FooTypeSupport** (p. 1118)

See also

com.rti.dds.topic.builtin.ServiceRequest (p. 1675)

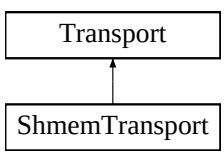
Author

erin

8.313 ShmemTransport Interface Reference

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

Inheritance diagram for ShmemTransport:



Classes

- class **Property_t**

  *Subclass of **com.rti.ndds.transport.Transport.Property_t** (p. 1401) allowing specification of parameters that are specific to the shared-memory transport.*

Static Public Attributes

- static final int **CLASSID** = 0x01000000
- static final int **MESSAGE_SIZE_MAX_DEFAULT**

  ***[default]** Value for **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404).*
- static final int **RECEIVED_MESSAGE_COUNT_MAX_DEFAULT**

  ***[default]** Value for **com.rti.ndds.transport.ShmemTransport.Property_t.received_message_count_max** (p. 1399).*
- static final int **RECEIVE_BUFFER_SIZE_DEFAULT**

  ***[default]** Value for **com.rti.ndds.transport.ShmemTransport.Property_t.receive_buffer_size** (p. 1400).*

8.313.1 Detailed Description

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

This transport plugin uses System Shared Memory to send messages between processes on the same node. This transport is installed as a built-in transport plugin with the alias **com.rti.dds.infrastructure.TransportBuiltinKind.↩ SHMEM_ALIAS** (p. 270).

8.313.2 Compatibility of Sender and Receiver Transports

Opening a receiver "port" on shared memory corresponds to creating a shared memory segment using a name based on the port number. The transport plugin's properties are embedded in the shared memory segment.

When a sender tries to send to the shared memory port, it verifies that properties of the receiver's shared memory transport are compatible with those specified in its transport plugin. If not, the sender will fail to attach to the port and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_attachShmem:failed to initialize incompatible properties

NDDS_Transport_Shmem_attachShmem:countMax 0 > -19417345 or max size -19416188 > 2147482624
```

In this scenario, the properties of the sender or receiver transport plugin instances should be adjusted, so that they are compatible.

8.313.3 Crashing and Restarting Programs

If a process using shared memory crashes (say because the user typed in ^C), resources associated with its shared memory ports may not be properly cleaned up. Later, if another RTI Connext process needs to open the same ports (say, the crashed program is restarted), it will attempt to reuse the shared memory segment left behind by the crashed process.

The reuse is allowed iff the properties of transport plugin are compatible with those embedded in the shared memory segment (i.e., of the original creator). Otherwise, the process will fail to open the ports, and will output messages such as below (with numbers appropriate to the properties of the transport plugins involved).

```
NDDS_Transport_Shmem_create_recvresource_rrEA:failed to initialize shared

    memory resource Cannot recycle existing shmem: size not compatible for key 0x1234
```

In this scenario, the shared memory segments must be cleaned up using appropriate platform specific commands. For details, please refer to the `Platform Notes`.

8.313.4 Shared Resource Keys

The transport uses the **shared memory segment keys**, given by the formula below.
`0x400000 + port`

The transport also uses signaling **shared semaphore** keys given by the formula below.
`0x800000 + port`

The transport also uses mutex **shared semaphore keys** given by the formula below.
`0xb00000 + port`

wher the `port` is a function of the `domain_id` and the `participant_id`, as described in **com.rti.dds.↩ infrastructure.WireProtocolQosPolicy.participant_id** (p. 1990)

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id (p. 1990)

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.313.5 Creating and Registering Shared Memory Transport Plugin

RTI Connext can implicitly create this plugin and register with the **com.rti.dds.domain.DomainParticipant** (p. 670) if this transport is specified in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843).

To specify the properties of the builtin shared memory transport that is implicitly registered, you can either:

- call **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property** (p. 1863) or
- specify the pre-defined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipant** (p. 670). (see **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1684)).

Builtin transport plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) always overwrite the ones specified through **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property()** (p. 1863). The default value is assumed on any unspecified property. Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

8.313.6 Shared Memory Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **com.rti.dds.domain.DomainParticipantQos.property** (p. 801) to to configure the builtin shared memory transport plugin.

Table 8.1013 ***Property Strings for Shared Memory Transport (*** p. ***1841)***

Name	Descriptions
dds.transport.shmem.builtin.parent.address_bit_count	See **com.rti.ndds.transport.Transport.Property_t.address_bit_count** (p. 1403)
dds.transport.shmem.builtin.parent.properties_bitmap	See **com.rti.ndds.transport.Transport.Property_t.properties_bitmap** (p. 1403)
dds.transport.shmem.builtin.parent.gather_send_buffer_count_max	See **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404)
dds.transport.shmem.builtin.parent.message_size_max	See **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404)
dds.transport.shmem.builtin.parent.allow_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404) and com.rti.ndds.transport.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
dds.transport.shmem.builtin.parent.deny_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405) and com.rti.ndds.transport.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
dds.transport.shmem.builtin.parent.allow_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list** (p. 1405) and com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list** (p. 1406) and com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.max_interface_count	See **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406)
dds.transport.shmem.builtin.parent.thread_name_prefix	See com.rti.ndds.transport.Transport.Property_t.thread_name_prefix

Name	Descriptions
dds.transport.shmem.builtin.received_message_↩ count_max	See **com.rti.ndds.transport.ShmemTransport.↩ Property_t.received_message_count_max** (p. 1399)
dds.transport.shmem.builtin.receive_buffer_size	See **com.rti.ndds.transport.ShmemTransport.↩ Property_t.receive_buffer_size** (p. 1400)
dds.transport.shmem.builtin.enable_udp_debugging	See com.rti.ndds.transport.ShmemTransport.Property↩ _t.enable_udp_debugging
dds.transport.shmem.builtin.udp_debugging_address	See com.rti.ndds.transport.ShmemTransport.Property↩ _t.udp_debugging_address
dds.transport.shmem.builtin.udp_debugging_port	See com.rti.ndds.transport.ShmemTransport.Property↩ _t.udp_debugging_port

8.313.7 Member Data Documentation

8.313.7.1 CLASSID

```
final int CLASSID = 0x01000000  [static]
```

The Shmem **Transport** (p. 1841) class id

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.313.7.2 MESSAGE_SIZE_MAX_DEFAULT

```
final int MESSAGE_SIZE_MAX_DEFAULT  [static]
```

[default] Value for **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404).

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.313.7.3 RECEIVED_MESSAGE_COUNT_MAX_DEFAULT

```
final int RECEIVED_MESSAGE_COUNT_MAX_DEFAULT  [static]
```

[default] Value for **com.rti.ndds.transport.ShmemTransport.Property_t.received_message_count_max** (p. 1399).

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.313.7.4 RECEIVE_BUFFER_SIZE_DEFAULT

```
final int RECEIVE_BUFFER_SIZE_DEFAULT  [static]
```

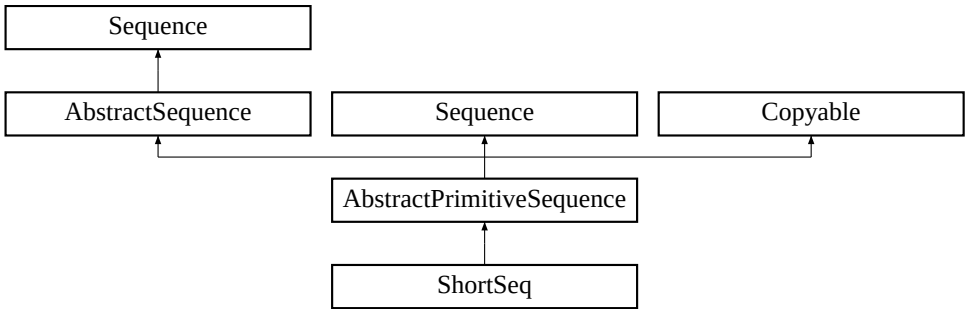
[default] Value for **com.rti.ndds.transport.ShmemTransport.Property_t.receive_buffer_size** (p. 1400).

Referenced by **ShmemTransport.Property_t.Property_t()**.

8.314 ShortSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < short >

Inheritance diagram for ShortSeq:



Public Member Functions

- **ShortSeq** ()

  Constructs an empty sequence of short integers with an initial maximum of zero.
- **ShortSeq** (int initialMaximum)

  Constructs an empty sequence of short integers with the given initial maximum.
- **ShortSeq** (short[ ] shorts)

  Constructs a new sequence containing the given shorts.
- boolean **addAllShort** (short[ ] elements, int offset, int length)

  Append `length` elements from the given array to this sequence, starting at index `offset` in that array.
- boolean **addAllShort** (short[ ] elements)
- void **addShort** (short element)

  Append the element to the end of the sequence.
- void **addShort** (int index, short element)

  Shift all elements in the sequence starting from the given index and add the element to the given index.
- short **getShort** (int index)

  Returns the short at the given index.
- short **setShort** (int index, short element)

  Set the new short at the given index and return the old short.
- void **setShort** (int dstIndex, short[ ] elements, int srcIndex, int length)

  Copy a portion of the given array into this sequence.

- short[ ] **toArrayShort** (short[ ] array)

  Return an array containing copy of the contents of this sequence.
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.
- Object **get** (int index)

  *A wrapper for **getShort(int)** (p. 1689) that returns a java.lang.Short.*
- Object **set** (int index, Object element)

  *A wrapper for **setShort()** (p. 1689).*
- void **add** (int index, Object element)

  A wrapper for addShort(int, int).

8.314.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < short >

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

short

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.314.2 Constructor & Destructor Documentation

8.314.2.1 ShortSeq() [1/3]

```
ShortSeq ( )
```

Constructs an empty sequence of short integers with an initial maximum of zero.

8.314.2.2 ShortSeq() [2/3]

```
ShortSeq (
            int initialMaximum )
```

Constructs an empty sequence of short integers with the given initial maximum.

8.314.2.3 ShortSeq() [3/3]

```
ShortSeq (
            short[] shorts )
```

Constructs a new sequence containing the given shorts.

Parameters

shorts	the initial contents of this sequence

Exceptions

NullPointerException	if the input array is null

References **ShortSeq.addAllShort()**.

8.314.3 Member Function Documentation

8.314.3.1 addAllShort() [1/2]

```
boolean addAllShort (
            short[] elements,
            int offset,
            int length )
```

Append `length` elements from the given array to this sequence, starting at index `offset` in that array.

Exceptions

NullPointerException	if the given array is null.

Referenced by **ShortSeq.addAllShort()**, and **ShortSeq.ShortSeq()**.

8.314.3.2 addAllShort() [2/2]

```
boolean addAllShort (
            short[] elements )
```

Exceptions

NullPointerException	if the given array is null

References **ShortSeq.addAllShort()**.

8.314.3.3 addShort() [1/2]

```
void addShort (
            short element )
```

Append the element to the end of the sequence.

Referenced by **ShortSeq.add()**.

8.314.3.4 addShort() [2/2]

```
void addShort (
            int index,
            short element )
```

Shift all elements in the sequence starting from the given index and add the element to the given index.

8.314.3.5 getShort()

```
short getShort (
            int index )
```

Returns the short at the given index.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **ShortSeq.get()**, **DynamicData.get_ushort_seq()**, and **DynamicData.set_uint8_seq()**.

8.314.3.6 setShort() [1/2]

```
short setShort (
            int index,
            short element )
```

Set the new short at the given index and return the old short.

Exceptions

IndexOutOfBoundsException	if the index is out of bounds.

Referenced by **ShortSeq.set()**.

8.314.3.7 setShort() [2/2]

```
void setShort (
            int dstIndex,
            short[] elements,
            int srcIndex,
            int length )
```

Copy a portion of the given array into this sequence.

Parameters

dstIndex	the index at which to start copying into this sequence.
elements	an array of primitive elements.
srcIndex	the index at which to start copying from the given array.
length	the number of elements to copy.

Exceptions

IndexOutOfBoundsException	if copying would cause access of data outside array bounds.

8.314.3.8 toArrayShort()

```
short[] toArrayShort (
            short[] array )
```

Return an array containing copy of the contents of this sequence.

Parameters

array	The array into which this sequence should be copied. It may be null. If it is, or if array length is too small, the array will be ignored, and a new array of the necessary length will be created and copied into instead.

Returns

A non-null array containing a copy of the contents of this sequence.

8.314.3.9 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 1692), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

Referenced by **DynamicData.get_short_seq()**, **DynamicData.get_uint8_seq()**, **DynamicData.get_ushort_seq()**, **DynamicData.set_short_seq()**, and **DynamicData.set_uint8_seq()**.

8.314.3.10 get()

```
Object get (
             int index )
```

A wrapper for **getShort(int)** (p. 1689) that returns a java.lang.Short.

See also

java.util.List::get(int)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **ShortSeq.getShort()**.

8.314.3.11 set()

```
Object set (
             int index,
             Object element )
```

A wrapper for **setShort()** (p. 1689).

Exceptions

ClassCastException	if the element is not of type Short.

See also

java.util.List::set(int, java.lang.Object)

Reimplemented from **AbstractPrimitiveSequence** (p. 322).

References **ShortSeq.setShort()**.

Referenced by **DynamicData.get_uint8_seq()**, and **DynamicData.set_ushort_seq()**.

8.314.3.12 add()

```
void add (
            int index,
            Object element )
```

A wrapper for addShort(int, int).

Exceptions

ClassCastException	if the element is not of type Short.

See also

java.util.List::add(int, java.lang.Object)

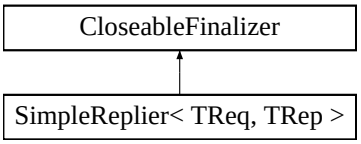
Reimplemented from **AbstractPrimitiveSequence** (p. 323).

References **ShortSeq.addShort()**.

8.315 SimpleReplier< TReq, TRep > Class Template Reference

A callback-based replier.

Inheritance diagram for SimpleReplier< TReq, TRep >:



Public Member Functions

- **SimpleReplier** (**DomainParticipant** participant, String serviceName, **SimpleReplierListener**< TReq, TRep > listener, **TypeSupport** requestTypeSupport, **TypeSupport** replyTypeSupport)

  *Creates a new **SimpleReplier** (p. 1692).*
- **SimpleReplier** (**SimpleReplierParams**< TReq, TRep > params)

  *Creates a new **SimpleReplier** (p. 1692).*
- void **close** ()

  *Releases the resources created by this **SimpleReplier** (p. 1692).*

8.315.1 Detailed Description

A callback-based replier.

A **SimpleReplier** (p. 1692) is based on a com.rti.connext.requestreply.SimpleReplierListener<TReq,TRep> that users provide . Requests are passed to the callback, which returns a reply. The reply is directed only to the **Requester** (p. 1563) that sent the request.

SimpleRepliers are useful for simple use cases where a single reply for a request can be generated quickly, for example, looking up a table.

When more than one reply for a request can be generated or the processing is complex or needs to happen asynchronously, use a com.rti.connext.requestreply.Replier<TReq,TRep> instead.

See also

com.rti.connext.requestreply.Replier<TReq,TRep>

com.rti.connext.requestreply.SimpleReplierListener<TReq,TRep>

SimpleReplier example (p. 152)

8.315.2 Constructor & Destructor Documentation

8.315.2.1 SimpleReplier() [1/2]

```
SimpleReplier (
            DomainParticipant participant,
           String serviceName,
            SimpleReplierListener< TReq, TRep > listener,
            TypeSupport requestTypeSupport,
            TypeSupport replyTypeSupport )
```

Creates a new **SimpleReplier** (p. 1692).

See also

com.rti.connext.requestreply.SimpleReplierParams<TReq,TRep>

com.rti.connext.requestreply.SimpleReplierListener<TReq,TRep>

References **SimpleReplierParams< TReq, TRep >.setServiceName()**.

8.315.2.2 SimpleReplier() [2/2]

```
SimpleReplier (
            SimpleReplierParams< TReq, TRep > params )
```

Creates a new **SimpleReplier** (p. 1692).

See also

com.rti.connext.requestreply.SimpleReplierParams<TReq,TRep>

com.rti.connext.requestreply.SimpleReplierListener<TReq,TRep>

References **ReplierParams< TReq, TRep >.setReplierListener()**.

8.315.3 Member Function Documentation

8.315.3.1 close()

```
void close ( )
```

Releases the resources created by this **SimpleReplier** (p. 1692).

See also

com.rti.connext.requestreply.Replier<TReq,TRep>.close() (p. 1546)

References **Replier< TReq, TRep >.close()**.

8.316 SimpleReplierListener< TReq, TRep > Interface Template Reference

The listener called by a **SimpleReplier** (p. 1692).

Inherits EventListener.

Inherited by RequestReplyHowTo.MySimpleReplierListener.

Public Member Functions

- TRep **onRequestAvailable** (**Sample**< TReq > request)

  User callback that receives a request and provides a reply.
- void **returnLoan** (TRep reply)

  Returns a previously generated reply to the user.

8.316.1 Detailed Description

The listener called by a **SimpleReplier** (p. 1692).

See also

com.rti.connext.requestreply.SimpleReplier<TReq,TRep>

SimpleReplier example (p. 152)

8.316.2 Member Function Documentation

8.316.2.1 onRequestAvailable()

```
TRep onRequestAvailable (
              Sample< TReq > request )
```

User callback that receives a request and provides a reply.

This operation gets called when a request is available and expects a reply that is automatically sent. Immediately after that, **returnLoan** (p. 1695) is called.

Parameters

request	The received request

Returns

A reply for that request

8.316.2.2 returnLoan()

```
void returnLoan (
             TRep reply )
```

Returns a previously generated reply to the user.

This operation is always called right after sending the reply created by **onRequestAvailable** (p. 1695). It can be used to release any resources from the reply creation. If there are no resources to release, the implementation body can be empty.

Parameters

reply	The reply previously provided in **onRequestAvailable** (p. 1695)

8.317 SimpleReplierParams< TReq, TRep > Class Template Reference

Contains the parameters for creating a com.rti.connext.requestreply.SimpleReplier<TReq,TRep>

Inherits EntityParams.

Public Member Functions

- **SimpleReplierParams** (**DomainParticipant** participant, **SimpleReplierListener**< TReq, TRep > listener, **TypeSupport** requestTypeSupport, **TypeSupport** replyTypeSupport)

  *Creates **SimpleReplierParams** (p. 1696) with the parameters that a **SimpleReplier** (p. 1692) always needs.*
- **SimpleReplierParams**< TReq, TRep > **setServiceName** (String serviceName)

  *The service name the **Replier** (p. 1542) offers and Requesters use to match.*
- **SimpleReplierParams**< TReq, TRep > **setRequestTopicName** (String requestTopicName)

  Sets a specific request topic name.
- **SimpleReplierParams**< TReq, TRep > **setReplyTopicName** (String replyTopicName)

  Sets a specific reply topic name.
- **SimpleReplierParams**< TReq, TRep > **setDatawriterQos** (**DataWriterQos** dataWriterQos)

  Sets the quality of service of the reply DataWriter.
- **SimpleReplierParams**< TReq, TRep > **setDatareaderQos** (**DataReaderQos** dataReaderQos)

  Sets the quality of service of the request DataReader.
- **SimpleReplierParams**< TReq, TRep > **setQosProfile** (String qosLibraryName, String qosProfileName)

  Sets a QoS profile for the entities in this replier.
- **SimpleReplierParams**< TReq, TRep > **setPublisher** (**Publisher** publisher)

  Sets a specific Publisher.
- **SimpleReplierParams**< TReq, TRep > **setSubscriber** (**Subscriber** subscriber)

  Sets a specific Subscriber.

8.317.1 Detailed Description

Contains the parameters for creating a com.rti.connext.requestreply.SimpleReplier<TReq,TRep>

The parameters for a **SimpleReplier** (p. 1692) are identical to those of the **Replier** (p. 1542), except for the **SimpleReplierListener** (p. 1694), which is required and has a different user callback.

See also

com.rti.connext.requestreply.ReplierParams<TReq,TRep>

8.317.2 Constructor & Destructor Documentation

8.317.2.1 SimpleReplierParams()

```
SimpleReplierParams (
            DomainParticipant participant,
            SimpleReplierListener< TReq, TRep > listener,
            TypeSupport requestTypeSupport,
            TypeSupport replyTypeSupport )
```

Creates **SimpleReplierParams** (p. 1696) with the parameters that a **SimpleReplier** (p. 1692) always needs.

In addition to these parameters, a **SimpleReplier** (p. 1692) needs either:

- A service name (**setServiceName(String)** (p. 1697)), or
- Custom topic names (**setReplyTopicName(String)** (p. 1698) and **setRequestTopicName(String)** (p. 1697)).

The other parameters are optional.

Parameters

participant	The DomainParticipant that this **SimpleReplier** (p. 1692) uses to join a domain.
listener	The listener where the user callback that a **SimpleReplier** (p. 1692) calls is implemented.
requestTypeSupport	The type support for type `TReq`
replyTypeSupport	The type support for type `Trep`

8.317.3 Member Function Documentation

8.317.3.1 setServiceName()

```
SimpleReplierParams< TReq, TRep > setServiceName (
            String serviceName )
```

The service name the **Replier** (p. 1542) offers and Requesters use to match.

See also

com.rti.connext.requestreply.RequesterParams.setServiceName(String) (p. 1587)

Referenced by **SimpleReplier< TReq, TRep >.SimpleReplier()**.

8.317.3.2 setRequestTopicName()

```
SimpleReplierParams< TReq, TRep > setRequestTopicName (
            String requestTopicName )
```

Sets a specific request topic name.

This is an alternative to **setServiceName(String)** (p. 1697)

8.317.3.3 setReplyTopicName()

```
SimpleReplierParams< TReq, TRep > setReplyTopicName (
            String replyTopicName )
```

Sets a specific reply topic name.

This is an alternative to **setServiceName(String)** (p. 1697)

8.317.3.4 setDatawriterQos()

```
SimpleReplierParams< TReq, TRep > setDatawriterQos (
            DataWriterQos dataWriterQos )
```

Sets the quality of service of the reply DataWriter.

See also

setQosProfile(String,String) (p. 1698)

8.317.3.5 setDatareaderQos()

```
SimpleReplierParams< TReq, TRep > setDatareaderQos (
            DataReaderQos dataReaderQos )
```

Sets the quality of service of the request DataReader.

See also

setQosProfile(String,String) (p. 1698)

8.317.3.6 setQosProfile()

```
SimpleReplierParams< TReq, TRep > setQosProfile (
            String qosLibraryName,
            String qosProfileName )
```

Sets a QoS profile for the entities in this replier.

See also

com.rti.connext.requestreply.ReplierParams<TReq,TRep>.setQosProfile(String,String) (p. 1558)

8.317.3.7 setPublisher()

```
SimpleReplierParams< TReq, TRep > setPublisher (
             Publisher publisher )
```

Sets a specific Publisher.

See also

com.rti.connext.requestreply.RequesterParams.setPublisher(Publisher) (p. 1589)

8.317.3.8 setSubscriber()

```
SimpleReplierParams< TReq, TRep > setSubscriber (
             Subscriber subscriber )
```

Sets a specific Subscriber.

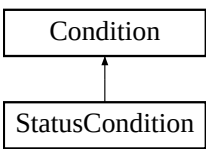
See also

com.rti.connext.requestreply.RequesterParams.setSubscriber(Subscriber) (p. 1589)

8.318 StatusCondition Interface Reference

<<***interface***>> *(p. 156)* A specific **com.rti.dds.infrastructure.Condition** (p. 429) that is associated with each **com.↩ rti.dds.infrastructure.Entity** (p. 1029).

Inheritance diagram for StatusCondition:



Public Member Functions

- int **get_enabled_statuses** ()

  *Get the list of statuses enabled on an **com.rti.dds.infrastructure.Entity*** (p. *1029*).
- void **set_enabled_statuses** (int mask)

  This operation defines the list of communication statuses that determine the `trigger_value` *of the **com.rti.dds.↩ infrastructure.StatusCondition*** (p. *1699*).
- **Entity get_entity** ()

  *Get the **com.rti.dds.infrastructure.Entity*** (p. *1029*) *associated with the **com.rti.dds.infrastructure.StatusCondition*** (p. *1699*).

8.318.1 Detailed Description

<<***interface***>> (p. *156*) A specific **com.rti.dds.infrastructure.Condition** (p. 429) that is associated with each **com.↩ rti.dds.infrastructure.Entity** (p. 1029).

The `trigger_value` of the **com.rti.dds.infrastructure.StatusCondition** (p. 1699) depends on the communication status of that entity (e.g., arrival of data, loss of information, etc.), 'filtered' by the set of `enabled_statuses` on the **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

See also

Status Kinds (p. 262)

com.rti.dds.infrastructure.WaitSet (p. 1973), **com.rti.dds.infrastructure.Condition** (p. 429)

com.rti.dds.infrastructure.Listener (p. 1236)

8.318.2 Member Function Documentation

8.318.2.1 get_enabled_statuses()

```
int get_enabled_statuses ( )
```

Get the list of statuses enabled on an **com.rti.dds.infrastructure.Entity** (p. 1029).

Returns

list of enabled statuses.

8.318.2.2 set_enabled_statuses()

```
void set_enabled_statuses (
            int mask )
```

This operation defines the list of communication statuses that determine the `trigger_value` of the **com.rti.dds.↩ infrastructure.StatusCondition** (p. 1699).

This operation may change the `trigger_value` of the **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

com.rti.dds.infrastructure.WaitSet (p. 1973) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **com.rti.dds.infrastructure.WaitSet** (p. 1973) to which the **com.rti.dds.↩ infrastructure.StatusCondition** (p. 1699) is attached is potentially affected by this operation.

If this function is not invoked, the default list of enabled statuses includes all the statuses.

Parameters

mask	<<***in***>> *(*p. *156)* the list of enables statuses (see **Status Kinds** (p. 262))

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.318.2.3 get_entity()

```
Entity get_entity ( )
```

Get the **com.rti.dds.infrastructure.Entity** (p. 1029) associated with the **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

There is exactly one **com.rti.dds.infrastructure.Entity** (p. 1029) assocated with each **com.rti.dds.infrastructure.↩ StatusCondition** (p. 1699).

Returns

com.rti.dds.infrastructure.Entity (p. 1029) associated with the **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

8.319 StatusKind Class Reference

Type for *status* kinds.

Static Public Attributes

- static final int **INCONSISTENT_TOPIC_STATUS**

  Another topic exists with the same name but different characteristics.
- static final int **OFFERED_DEADLINE_MISSED_STATUS**

  *The deadline that the **com.rti.dds.publication.DataWriter** (*p. *553) has committed through its **com.rti.dds.↩ infrastructure.DeadlineQosPolicy** (*p. *632) was not respected for a specific instance.*
- static final int **REQUESTED_DEADLINE_MISSED_STATUS**

  *The deadline that the **com.rti.dds.subscription.DataReader** (*p. *450) was expecting through its **com.rti.dds.↩ infrastructure.DeadlineQosPolicy** (*p. *632) was not respected for a specific instance.*
- static final int **OFFERED_INCOMPATIBLE_QOS_STATUS**

  *A **QosPolicy** (*p. *1501) value was incompatible with what was requested.*
- static final int **REQUESTED_INCOMPATIBLE_QOS_STATUS**

  *A **QosPolicy** (*p. *1501) value was incompatible with what is offered.*
- static final int **SAMPLE_LOST_STATUS**

A sample has been lost (i.e., was never received).

- static final int **SAMPLE_REJECTED_STATUS**

  A (received) sample has been rejected.

- static final int **DATA_ON_READERS_STATUS**

  New data is available.

- static final int **DATA_AVAILABLE_STATUS**

  One or more new data samples have been received.

- static final int **LIVELINESS_LOST_STATUS**

  *The liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed to through its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1243) was not respected, thus **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer alive.*

- static final int **LIVELINESS_CHANGED_STATUS**

  *The liveliness of one or more **com.rti.dds.publication.DataWriter** (p. 553) that were writing instances read through the **com.rti.dds.subscription.DataReader** (p. 450) has changed. Some **com.rti.dds.publication.DataWriter** (p. 553) have become alive or not_alive.*

- static final int **PUBLICATION_MATCHED_STATUS**

  *The **com.rti.dds.publication.DataWriter** (p. 553) has found **com.rti.dds.subscription.DataReader** (p. 450) that matches the **com.rti.dds.topic.Topic** (p. 1807) and has compatible QoS.*

- static final int **SUBSCRIPTION_MATCHED_STATUS**

  *The **com.rti.dds.subscription.DataReader** (p. 450) has found **com.rti.dds.publication.DataWriter** (p. 553) that matches the **com.rti.dds.topic.Topic** (p. 1807) and has compatible QoS.*

- static final int **INVALID_LOCAL_IDENTITY_ADVANCE_NOTICE_STATUS**

  *<<**extension**>> (p. 155) The local **com.rti.dds.domain.DomainParticipant** (p. 670) has or is about to have an invalid identity credential.*

- static final int **SERVICE_REQUEST_ACCEPTED_STATUS**

  *<<**extension**>> (p. 155) A **com.rti.dds.publication.DataWriter** (p. 553) has been issued a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675)*

- static final int **DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS**

  *<<**extension**>> (p. 155) A **com.rti.dds.publication.DataWriter** (p. 553) has received an application-level acknowledgment for a sample*

- static final int **DATA_WRITER_INSTANCE_REPLACED_STATUS**

  *<<**extension**>> (p. 155) A **com.rti.dds.publication.DataWriter** (p. 553) instance has been replaced*

- static final int **RELIABLE_WRITER_CACHE_CHANGED_STATUS**

  *<<**extension**>> (p. 155) The number of unacknowledged samples in a reliable writer's cache has changed such that it has reached a pre-defined trigger point.*

- static final int **RELIABLE_READER_ACTIVITY_CHANGED_STATUS**

  *<<**extension**>> (p. 155) One or more reliable readers has become active or inactive.*

- static final int **DDS_DATA_WRITER_CACHE_STATUS**

  *<<**extension**>> (p. 155) The status of the writer's cache.*

- static final int **DDS_DATA_WRITER_PROTOCOL_STATUS**

  *<<**extension**>> (p. 155) The status of a writer's internal protocol related metrics*

- static final int **DDS_DATA_READER_CACHE_STATUS**

  *<<**extension**>> (p. 155) The status of the reader's cache.*

- static final int **DATA_READER_PROTOCOL_STATUS**

  *<<**extension**>> (p. 155) The status of a reader's internal protocol related metrics*

- static final int **STATUS_MASK_NONE**

  No bits are set.

- static final int **STATUS_MASK_ALL**

  All bits are set.

8.319.1 Detailed Description

Type for *status* kinds.

Each concrete **com.rti.dds.infrastructure.Entity** (p. 1029) is associated with a set of `*Status` objects whose values represent the communication status of that **com.rti.dds.infrastructure.Entity** (p. 1029).

The communication statuses whose changes can be communicated to the application depend on the **com.rti.dds.↩ infrastructure.Entity** (p. 1029).

Each status value can be accessed with a corresponding method on the **com.rti.dds.infrastructure.Entity** (p. 1029). The changes on these status values cause activation of the corresponding **com.rti.dds.infrastructure.Status↩ Condition** (p. 1699) objects and trigger invocation of the proper **com.rti.dds.infrastructure.Listener** (p. 1236) objects to asynchronously inform the application. Note that not all statuses will activate the **com.rti.dds.infrastructure.↩ StatusCondition** (p. 1699) or have a corresponding listener callback. Refer to the documentation of the individual statuses for that information.

See also

com.rti.dds.infrastructure.Entity (p. 1029), **com.rti.dds.infrastructure.StatusCondition** (p. 1699), **com.rti.↩ dds.infrastructure.Listener** (p. 1236)

8.319.2 Member Data Documentation

8.319.2.1 INCONSISTENT_TOPIC_STATUS

```
final int INCONSISTENT_TOPIC_STATUS  [static]
```

Another topic exists with the same name but different characteristics.

Entity:

com.rti.dds.topic.Topic (p. 1807)

Status:

com.rti.dds.topic.InconsistentTopicStatus (p. 1149)

Listener:

com.rti.dds.topic.TopicListener (p. 1822)

8.319.2.2 OFFERED_DEADLINE_MISSED_STATUS

```
final int OFFERED_DEADLINE_MISSED_STATUS  [static]
```

The deadline that the **com.rti.dds.publication.DataWriter** (p. 553) has committed through its **com.rti.dds.↩ infrastructure.DeadlineQosPolicy** (p. 632) was not respected for a specific instance.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

QoS:

DEADLINE (p. 217)

Status:

com.rti.dds.publication.OfferedDeadlineMissedStatus (p. 1339)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.3 REQUESTED_DEADLINE_MISSED_STATUS

```
final int REQUESTED_DEADLINE_MISSED_STATUS  [static]
```

The deadline that the **com.rti.dds.subscription.DataReader** (p. 450) was expecting through its **com.rti.dds.↩ infrastructure.DeadlineQosPolicy** (p. 632) was not respected for a specific instance.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

QoS:

DEADLINE (p. 217)

Status:

com.rti.dds.subscription.RequestedDeadlineMissedStatus (p. 1560)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.4 OFFERED_INCOMPATIBLE_QOS_STATUS

```
final int OFFERED_INCOMPATIBLE_QOS_STATUS  [static]
```

A **QosPolicy** (p. 1501) value was incompatible with what was requested.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Status:

com.rti.dds.publication.OfferedIncompatibleQosStatus (p. 1340)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.5 REQUESTED_INCOMPATIBLE_QOS_STATUS

```
final int REQUESTED_INCOMPATIBLE_QOS_STATUS  [static]
```

A **QosPolicy** (p. 1501) value was incompatible with what is offered.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Status:

com.rti.dds.subscription.RequestedIncompatibleQosStatus (p. 1561)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.6 SAMPLE_LOST_STATUS

```
final int SAMPLE_LOST_STATUS  [static]
```

A sample has been lost (i.e., was never received).

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Status:

com.rti.dds.subscription.SampleLostStatus (p. 1648)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.7 SAMPLE_REJECTED_STATUS

```
final int SAMPLE_REJECTED_STATUS  [static]
```

A (received) sample has been rejected.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

QoS:

RESOURCE_LIMITS (p. 259)

Status:

com.rti.dds.subscription.SampleRejectedStatus (p. 1657)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.8 DATA_ON_READERS_STATUS

```
final int DATA_ON_READERS_STATUS  [static]
```

New data is available.

Entity:

com.rti.dds.subscription.Subscriber (p. 1730)

Listener:

com.rti.dds.subscription.SubscriberListener (p. 1755)

8.319.2.9 DATA_AVAILABLE_STATUS

```
final int DATA_AVAILABLE_STATUS  [static]
```

One or more new data samples have been received.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.10 LIVELINESS_LOST_STATUS

```
final int LIVELINESS_LOST_STATUS  [static]
```

The liveliness that the **com.rti.dds.publication.DataWriter** (p. 553) has committed to through its **com.rti.dds.↩ infrastructure.LivelinessQosPolicy** (p. 1243) was not respected, thus **com.rti.dds.subscription.DataReader** (p. 450) entities will consider the **com.rti.dds.publication.DataWriter** (p. 553) as no longer alive.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

QoS:

LIVELINESS (p. 239)

Status:

com.rti.dds.publication.LivelinessLostStatus (p. 1242)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.11 LIVELINESS_CHANGED_STATUS

```
final int LIVELINESS_CHANGED_STATUS  [static]
```

The liveliness of one or more **com.rti.dds.publication.DataWriter** (p. 553) that were writing instances read through the **com.rti.dds.subscription.DataReader** (p. 450) has changed. Some **com.rti.dds.publication.DataWriter** (p. 553) have become alive or not_alive.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

QoS:

LIVELINESS (p. 239)

Status:

com.rti.dds.subscription.LivelinessChangedStatus (p. 1239)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.12 PUBLICATION_MATCHED_STATUS

```
final int PUBLICATION_MATCHED_STATUS  [static]
```

The **com.rti.dds.publication.DataWriter** (p. 553) has found **com.rti.dds.subscription.DataReader** (p. 450) that matches the **com.rti.dds.topic.Topic** (p. 1807) and has compatible QoS.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Status:

com.rti.dds.publication.PublicationMatchedStatus (p. 1463)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.13 SUBSCRIPTION_MATCHED_STATUS

```
final int SUBSCRIPTION_MATCHED_STATUS  [static]
```

The **com.rti.dds.subscription.DataReader** (p. 450) has found **com.rti.dds.publication.DataWriter** (p. 553) that matches the **com.rti.dds.topic.Topic** (p. 1807) and has compatible QoS.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Status:

com.rti.dds.subscription.SubscriptionMatchedStatus (p. 1773)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 497)

8.319.2.14 INVALID_LOCAL_IDENTITY_ADVANCE_NOTICE_STATUS

```
final int INVALID_LOCAL_IDENTITY_ADVANCE_NOTICE_STATUS  [static]
```

<<***extension***>> *(p. 155)* The local **com.rti.dds.domain.DomainParticipant** (p. 670) has or is about to have an invalid identity credential.

Enables a **com.rti.dds.domain.DomainParticipant** (p. 670) callback that is called when the local **com.rti.dds.domain.DomainParticipant** (p. 670) has or is about to have an invalid identity credential. Currently, this status is only triggered when enabling the RTI Security Plugins. Please refer to the `RTI Security Plugins User's Manual` for more information.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Listener:

com.rti.dds.domain.DomainParticipantListener (p. 792)

8.319.2.15 SERVICE_REQUEST_ACCEPTED_STATUS

```
final int SERVICE_REQUEST_ACCEPTED_STATUS  [static]
```

<<***extension***>> *(*p. *155)* A **com.rti.dds.publication.DataWriter** (p. 553) has been issued a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675)

Enables a **com.rti.dds.publication.DataWriter** (p. 553) callback that is called when a **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) has been accepted and dispatched to the DataWriter.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.16 DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS

```
final int DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS  [static]
```

<<***extension***>> *(*p. *155)* A **com.rti.dds.publication.DataWriter** (p. 553) has received an application-level acknowledgment for a sample

Enables a **com.rti.dds.publication.DataWriter** (p. 553) callback that is called when an application-level acknowledgment from a **com.rti.dds.subscription.DataReader** (p. 450) is received. The callback is called for each sample that is application-level acknowledged. Changes to this status do not trigger a **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.17 DATA_WRITER_INSTANCE_REPLACED_STATUS

```
final int DATA_WRITER_INSTANCE_REPLACED_STATUS  [static]
```

<<***extension***>> *(*p. *155)* A **com.rti.dds.publication.DataWriter** (p. 553) instance has been replaced

Enables a **com.rti.dds.publication.DataWriter** (p. 553) callback that is called when an instance in the writer queue is replaced.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.18 RELIABLE_WRITER_CACHE_CHANGED_STATUS

```
final int RELIABLE_WRITER_CACHE_CHANGED_STATUS  [static]
```

<<***extension***>> *(*p. *155)* The number of unacknowledged samples in a reliable writer's cache has changed such that it has reached a pre-defined trigger point.

This status is considered changed at the following times: the cache is empty (i.e., contains no unacknowledged samples), the cache is full (i.e., the sample count has reached the value specified in **com.rti.dds.infrastructure.Resource↩ LimitsQosPolicy.max_samples** (p. 1592)), or the number of samples has reached a high (see **com.rti.dds.↩ infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1607)) or low (see **com.rti.dds.infrastructure.↩ RtpsReliableWriterProtocol_t.low_watermark** (p. 1607)) watermark.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Status:

com.rti.dds.publication.ReliableWriterCacheChangedStatus (p. 1535)

Listener:

com.rti.dds.publication.DataWriterListener (p. 589)

8.319.2.19 RELIABLE_READER_ACTIVITY_CHANGED_STATUS

```
final int RELIABLE_READER_ACTIVITY_CHANGED_STATUS  [static]
```

<<***extension***>> *(*p. *155)* One or more reliable readers has become active or inactive.

A reliable reader is considered active by a reliable writer with which it is matched if that reader acknowledges the samples it has been sent in a timely fashion. For the definition of "timely" in this case, see **com.rti.dds.infrastructure.Rtps↩ ReliableWriterProtocol_t** (p. 1605) and **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1532).

See also

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t (p. 1605)

com.rti.dds.publication.ReliableReaderActivityChangedStatus (p. 1532)

8.319.2.20 DDS_DATA_WRITER_CACHE_STATUS

```
final int DDS_DATA_WRITER_CACHE_STATUS  [static]
```

<<***extension***>> *(*p. *155)* The status of the writer's cache.

Changes to this status do not trigger a **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

8.319.2.21 DDS_DATA_WRITER_PROTOCOL_STATUS

```
final int DDS_DATA_WRITER_PROTOCOL_STATUS  [static]
```

<<***extension***>> *(*p. *155)* The status of a writer's internal protocol related metrics

The status of a writer's internal protocol-related metrics, such as the number of samples pushed, pulled, and filtered and the status of wire protocol traffic. Changes to this status information do not trigger a **com.rti.dds.infrastructure.↩ StatusCondition** (p. 1699).

8.319.2.22 DDS_DATA_READER_CACHE_STATUS

```
final int DDS_DATA_READER_CACHE_STATUS  [static]
```

<<***extension***>> *(*p. *155)* The status of the reader's cache.

Changes to this status do not trigger a **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

8.319.2.23 DATA_READER_PROTOCOL_STATUS

```
final int DATA_READER_PROTOCOL_STATUS  [static]
```

<<***extension***>> *(*p. *155)* The status of a reader's internal protocol related metrics

The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic. Changes to this status do not trigger a **com.rti.dds.infrastructure.StatusCondition** (p. 1699).

8.320 StreamKind Class Reference

Indicates if the samples are **TopicQuery** (p. 1830) samples or not.

Static Public Attributes

- static final int **LIVE_STREAM** = 0x0001 << 0

  Sample is a live data sample.
- static final int **TOPIC_QUERY_STREAM** = 0x0001 << 1

  *Sample is a **TopicQuery** (*p. *1830) sample.*
- static final int **ANY_STREAM** = 0xffff

  Any stream kind LIVE_STREAM | TOPIC_QUERY_STREAM.

8.320.1 Detailed Description

Indicates if the samples are **TopicQuery** (p. 1830) samples or not.

8.320.2 Member Data Documentation

8.320.2.1 LIVE_STREAM

```
final int LIVE_STREAM = 0x0001 << 0  [static]
```

Sample is a live data sample.

8.320.2.2 TOPIC_QUERY_STREAM

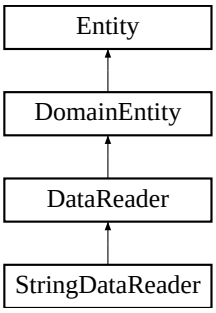
```
final int TOPIC_QUERY_STREAM = 0x0001 << 1  [static]
```

Sample is a **TopicQuery** (p. 1830) sample.

8.321 StringDataReader Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataReader` < com.rti.dds.infrastructure.String >.

Inheritance diagram for StringDataReader:



Public Member Functions

- void **read** (**StringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view↩ _states, int instance_states)

  *Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **take** (**StringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view↩ _states, int instance_states)

  *Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).*
- void **read_w_condition** (**StringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩ **Condition** condition)

  *Accesses via **com.rti.dds.type.builtin.StringDataReader.read** (p. 1715) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).*
- void **take_w_condition** (**StringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **Read**↩ **Condition** condition)

  *Analogous to **com.rti.dds.type.builtin.StringDataReader.read_w_condition** (p. 1715) except it accesses samples via the **com.rti.dds.type.builtin.StringDataReader.take** (p. 1715) operation.*
- String **read_next_sample** (**SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*
- String **take_next_sample** (**SampleInfo** sample_info)

  *Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).*

Additional Inherited Members

8.321.1 Detailed Description

<<***interface***>> *(*p. *156)* Instantiates `DataReader` < com.rti.dds.infrastructure.String >.

See also

com.rti.ndds.example.FooDataReader (p. 1067)
com.rti.dds.subscription.DataReader (p. 450)

8.321.2 Member Function Documentation

8.321.2.1 read()

```
void read (
              StringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read (p. 1069)

References **DataReader.read_untyped()**.

8.321.2.2 take()

```
void take (
              StringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
             int sample_states,
             int view_states,
             int instance_states )
```

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.take (p. 1071)

References **DataReader.take_untyped()**.

8.321.2.3 read_w_condition()

```
void read_w_condition (
              StringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Accesses via **com.rti.dds.type.builtin.StringDataReader.read** (p. 1715) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1514).

See also

com.rti.ndds.example.FooDataReader.read_w_condition (p. 1076)

References **DataReader.read_w_condition_untyped()**.

8.321.2.4 take_w_condition()

```
void take_w_condition (
              StringSeq received_data,
              SampleInfoSeq info_seq,
             int max_samples,
              ReadCondition condition )
```

Analogous to **com.rti.dds.type.builtin.StringDataReader.read_w_condition** (p. 1715) except it accesses samples via the **com.rti.dds.type.builtin.StringDataReader.take** (p. 1715) operation.

See also

com.rti.ndds.example.FooDataReader.take_w_condition (p. 1077)

References **DataReader.take_w_condition_untyped()**.

8.321.2.5 read_next_sample()

```
String read_next_sample (
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

com.rti.ndds.example.FooDataReader.read_next_sample (p. 1078)

References **DataReader.read_next_sample_untyped()**.

8.321.2.6 take_next_sample()

```
String take_next_sample (
              SampleInfo sample_info )
```

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 450).

See also

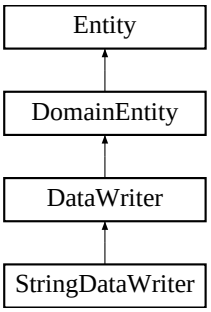
com.rti.ndds.example.FooDataReader.take_next_sample (p. 1079)

References **DataReader.take_next_sample_untyped()**.

8.322 StringDataWriter Class Reference

<<***interface***>> *(p. 156)* Instantiates `DataWriter` < com.rti.dds.infrastructure.String >.

Inheritance diagram for StringDataWriter:



Public Member Functions

- void **write** (String instance_data, **InstanceHandle_t** handle)

  Modifies the value of a string data instance.
- void **write_w_timestamp** (String instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  *Performs the same function as **com.rti.dds.type.builtin.StringDataWriter.write** (p. 1718) except that it also provides the value for the `source_timestamp`.*

8.322.1 Detailed Description

<<***interface***>> *(p. 156)* Instantiates `DataWriter` < com.rti.dds.infrastructure.String >.

See also

com.rti.ndds.example.FooDataWriter (p. 1097)

com.rti.dds.publication.DataWriter (p. 553)

8.322.2 Member Function Documentation

8.322.2.1 write()

```
void write (
             String instance_data,
              InstanceHandle_t handle )
```

Modifies the value of a string data instance.

See also

com.rti.ndds.example.FooDataWriter.write (p. 1105)

References **DataWriter.write_untyped()**.

8.322.2.2 write_w_timestamp()

```
void write_w_timestamp (
             String instance_data,
              InstanceHandle_t handle,
              Time_t source_timestamp )
```

Performs the same function as **com.rti.dds.type.builtin.StringDataWriter.write** (p. 1718) except that it also provides the value for the `source_timestamp`.

See also

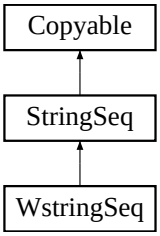
com.rti.ndds.example.FooDataWriter.write_w_timestamp (p. 1109)

References **DataWriter.write_w_timestamp_untyped()**.

8.323 StringSeq Class Reference

Declares IDL `sequence` < com.rti.dds.infrastructure.String > .

Inheritance diagram for StringSeq:



Public Member Functions

- **StringSeq** ()

  Constructs an empty sequence of strings with an initial maximum of zero.
- **StringSeq** (int initialMaximum)

  Constructs an empty sequence of strings with the given initial maximum.
- **StringSeq** (Collection<?> strings)

  Constructs a new sequence containing the given strings.
- final Object **copy_from** (Object src)

Static Public Member Functions

- static void **readStringArray** (String[ ] value, CdrObjectInput in, int length) throws IOException
- static void **writeStringArray** (String[ ] value, CdrObjectOutput out, int length, int maxStringLength) throws IOException

8.323.1 Detailed Description

Declares IDL `sequence` < com.rti.dds.infrastructure.String > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ String > with value type semantics.

StringSeq (p. 1718) is a sequence that contains strings.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.323.2 Constructor & Destructor Documentation

8.323.2.1 StringSeq() [1/3]

```
StringSeq ( )
```

Constructs an empty sequence of strings with an initial maximum of zero.

Referenced by **StringSeq.copy_from()**.

8.323.2.2 StringSeq() [2/3]

```
StringSeq (
            int initialMaximum )
```

Constructs an empty sequence of strings with the given initial maximum.

8.323.2.3 StringSeq() [3/3]

```
StringSeq (
            Collection<?> strings )
```

Constructs a new sequence containing the given strings.

Parameters

strings	the initial contents of this sequence

Exceptions

NullPointerException	if the input collection is null

8.323.3 Member Function Documentation

8.323.3.1 copy_from()

```
final Object copy_from (
            Object src )
```

Copy data into `this` object from another. The result of this method is that both `this` and `src` will be the same size and contain the same data.

Parameters

src	The Object which contains the data to be copied

Returns

Generally, return `this` but special cases (such as `Enum`) exist.

Exceptions

NullPointerException	If `src` is null OR if there are null objects contained in this sequence.
ClassCastException	If `src` is not the same type as `this`.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

References **StringSeq.StringSeq()**.

Referenced by **QueryConditionParams.copy_from()**, **TopicQuerySelection.copy_from()**, **MonitoringMetric↩ Selection.MonitoringMetricSelection()**, and **QueryConditionParams.QueryConditionParams()**.

8.323.3.2 readStringArray()

```
static void readStringArray (
             String[] value,
             CdrObjectInput in,
             int length ) throws IOException  [static]
```

Read array of strings. The length specified **must** match the expected length of array. Otherwise, the stream will be positioned incorrectly, leading to corrupt reads. The length of array must be at least the value of length parameter (otherwise, ArrayOutOfBoundException will be thrown).

Parameters

value	array to read into
in	Interface for reading object in CDR encoding.
length	the length of array (<= value.length)

8.323.3.3 writeStringArray()

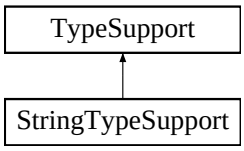
```
static void writeStringArray (
            String[] value,
            CdrObjectOutput out,
            int length,
            int maxStringLength ) throws IOException  [static]
```

Write array of string up to the specified length

8.324 StringTypeSupport Class Reference

<<***interface***>> *(*p. *156)* String type support.

Inheritance diagram for StringTypeSupport:



Public Member Functions

- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, String src)

  <<***extension***>> *(*p. *155) Serializes the input sample into a CDR buffer of octets.*
- long **serialize_to_cdr_buffer** (byte[ ] buffer, long length, String src, short representation)

  <<***extension***>> *(*p. *155) Serializes the input sample into a buffer of octets.*
- String **data_to_string** (String src, **PrintFormatProperty** property)

  <<***extension***>> *(*p. *155) Get the string representation of an input sample.*
- String **data_to_string** (String src)

  <<***extension***>> *(*p. *155) Get the string representation of an input sample.*
- String **deserialize_from_cdr_buffer** (byte[ ] buffer, long length)

  <<***extension***>> *(*p. *155) Deserializes a sample from a buffer of octets.*

Static Public Member Functions

- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of the com.rti.dds.infrastructure.String data type.
- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the com.rti.dds.infrastructure.String data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.
- static String **get_type_name** ()

  Get the default name for the com.rti.dds.infrastructure.String type.

8.324.1 Detailed Description

<<***interface***>> *(*p. *156)* String type support.

8.324.2 Member Function Documentation

8.324.2.1 register_type()

```
static void register_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to communicate to RTI Connext the existence of the com.rti.dds.infrastructure.String data type.

By default, The com.rti.dds.infrastructure.String built-in type is automatically registered when a DomainParticipant is created using the type_name returned by com.rti.dds.type.builtin.com.rti.dds.type.builtin.StringTypeSupport.get_type↩ _name. Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same com.rti.dds.type.builtin.com.rti.dds.type.builtin.StringTypeSupport with a **com.rti.dds.domain.DomainParticipant** (p. 670) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 670) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

participant	<<***in***>> *(*p. *156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to register the data type com.rti.dds.infrastructure.String with. Cannot be null.
type_name	<<***in***>> *(*p. *156)* the type name under with the data type com.rti.dds.infrastructure.String is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1807). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 706).) The name may not be null or longer than 255 characters.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

MT Safety:

UNSAFE on the FIRST call. It is not safe for two threads to simultaneously make the first call to register a type. Subsequent calls are thread safe.

See also

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.324.2.2 unregister_type()

```
static void unregister_type (
              DomainParticipant participant,
             String type_name )  [static]
```

Allows an application to unregister the com.rti.dds.infrastructure.String data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition

The com.rti.dds.infrastructure.String type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1807) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **com.rti.dds.topic.Topic** (p. 1807) is associated with the type, the operation will fail with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

participant	<<***in***>> *(p. 156)* the **com.rti.dds.domain.DomainParticipant** (p. 670) to unregister the data type com.rti.dds.infrastructure.String from. Cannot be null.
type_name	<<***in***>> *(p. 156)* the type name under with the data type com.rti.dds.infrastructure.String is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_BAD_PARAMETER** (p. 1594) or **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595)

MT Safety:

SAFE.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.StringTypeSupport.register_type

8.324.2.3 get_type_name()

```
static String get_type_name ( )  [static]
```

Get the default name for the com.rti.dds.infrastructure.String type.

Can be used for calling com.rti.dds.type.builtin.com.rti.dds.type.builtin.StringTypeSupport.register_type or creating **com.rti.dds.topic.Topic** (p. 1807).

Returns

default name for the com.rti.dds.infrastructure.String type.

See also

com.rti.dds.type.builtin.com.rti.dds.type.builtin.StringTypeSupport.register_type

com.rti.dds.domain.DomainParticipant.create_topic (p. 706)

8.324.2.4 serialize_to_cdr_buffer() [1/2]

```
long serialize_to_cdr_buffer (
             byte[] buffer,
             long length,
             String src )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a CDR buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

Referenced by **StringTypeSupport.data_to_string()**.

8.324.2.5 serialize_to_cdr_buffer() [2/2]

```
long serialize_to_cdr_buffer (
             byte[] buffer,
             long length,
             String src,
             short representation )
```

<<***extension***>> *(p. 155)* Serializes the input sample into a buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.serialize_to_cdr_buffer (p. 1121)

8.324.2.6 data_to_string() [1/2]

```
String data_to_string (
            String src,
             PrintFormatProperty property )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **DynamicData.from_cdr_buffer()**, **DynamicData.PROPERTY_DEFAULT**, **StringTypeSupport.↩ serialize_to_cdr_buffer()**, and **DynamicData.to_string()**.

Referenced by **StringTypeSupport.data_to_string()**.

8.324.2.7 data_to_string() [2/2]

```
String data_to_string (
            String src )
```

<<***extension***>> *(*p. *155)* Get the string representation of an input sample.

Use the default values for **com.rti.dds.topic.PrintFormatProperty** (p. 1387) to create the output string.

See also

com.rti.ndds.example.FooTypeSupport.data_to_string (p. 1124)

References **StringTypeSupport.data_to_string()**.

8.324.2.8 deserialize_from_cdr_buffer()

```
String deserialize_from_cdr_buffer (
            byte[] buffer,
            long length )
```

<<***extension***>> *(*p. *155)* Deserializes a sample from a buffer of octets.

See also

com.rti.ndds.example.FooTypeSupport.deserialize_from_cdr_buffer (p. 1123)

8.325 StructMember Class Reference

A description of a member of a struct.

Inherits Serializable.

Public Member Functions

- **StructMember** (String **name**, boolean **is_pointer**, short **bits**, boolean **is_key**, **TypeCode type**)

Public Attributes

- String **name**

  The name of the struct member.
- **TypeCode type**

  The type of the struct member.
- boolean **is_pointer**

  Indicates whether the struct member is a pointer or not.
- short **bits**

  Number of bits of a bitfield member.
- boolean **is_key**

  Indicates if the struct member is a key member or not.
- int **id**

  The member ID.
- boolean **is_optional**

  Indicates if the struct member is optional or required.

8.325.1 Detailed Description

A description of a member of a struct.

See also

com.rti.dds.typecode.TypeCodeFactory.create_struct_tc (p. 1923)

8.325.2 Constructor & Destructor Documentation

8.325.2.1 StructMember()

```
StructMember (
            String name,
            boolean is_pointer,
            short bits,
            boolean is_key,
             TypeCode type )
```

Constructs a **StructMember** (p. 1727) object initialized with the given values.

References **StructMember.bits**, **StructMember.is_key**, **StructMember.is_pointer**, **TypeCode.MEMBER_ID_INVALID**, **StructMember.name**, and **StructMember.type**.

8.325.3 Member Data Documentation

8.325.3.1 name

```
String name
```

The name of the struct member.

Cannot be null.

Referenced by **TypeCode.member_name()**, and **StructMember.StructMember()**.

8.325.3.2 type

```
TypeCode type
```

The type of the struct member.

Cannot be null.

Referenced by **TypeCode.member_type()**, and **StructMember.StructMember()**.

8.325.3.3 is_pointer

```
boolean is_pointer
```

Indicates whether the struct member is a pointer or not.

Referenced by **TypeCode.is_member_pointer()**, and **StructMember.StructMember()**.

8.325.3.4 bits

```
short bits
```

Number of bits of a bitfield member.

If the struct member is a bitfield, this field contains the number of bits of the bitfield. Otherwise, bits should contain **com.rti.dds.typecode.TypeCode.NOT_BITFIELD** (p. 1920).

Referenced by **TypeCode.member_bitfield_bits()**, and **StructMember.StructMember()**.

8.325.3.5 is_key

```
boolean is_key
```

Indicates if the struct member is a key member or not.

Referenced by **TypeCode.is_member_key()**, and **StructMember.StructMember()**.

8.325.3.6 id

```
int id
```

The member ID.

Use **com.rti.dds.typecode.TypeCode.MEMBER_ID_INVALID** (p. 1918) to have the member ID automatically assigned.

Referenced by **TypeCode.member_id()**.

8.325.3.7 is_optional

```
boolean is_optional
```

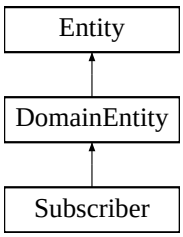
Indicates if the struct member is optional or required.

Referenced by **TypeCode.is_member_required()**.

8.326 Subscriber Interface Reference

<<***interface***>> *(*p. *156)* A subscriber is the object responsible for actually receiving data from a subscription.

Inheritance diagram for Subscriber:



Public Member Functions

- void **get_default_datareader_qos** (**DataReaderQos** qos)

  Copies the default ***com.rti.dds.subscription.DataReaderQos*** *(*p. *517) values into the provided* ***com.rti.dds.subscription.DataReaderQos*** *(*p. *517) instance.*
- void **set_default_datareader_qos** (**DataReaderQos** qos)

  Sets the default ***com.rti.dds.subscription.DataReaderQos*** *(*p. *517) values for this subscriber.*
- void **set_default_datareader_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Set the default* ***com.rti.dds.subscription.DataReaderQos*** *(*p. *517) values for this subscriber based on the input XML QoS profile.*
- **DataReader create_datareader** (**TopicDescription** topic, **DataReaderQos** qos, **DataReaderListener** listener, int mask)

  Creates a ***com.rti.dds.subscription.DataReader*** *(*p. *450) that will be attached and belong to the* ***com.rti.dds.subscription.Subscriber*** *(*p. *1730).*
- **DataReader create_datareader_with_profile** (**TopicDescription** topic, String library_name, String profile_name, **DataReaderListener** listener, int mask)

  <<***extension***>> *(*p. *155) Creates a* ***com.rti.dds.subscription.DataReader*** *(*p. *450) object using the* ***com.rti.dds.subscription.DataReaderQos*** *(*p. *517) associated with the input XML QoS profile.*
- void **delete_datareader** (**DataReader** a_datareader)

  Deletes a ***com.rti.dds.subscription.DataReader*** *(*p. *450) that belongs to the* ***com.rti.dds.subscription.Subscriber*** *(*p. *1730).*
- **DataReader lookup_datareader** (String topic_name)

  Retrieves an existing ***com.rti.dds.subscription.DataReader*** *(*p. *450).*
- void **get_datareaders** (**DataReaderSeq** readers, int sample_states, int view_states, int instance_states)

  Allows the application to access the ***com.rti.dds.subscription.DataReader*** *(*p. *450) objects that contain samples with the specified* `sample_states`, `view_states` *and* `instance_states`.
- void **get_all_datareaders** (**DataReaderSeq** readers)

  Retrieve all the DataReaders created from this ***Subscriber*** *(*p. *1730).*
- void **notify_datareaders** ()

  Invokes the operation ***com.rti.dds.subscription.DataReaderListener::on_data_available()*** *(*p. *500) on the* ***com.rti.dds.subscription.DataReaderListener*** *(*p. *497) objects attached to contained* ***com.rti.dds.subscription.DataReader*** *(*p. *450) entities with com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS that is considered changed as described in* ***Changes in read communication status*** *(*p. *264).*
- void **set_qos** (**SubscriberQos** qos)

Sets the subscriber QoS.

- void **set_qos_with_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Change the QoS of this subscriber using the input XML QoS profile.*
- void **get_qos** (**SubscriberQos** qos)

  Gets the subscriber QoS.
- String **get_default_library** ()

  *<<**extension**>> (p. 155) Gets the default XML library associated with a **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **set_default_library** (String library_name)

  *<<**extension**>> (p. 155) Sets the default XML library for a **com.rti.dds.subscription.Subscriber** (p. 1730).*
- String **get_default_profile** ()

  *<<**extension**>> (p. 155) Gets the default XML profile associated with a **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **set_default_profile** (String library_name, String profile_name)

  *<<**extension**>> (p. 155) Sets the default XML profile for a **com.rti.dds.subscription.Subscriber** (p. 1730).*
- String **get_default_profile_library** ()

  *<<**extension**>> (p. 155) Gets the library where the default XML QoS profile is contained for a **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **set_listener** (**SubscriberListener** l, int mask)

  Sets the subscriber listener.
- **SubscriberListener** **get_listener** ()

  Get the subscriber listener.
- void **call_listenerT** (int mask)

  Call the subscriber listener.
- void **begin_access** ()

  *Indicates that the application is about to access the data samples in any of the **com.rti.dds.subscription.DataReader** (p. 450) objects attached to the **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **end_access** ()

  *Indicates that the application has finished accessing the data samples in **com.rti.dds.subscription.DataReader** (p. 450) objects managed by the **com.rti.dds.subscription.Subscriber** (p. 1730).*
- void **copy_from_topic_qos** (**DataReaderQos** datareader_qos, **TopicQos** topic_qos)

  *Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.dds.subscription.DataReaderQos** (p. 517).*
- **DomainParticipant** **get_participant** ()

  *Returns the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.subscription.Subscriber** (p. 1730) belongs.*
- void **delete_contained_entities** ()

  *Deletes all the entities that were created by means of the "create" operation on the **com.rti.dds.subscription.Subscriber** (p. 1730).*
- **DataReader** **lookup_datareader_by_name** (String datareader_name)

  *<<**extension**>> (p. 155) Retrieves a **com.rti.dds.subscription.DataReader** (p. 450) contained within the **com.rti.dds.subscription.Subscriber** (p. 1730) the **com.rti.dds.subscription.DataReader** (p. 450) entity name.*

Static Public Attributes

- static final **DataReaderQos** **DATAREADER_QOS_DEFAULT**

  Special value for creating data reader with default QoS.
- static final **DataReaderQos** **DATAREADER_QOS_USE_TOPIC_QOS** = new **DataReaderQos**()

Special value for creating ***com.rti.dds.subscription.DataReader*** *(p. 450) with a combination of the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 517) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1824).*

- static final **DataReaderQos** **DATAREADER_QOS_PRINT_ALL** = new **DataReaderQos**()

  Special value which can be supplied to ***com.rti.dds.subscription.DataReaderQos.toString(DataReaderQos baseQos, QosPrintFormat format)*** *(p. 520) indicating that all of the QoS should be printed.*

8.326.1 Detailed Description

<<***interface***>> *(p. 156)* A subscriber is the object responsible for actually receiving data from a subscription.

QoS:

com.rti.dds.subscription.SubscriberQos (p. 1756)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS

Listener:

com.rti.dds.subscription.SubscriberListener (p. 1755)

A subscriber acts on the behalf of one or several **com.rti.dds.subscription.DataReader** (p. 450) objects that are related to it. When it receives data (from the other parts of the system), it builds the list of concerned **com.rti.dds.subscription.DataReader** (p. 450) objects and then indicates to the application that data is available through its listener or by enabling related conditions.

The application can access the list of concerned **com.rti.dds.subscription.DataReader** (p. 450) objects through the operation **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1741) and then access the data available through operations on the **com.rti.dds.subscription.DataReader** (p. 450).

The following operations may be called even if the **com.rti.dds.subscription.Subscriber** (p. 1730) is not enabled. Other operations will the value **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) if called on a disabled **com.rti.dds.subscription.Subscriber** (p. 1730):

- **com.rti.dds.infrastructure.Entity.enable** (p. 1032),
- **com.rti.dds.subscription.Subscriber.set_qos** (p. 1744), **com.rti.dds.subscription.Subscriber.get_qos** (p. 1745) , idref_Subscriber_set_qos_with_profile
- **com.rti.dds.subscription.Subscriber.set_listener** (p. 1748), **com.rti.dds.subscription.Subscriber.get_listener** (p. 1749),
- **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034)
- **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735), **com.rti.dds.subscription.Subscriber.create_datareader_with_profile** (p. 1737), **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1752), **com.rti.dds.subscription.Subscriber.delete_datareader** (p. 1739),

- **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734), **com.rti.dds.subscription.↩ Subscriber.set_default_datareader_qos_with_profile** (p. 1734), **com.rti.dds.subscription.Subscriber.↩ get_default_datareader_qos** (p. 1733), **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1746), **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747)

All operations except for **com.rti.dds.subscription.Subscriber.set_qos** (p. 1744), **com.rti.dds.subscription.↩ Subscriber.set_qos_with_profile** (p. 1744), **com.rti.dds.subscription.Subscriber.get_qos** (p. 1745), **com.rti.↩ dds.subscription.Subscriber.set_listener** (p. 1748), **com.rti.dds.subscription.Subscriber.get_listener** (p. 1749), **com.rti.dds.infrastructure.Entity.enable** (p. 1032) and **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735) may fail with **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

Operations Allowed in Listener Callbacks (p. 1238)

8.326.2 Member Function Documentation

8.326.2.1 get_default_datareader_qos()

```
void get_default_datareader_qos (
              DataReaderQos qos )
```

Copies the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values into the provided **com.rti.dds.↩ subscription.DataReaderQos** (p. 517) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **com.rti.dds.subscription.↩ Subscriber.set_default_datareader_qos** (p. 1734), or **com.rti.dds.subscription.Subscriber.set_default_↩ datareader_qos_with_profile** (p. 1734), or else, if the call was never made, the default values from is owning **com.rti.dds.domain.DomainParticipant** (p. 670).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a subscriber while another thread may be simultaneously calling **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734)

Parameters

qos	<<***inout***>> *(p. 156)* **com.rti.dds.subscription.DataReaderQos** (p. 517) to be filled-up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT (p. 80)

com.rti.dds.subscription.Subscriber.create_datareader (p. 1735)

8.326.2.2 set_default_datareader_qos()

```
void set_default_datareader_qos (
              DataReaderQos qos )
```

Sets the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this subscriber.

This call causes the default values inherited from the owning **com.rti.dds.domain.DomainParticipant** (p. 670) to be overridden.

This default value will be used for newly created **com.rti.dds.subscription.DataReader** (p. 450) if **com.rti.dds.↩ subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) is specified as the `qos` parameter when **com.rti.↩ dds.subscription.Subscriber.create_datareader** (p. 1735) is called.

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **com.rti.↩ dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a subscriber while another thread may be simultaneously calling **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734), **com.rti.↩ dds.subscription.Subscriber.get_default_datareader_qos** (p. 1733) or calling **com.rti.dds.subscription.↩ Subscriber.create_datareader** (p. 1735) with **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_↩ DEFAULT** (p. 80) as the `qos` parameter.

Parameters

qos	<<***in***>> *(p. 156)* The default **com.rti.dds.subscription.DataReaderQos** (p. 517) to be set to. The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734) had never been called. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), or or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

8.326.2.3 set_default_datareader_qos_with_profile()

```
void set_default_datareader_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 517) values for this subscriber based on the input XML QoS profile.

This default value will be used for newly created **com.rti.dds.subscription.DataReader** (p. 450) if **com.rti.dds.↩ subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) is specified as the `qos` parameter when **com.rti.↩ dds.subscription.Subscriber.create_datareader** (p. 1735) is called.

Precondition

The **com.rti.dds.subscription.DataReaderQos** (p. 517) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.subscription.Subscriber** (p. 1730) while another thread may be simultaneously calling **com.rti.dds.subscription.Subscriber.set_default_↩ datareader_qos** (p. 1734), **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1733) or calling **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1735) with **com.rti.dds.subscription.↩ Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) as the `qos` parameter.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1746)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747)).

If the input profile cannot be found the method fails with **com.rti.dds.infrastructure.RETCODE_ERROR** (p. 1595).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596)

See also

com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT (p. 80)

com.rti.dds.subscription.Subscriber.create_datareader_with_profile (p. 1737)

8.326.2.4 create_datareader()

```
DataReader create_datareader (
            TopicDescription topic,
            DataReaderQos qos,
            DataReaderListener listener,
           int mask )
```

Creates a **com.rti.dds.subscription.DataReader** (p. 450) that will be attached and belong to the **com.rti.dds.↩ subscription.Subscriber** (p. 1730).

For each application-defined type **com.rti.ndds.example.Foo** (p. 1066), there is an implied, auto-generated class **com.rti.ndds.example.FooDataReader** (p. 1067) (an incarnation of FooDataReader) that extends **com.rti.dds.subscription.DataReader** (p. 450) and contains the operations to read data of type **com.rti.ndds.↩ example.Foo** (p. 1066).

Note that a common application pattern to construct the QoS for the **com.rti.dds.subscription.DataReader** (p. 450) is to:

- Retrieve the QoS policies on the associated **com.rti.dds.topic.Topic** (p. 1807) by means of the **com.rti.dds.↩ topic.Topic.get_qos** (p. 1810) operation.
- Retrieve the default **com.rti.dds.subscription.DataReader** (p. 450) qos by means of the **com.rti.dds.↩ subscription.Subscriber.get_default_datareader_qos** (p. 1733) operation.
- Combine those two QoS policies (for example, using **com.rti.dds.subscription.Subscriber.copy_from_topic↩ _qos** (p. 1751)) and selectively modify policies as desired
- Use the resulting QoS policies to construct the **com.rti.dds.subscription.DataReader** (p. 450).

When a **com.rti.dds.subscription.DataReader** (p. 450) is created, only those transports already registered are available to the **com.rti.dds.subscription.DataReader** (p. 450). See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **com.rti.↩ dds.subscription.Subscriber.set_default_datareader_qos** (p. 1734).

Precondition

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **com.rti.dds.↩ subscription.DataReader** (p. 450) will be created.

The given **com.rti.dds.topic.TopicDescription** (p. 1820) must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.

If `qos` is **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 80), `topic` cannot be **com.rti.dds.topic.MultiTopic** (p. 1320), or else this method will return NULL.

Parameters

topic	<<***in***>> *(*p. *156)* The **com.rti.dds.topic.TopicDescription** (p. 1820) that the **com.rti.dds.subscription.DataReader** (p. 450) will be associated with. Cannot be NULL.
qos	<<***in***>> *(*p. *156)* The qos of the **com.rti.dds.subscription.DataReader** (p. 450). The special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT** (p. 80) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 450) should be created with the default **com.rti.dds.subscription.DataReaderQos** (p. 517) set in the **com.rti.dds.subscription.Subscriber** (p. 1730). If **com.rti.dds.topic.TopicDescription** (p. 1820) is of type **com.rti.dds.topic.Topic** (p. 1807) or **com.rti.dds.topic.ContentFilteredTopic** (p. 436), the special value **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 80) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 450) should be created with the combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 517) set on the **com.rti.dds.subscription.Subscriber** (p. 1730) and the **com.rti.dds.topic.TopicQos** (p. 1824) (in the case of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), the **com.rti.dds.topic.TopicQos** (p. 1824) of the related **com.rti.dds.topic.Topic** (p. 1807)). if **com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 80) is used, `topic` cannot be a **com.rti.dds.topic.MultiTopic** (p. 1320). Cannot be NULL.
listener	<<***in***>> *(*p. *156)* The listener of the **com.rti.dds.subscription.DataReader** (p. 450).
mask	<<***in***>> *(*p. *156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.subscription.DataReader** (p. 450) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

com.rti.dds.subscription.Subscriber.create_datareader_with_profile (p. 1737)

com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1733)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1751)

com.rti.dds.subscription.DataReader.set_listener (p. 459)

8.326.2.5 create_datareader_with_profile()

```
DataReader create_datareader_with_profile (
             TopicDescription topic,
            String library_name,
            String profile_name,
```

```
 DataReaderListener listener,
int mask )
```

<<***extension***>> *(p. 155)* Creates a **com.rti.dds.subscription.DataReader** (p. 450) object using the **com.rti.dds.↩ subscription.DataReaderQos** (p. 517) associated with the input XML QoS profile.

The **com.rti.dds.subscription.DataReader** (p. 450) will be attached and belong to the **com.rti.dds.subscription.↩ Subscriber** (p. 1730).

For each application-defined type **com.rti.ndds.example.Foo** (p. 1066), there is an implied, auto-generated class **com.rti.ndds.example.FooDataReader** (p. 1067) (an incarnation of FooDataReader) that extends **com.rti.dds.subscription.DataReader** (p. 450) and contains the operations to read data of type **com.rti.ndds.↩ example.Foo** (p. 1066).

When a **com.rti.dds.subscription.DataReader** (p. 450) is created, only those transports already registered are available to the **com.rti.dds.subscription.DataReader** (p. 450). See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

Precondition

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **com.rti.dds.↩ subscription.DataReader** (p. 450) will be created.

The given **com.rti.dds.topic.TopicDescription** (p. 1820) must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.

Parameters

topic	<<***in***>> *(p. 156)* The **com.rti.dds.topic.TopicDescription** (p. 1820) that the **com.rti.dds.subscription.DataReader** (p. 450) will be associated with. Cannot be NULL.
library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1746)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747)).
listener	<<***in***>> *(p. 156)* The listener of the **com.rti.dds.subscription.DataReader** (p. 450).
mask	<<***in***>> *(p. 156)*. Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Returns

A **com.rti.dds.subscription.DataReader** (p. 450) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1807) or NULL if an error occurred.

See also

com.rti.ndds.example.FooDataReader (p. 1067)

Specifying QoS on entities (p. 255) for information on setting QoS before entity creation

com.rti.dds.subscription.DataReaderQos (p. 517) for rules on consistency among QoS

com.rti.dds.subscription.Subscriber.DATAREADER_QOS_DEFAULT (p. 80)

com.rti.dds.subscription.Subscriber.DATAREADER_QOS_USE_TOPIC_QOS (p. 80)

com.rti.dds.subscription.Subscriber.create_datareader (p. 1735)

com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1733)

com.rti.dds.topic.Topic.set_qos (p. 1809)

com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1751)

com.rti.dds.subscription.DataReader.set_listener (p. 459)

8.326.2.6 delete_datareader()

```
void delete_datareader (
            DataReader a_datareader )
```

Deletes a **com.rti.dds.subscription.DataReader** (p. 450) that belongs to the **com.rti.dds.subscription.Subscriber** (p. 1730).

Precondition

If the **com.rti.dds.subscription.DataReader** (p. 450) does not belong to the **com.rti.dds.subscription.Subscriber** (p. 1730), or if ther are any existing **com.rti.dds.subscription.ReadCondition** (p. 1514) or **com.rti.dds.subscription.QueryCondition** (p. 1510) objects that are attached to the **com.rti.dds.subscription.DataReader** (p. 450), or if there are outstanding loans on samples (as a result of a call to read(), take(), or one of the variants thereof), the operation fails with the error **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Postcondition

Listener installed on the **com.rti.dds.subscription.DataReader** (p. 450) will not be called after this method completes successfully.

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Parameters

a_datareader	*<<**in**>> (p. 156)* The **com.rti.dds.subscription.DataReader** (p. 450) to be deleted.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.326.2.7 lookup_datareader()

```
DataReader lookup_datareader (
            String topic_name )
```

Retrieves an existing **com.rti.dds.subscription.DataReader** (p. 450).

Use this operation on the built-in **com.rti.dds.subscription.Subscriber** (p. 1730) (**Built-in Topics** (p. 51)) to access the built-in **com.rti.dds.subscription.DataReader** (p. 450) entities for the built-in topics.

The built-in **com.rti.dds.subscription.DataReader** (p. 450) is created when this operation is called on a built-in topic for the first time. The built-in **com.rti.dds.subscription.DataReader** (p. 450) is deleted automatically when the **com.↩ rti.dds.domain.DomainParticipant** (p. 670) is deleted.

To ensure that builtin **com.rti.dds.subscription.DataReader** (p. 450) entities receive all the discovery traffic, it is suggested that you lookup the builtin **com.rti.dds.subscription.DataReader** (p. 450) before the **com.rti.dds.domain.↩ DomainParticipant** (p. 670) is enabled. Looking up builtin **com.rti.dds.subscription.DataReader** (p. 450) may implicitly register builtin transports due to creation of **com.rti.dds.subscription.DataReader** (p. 450) (see **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered). Therefore, if you are want to modify builtin transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up builtin DataReaders is:

- Create a disabled **com.rti.dds.domain.DomainParticipant** (p. 670).
- (optional) Modify builtin transport properties
- Call **com.rti.dds.domain.DomainParticipant.get_builtin_subscriber()** (p. 720).
- Call **com.rti.dds.subscription.Subscriber.lookup_datareader()** (p. 1740).
- Call **enable()** (p. 1032) on the DomainParticipant.

Parameters

topic_name	<<***in***>> *(p. 156)* Name of the **com.rti.dds.topic.TopicDescription** (p. 1820) that the retrieved **com.rti.dds.subscription.DataReader** (p. 450) is attached to. Cannot be NULL.

Returns

A **com.rti.dds.subscription.DataReader** (p. 450) that belongs to the **com.rti.dds.subscription.Subscriber** (p. 1730) attached to the **com.rti.dds.topic.TopicDescription** (p. 1820) with `topic_name`. If no such **com.↩ rti.dds.subscription.DataReader** (p. 450) exists, this operation returns NULL.

The returned **com.rti.dds.subscription.DataReader** (p. 450) may be enabled or disabled.

If more than one **com.rti.dds.subscription.DataReader** (p. 450) is attached to the **com.rti.dds.subscription.↩ Subscriber** (p. 1730), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **com.rti.dds.subscription.DataReader** (p. 450) in one thread while another thread is simultaneously creating or destroying that **com.rti.dds.subscription.DataReader** (p. 450).

8.326.2.8 get_datareaders()

```
void get_datareaders (
              DataReaderSeq readers,
             int sample_states,
             int view_states,
             int instance_states )
```

Allows the application to access the **com.rti.dds.subscription.DataReader** (p. 450) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a **begin_access()** (p. 1749)/end_access() block, or if the **com.rti.dds.infrastructure.PresentationQosPolicy::access_scope** (p. 1382) of the **com.rti.dds.subscription.Subscriber** (p. 1730) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS or com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS, or if the **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) of the **com.rti.dds.subscription.Subscriber** (p. 1730) is com.rti.dds.infrastructure.false, the returned collection is a 'set' containing each **com.rti.dds.subscription.DataReader** (p. 450) at most once, in no specified order.

If the application is within a **begin_access()** (p. 1749)/end_access() block, and the **PRESENTATION** (p. 247) policy of the **com.rti.dds.subscription.Subscriber** (p. 1730) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS or com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS, and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1383) in the **com.rti.dds.subscription.Subscriber** (p. 1730) is com.rti.dds.infrastructure.true, the returned collection is a 'list' of DataReaders where a **DataReader** (p. 450) may appear more than one time.

To retrieve the samples in the order they were published across DataWriters of the same group (**com.rti.dds.publication.Publisher** (p. 1466) configured with com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS), the application should read()/take() from each **DataReader** (p. 450) in the same order as it appears in the output sequence. The application will move to the next **DataReader** (p. 450) when the read()/take() operation fails with **com.rti.dds.infrastructure.RETCODE_NO_DATA** (p. 1597).

Parameters

readers	<<***inout***>> *(p. 156)* a **com.rti.dds.subscription.DataReaderSeq** (p. 543) object where the set or list of readers will be returned. Cannot be NULL.
sample_states	<<***in***>> *(p. 156)* the returned **DataReader** (p. 450) must contain samples that have one of these `sample_states`.
view_states	<<***in***>> *(p. 156)* the returned **DataReader** (p. 450) must contain samples that have one of these `view_states`.
instance_states	<<***in***>> *(p. 156)* the returned **DataReader** (p. 450) must contain samples that have one of these `instance_states`.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

Access to data samples (p. 78)

com.rti.dds.subscription.Subscriber.begin_access (p. 1749)

com.rti.dds.subscription.Subscriber.end_access (p. 1750)

PRESENTATION (p. 247)

8.326.2.9 get_all_datareaders()

```
void get_all_datareaders (
              DataReaderSeq readers )
```

Retrieve all the DataReaders created from this **Subscriber** (p. 1730).

Parameters

readers	<<***inout***>> *(*p. *156)* Sequence where the DataReaders will be added

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.326.2.10 notify_datareaders()

```
void notify_datareaders ( )
```

Invokes the operation **com.rti.dds.subscription.DataReaderListener::on_data_available()** (p. 500) on the **com.rti.↩ dds.subscription.DataReaderListener** (p. 497) objects attached to contained **com.rti.dds.subscription.DataReader** (p. 450) entities with com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS that is considered changed as described in **Changes in read communication status** (p. 264).

This operation is typically invoked from the **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1756) operation in the **com.rti.dds.subscription.SubscriberListener** (p. 1755). That way the **com.rti.dds.↩ subscription.SubscriberListener** (p. 1755) can delegate to the **com.rti.dds.subscription.DataReaderListener** (p. 497) objects the handling of the data.

The operation will notify the data readers that have a `sample_state` of com.rti.dds.subscription.SampleStateKind.↩ SampleStateKind.NOT_READ_SAMPLE_STATE, `view_state` of **com.rti.dds.subscription.SampleStateKind.↩ ANY_SAMPLE_STATE** (p. 88) and `instance_state` of **com.rti.dds.subscription.InstanceStateKind.ANY_↩ INSTANCE_STATE** (p. 90).

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.326.2.11 set_qos()

```
void set_qos (
              SubscriberQos qos )
```

Sets the subscriber QoS.

This operation modifies the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1730).

The **com.rti.dds.subscription.SubscriberQos.group_data** (p. 1760), **com.rti.dds.subscription.SubscriberQos.↩ partition** (p. 1760) and **com.rti.dds.subscription.SubscriberQos.entity_factory** (p. 1760) can be changed. The other policies are immutable.

Parameters

qos	*<<**in**>> (p. 156)* **com.rti.dds.subscription.SubscriberQos** (p. 1756) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.subscription.Subscriber** (p. 1730) is enabled. The special value **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 47) can be used to indicate that the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1730) should be changed to match the current default **com.rti.dds.subscription.SubscriberQos** (p. 1756) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.subscription.SubscriberQos (p. 1756) for rules on consistency among QoS

set_qos (abstract) (p. 1030)

Operations Allowed in Listener Callbacks (p. 1238)

8.326.2.12 set_qos_with_profile()

```
void set_qos_with_profile (
             String library_name,
             String profile_name )
```

<<***extension***>> *(p. 155)* Change the QoS of this subscriber using the input XML QoS profile.

This operation modifies the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1730).

The **com.rti.dds.subscription.SubscriberQos.group_data** (p. 1760), **com.rti.dds.subscription.SubscriberQos.↩ partition** (p. 1760) and **com.rti.dds.subscription.SubscriberQos.entity_factory** (p. 1760) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1746)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596), or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596).

See also

com.rti.dds.subscription.SubscriberQos (p. 1756) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1238)

8.326.2.13 get_qos()

```
void get_qos (
              SubscriberQos qos )
```

Gets the subscriber QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***in***>> *(p. 156)* **com.rti.dds.subscription.SubscriberQos** (p. 1756) to be filled in. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.326.2.14 get_default_library()

```
String get_default_library ( )
```

<<***extension***>> *(p. 155)* Gets the default XML library associated with a **com.rti.dds.subscription.Subscriber** (p. 1730).

Returns

The default library or null if the default library was not set.

See also

com.rti.dds.subscription.Subscriber.set_default_library (p. 1746)

8.326.2.15 set_default_library()

```
void set_default_library (
             String library_name )
```

<<***extension***>> *(p. 155)* Sets the default XML library for a **com.rti.dds.subscription.Subscriber** (p. 1730).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this **Subscriber** (p. 1730)'s operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **com.rti.dds.subscription.Subscriber** (p. 1730) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 670) (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).

Parameters

library_name	<<***in***>> *(p. 156)* Library name. If library_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.subscription.Subscriber.get_default_library (p. 1746)

8.326.2.16 get_default_profile()

```
String get_default_profile ( )
```

<<***extension***>> *(p. 155)* Gets the default XML profile associated with a **com.rti.dds.subscription.Subscriber** (p. 1730).

Returns

The default profile or null if the default profile was not set.

See also

com.rti.dds.subscription.Subscriber.set_default_profile (p. 1747)

8.326.2.17 set_default_profile()

```
void set_default_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Sets the default XML profile for a **com.rti.dds.subscription.Subscriber** (p. 1730).

This method specifies the profile that will be used as the default the next time a default **Subscriber** (p. 1730) profile is needed during a call to one of this **Subscriber** (p. 1730)'s operations. When calling a **com.rti.dds.subscription.Subscriber** (p. 1730) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **com.rti.dds.subscription.Subscriber** (p. 1730) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 670) (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

This method does not set the default QoS for **com.rti.dds.subscription.DataReader** (p. 450) objects created by this **com.rti.dds.subscription.Subscriber** (p. 1730); for this functionality, use **com.rti.dds.subscription.Subscriber.set_default_datareader_qos_with_profile** (p. 1734) (you may pass in NULL after having called **set_default_profile()** (p. 1747)).

This method does not set the default QoS for newly created Subscribers; for this functionality, use **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos_with_profile** (p. 688).

Parameters

library_name	<<***in***>> *(p. 156)* The library name containing the profile.
profile_name	<<***in***>> *(p. 156)* The profile name. If profile_name is null any previous default is unset.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.subscription.Subscriber.get_default_profile (p. 1747)

com.rti.dds.subscription.Subscriber.get_default_profile_library (p. 1748)

8.326.2.18 get_default_profile_library()

```
String get_default_profile_library ( )
```

<<***extension***>> *(p. 155)* Gets the library where the default XML QoS profile is contained for a **com.rti.dds.subscription.Subscriber** (p. 1730).

The default profile library is automatically set when **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1747) is called.

This library can be different than the **com.rti.dds.subscription.Subscriber** (p. 1730) default library (see **com.rti.dds.subscription.Subscriber.get_default_library** (p. 1746)).

Returns

The default profile library or null if the default profile was not set.

See also

com.rti.dds.subscription.Subscriber.set_default_profile (p. 1747)

8.326.2.19 set_listener()

```
void set_listener (
              SubscriberListener l,
             int mask )
```

Sets the subscriber listener.

Parameters

l	*<<**in**>> (p. 156)* **com.rti.dds.subscription.SubscriberListener** (p. 1755) to set to.
mask	*<<**in**>> (p. 156)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.subscription.SubscriberListener** (p. 1755).

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.326.2.20 get_listener()

```
SubscriberListener get_listener ( )
```

Get the subscriber listener.

Returns

com.rti.dds.subscription.SubscriberListener (p. 1755) of the **com.rti.dds.subscription.Subscriber** (p. 1730).

See also

get_listener (abstract) (p. 1032)

8.326.2.21 call_listenerT()

```
void call_listenerT (
             int mask )
```

Call the subscriber listener.

8.326.2.22 begin_access()

```
void begin_access ( )
```

Indicates that the application is about to access the data samples in any of the **com.rti.dds.subscription.DataReader** (p. 450) objects attached to the **com.rti.dds.subscription.Subscriber** (p. 1730).

If the **com.rti.dds.infrastructure.PresentationQosPolicy::access_scope** (p. 1382) of the **com.rti.dds.subscription.Subscriber** (p. 1730) is com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS or com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy::ordered_access** (p. 1383) is com.rti.dds.infrastructure.true, the application is required to use this operation to access the samples in order across DataWriters of the same group (**com.rti.dds.publication.Publisher** (p. 1466) with **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) set to com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS).

In the above case, this operation must be called prior to calling any of the sample-accessing operations, **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1741) on the **com.rti.dds.subscription.Subscriber** (p. 1730), and **com.rti.ndds.example.FooDataReader.read** (p. 1069), **com.rti.ndds.example.FooDataReader.take** (p. 1071), **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076), and **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077) on any **com.rti.dds.subscription.DataReader** (p. 450).

Once the application has finished accessing the data samples, it must call **com.rti.dds.subscription.Subscriber.end_access** (p. 1750)

The application is not required to call **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) / **com.rti.dds.subscription.Subscriber.end_access** (p. 1750) to access the samples in order if the **PRESENTATION** (p. 247) policy in the **com.rti.dds.publication.Publisher** (p. 1466) has **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1382) set to something other than com.rti.dds.infrastructure.PresentationQosPolicyAccessScopeKind.PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS. In this case, calling **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) / **com.rti.dds.subscription.Subscriber.end_access** (p. 1750) is not considered an error and has no effect.

Calls to **com.rti.dds.subscription.Subscriber.begin_access** (p. 1749) / **com.rti.dds.subscription.Subscriber.end_access** (p. 1750) may be nested and must be balanced.

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

See also

Access to data samples (p. 78)

com.rti.dds.subscription.Subscriber.get_datareaders (p. 1741)

PRESENTATION (p. 247)

8.326.2.23 end_access()

```
void end_access ( )
```

Indicates that the application has finished accessing the data samples in **com.rti.dds.subscription.DataReader** (p. 450) objects managed by the **com.rti.dds.subscription.Subscriber** (p. 1730).

This operation must be used to close a corresponding **begin_access()** (p. 1749).

This call must close a previous call to **com.rti.dds.subscription.Subscriber.begin_access()** (p. 1749), otherwise the operation will fail with the error **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597).

8.326.2.24 copy_from_topic_qos()

```
void copy_from_topic_qos (
               DataReaderQos datareader_qos,
               TopicQos topic_qos )
```

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.dds.↩ subscription.DataReaderQos** (p. 517).

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1824) to the corresponding policies in the **com.rti.dds.↩ subscription.DataReaderQos** (p. 517) (replacing values in the **com.rti.dds.subscription.DataReaderQos** (p. 517), if present).

This is a "convenience" operation most useful in combination with the operations **com.rti.dds.subscription.↩ Subscriber.get_default_datareader_qos** (p. 1733) and **com.rti.dds.topic.Topic.get_qos** (p. 1810). The operation **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1751) can be used to merge the **com.rti.dds.↩ subscription.DataReader** (p. 450) default QoS policies with the corresponding ones on the **com.rti.dds.topic.Topic** (p. 1807). The resulting QoS can then be used to create a new **com.rti.dds.subscription.DataReader** (p. 450), or set its QoS.

This operation does not check the resulting **com.rti.dds.subscription.DataReaderQos** (p. 517) for consistency. This is because the 'merged' **com.rti.dds.subscription.DataReaderQos** (p. 517) may not be the final one, as the application can still modify some policies prior to applying the policies to the **com.rti.dds.subscription.DataReader** (p. 450).

Parameters

datareader_qos	*<<**inout**>> (p. 156)* **com.rti.dds.subscription.DataReaderQos** (p. 517) to be filled-up. Cannot be NULL.
topic_qos	*<<**in**>> (p. 156)* **com.rti.dds.topic.TopicQos** (p. 1824) to be merged with **com.rti.dds.subscription.DataReaderQos** (p. 517). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.326.2.25 get_participant()

```
DomainParticipant get_participant ( )
```

Returns the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.subscription.Subscriber** (p. 1730) belongs.

Returns

the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.subscription.Subscriber** (p. 1730) belongs.

8.326.2.26 delete_contained_entities()

```
void delete_contained_entities ( )
```

Deletes all the entities that were created by means of the "create" operation on the **com.rti.dds.subscription.↩ Subscriber** (p. 1730).

Deletes all contained **com.rti.dds.subscription.DataReader** (p. 450) objects. This pattern is applied recursively. In this manner, the operation **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1752) on the **com.↩ rti.dds.subscription.Subscriber** (p. 1730) will end up deleting all the entities recursively contained in the **com.rti.↩ dds.subscription.Subscriber** (p. 1730), including the **com.rti.dds.subscription.QueryCondition** (p. 1510), **com.↩ rti.dds.subscription.ReadCondition** (p. 1514), and **com.rti.dds.subscription.TopicQuery** (p. 1830) objects belonging to the contained **com.rti.dds.subscription.DataReader** (p. 450).

The operation will fail with **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) if any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained **com.rti.dds.↩ subscription.DataReader** (p. 450) cannot be deleted because the application has called a **com.rti.ndds.example.↩ FooDataReader.read** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) operation and has not called the corresponding **com.rti.ndds.example.FooDataReader.return_loan** (p. 1094) operation to return the loaned samples.

Once **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1752) completes successfully, the application may delete the **com.rti.dds.subscription.Subscriber** (p. 1730).

MT Safety:

UNSAFE. It is not safe to delete an entity while another thread may be simultaneously calling an API that uses the entity.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598)

8.326.2.27 lookup_datareader_by_name()

```
DataReader lookup_datareader_by_name (
            String datareader_name )
```

<<***extension***>> *(p. 155)* Retrieves a **com.rti.dds.subscription.DataReader** (p. 450) contained within the **com.rti.dds.subscription.Subscriber** (p. 1730) the **com.rti.dds.subscription.DataReader** (p. 450) entity name.

Every **com.rti.dds.subscription.DataReader** (p. 450) in the system has an entity name which is configured and stored in the <<***extension***>> *(p. 155)* EntityName policy, **ENTITY_NAME** (p. 234).

This operation retrieves the **com.rti.dds.subscription.DataReader** (p. 450) within the **com.rti.dds.subscription.Subscriber** (p. 1730) whose name matches the one specified. If there are several **com.rti.dds.subscription.DataReader** (p. 450) with the same name within the **com.rti.dds.subscription.Subscriber** (p. 1730), the operation returns the first matching occurrence.

Parameters

datareader_name	<<***in***>> *(p. 156)* Entity name of the **com.rti.dds.subscription.DataReader** (p. 450).

Returns

The first **com.rti.dds.subscription.DataReader** (p. 450) found with the specified name or NULL if it is not found.

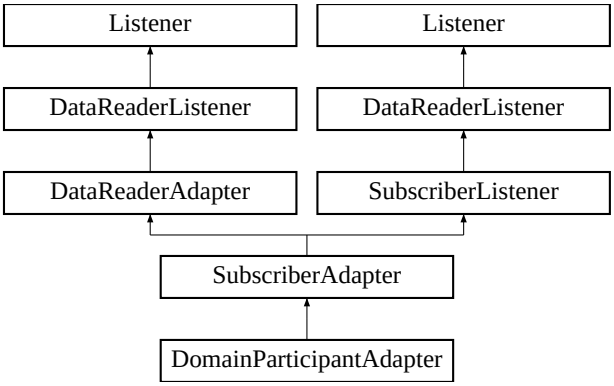
See also

com.rti.dds.domain.DomainParticipant.lookup_datareader_by_name (p. 747)

8.327 SubscriberAdapter Class Reference

A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Inheritance diagram for SubscriberAdapter:



Public Member Functions

- void **on_data_on_readers** (**Subscriber** subs)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS communication status.

8.327.1 Detailed Description

A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.327.2 Member Function Documentation

8.327.2.1 on_data_on_readers()

```
void on_data_on_readers (
              Subscriber subs )
```

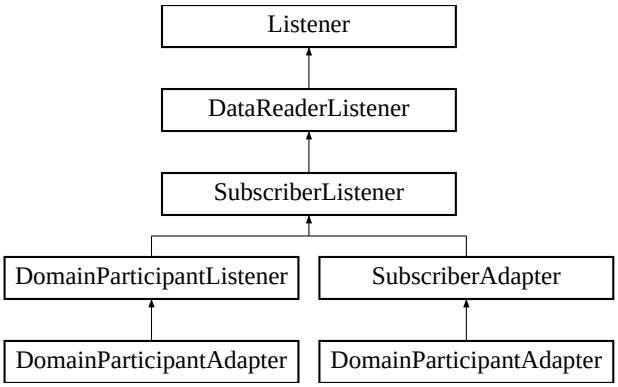
Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS communication status.

Implements **SubscriberListener** (p. 1756).

8.328 SubscriberListener Interface Reference

<<***interface***>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for status about a subscriber.

Inheritance diagram for SubscriberListener:



Public Member Functions

- void **on_data_on_readers** (**Subscriber** subs)

  Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS communication status.

8.328.1 Detailed Description

<<***interface***>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for status about a subscriber.

Entity:

com.rti.dds.subscription.Subscriber (p. 1730)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS;
com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS;
com.rti.dds.infrastructure.StatusKind.StatusKind.LIVELINESS_CHANGED_STATUS, **com.rti.dds.subscription.↩ LivelinessChangedStatus** (p. 1239);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_DEADLINE_MISSED_STATUS, **com.rti.dds.↩ subscription.RequestedDeadlineMissedStatus** (p. 1560);
com.rti.dds.infrastructure.StatusKind.StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.↩ subscription.RequestedIncompatibleQosStatus** (p. 1561);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_LOST_STATUS_STATUS, **com.rti.dds.subscription.↩ SampleLostStatus** (p. 1648);
com.rti.dds.infrastructure.StatusKind.StatusKind.SAMPLE_REJECTED_STATUS, **com.rti.dds.subscription.↩ SampleRejectedStatus** (p. 1657);
com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS, **com.rti.dds.subscription.↩ SubscriptionMatchedStatus** (p. 1773);

See also

com.rti.dds.infrastructure.Listener (p. 1236)

Status Kinds (p. 262)

Operations Allowed in Listener Callbacks (p. 1238)

8.328.2 Member Function Documentation

8.328.2.1 on_data_on_readers()

```
void on_data_on_readers (
              Subscriber subs )
```

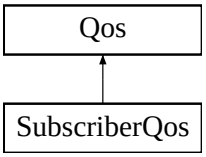
Handles the com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS communication status.

Implemented in **SubscriberAdapter** (p. 1754).

8.329 SubscriberQos Class Reference

QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1730) entity.

Inheritance diagram for SubscriberQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**SubscriberQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **SubscriberQos** (p. 1756) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **SubscriberQos** (p. 1756) object.*
- String **toString** (**SubscriberQos** baseQos)

  *Obtains a string representation of a **DataReaderQos** (p. 517) object.*

Public Attributes

- final **PresentationQosPolicy presentation**

  *Presentation policy, **PRESENTATION** (p. 247).*
- final **PartitionQosPolicy partition**

  *Partition policy, **PARTITION** (p. 246).*
- final **GroupDataQosPolicy group_data**

  *Group data policy, **GROUP_DATA** (p. 236).*
- final **EntityFactoryQosPolicy entity_factory**

  *Entity factory policy, **ENTITY_FACTORY** (p. 234).*
- final **EntityNameQosPolicy subscriber_name**

  <<***extension***>> *(p. 155) EntityName policy, **ENTITY_NAME** (p. 234).*

8.329.1 Detailed Description

QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1730) entity.

You must set certain members in a consistent manner:

length of **com.rti.dds.infrastructure.GroupDataQosPolicy.value** (p. 1128) <= **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.subscriber_group_data_max_length** (p. 815)

length of **com.rti.dds.infrastructure.PartitionQosPolicy.name** (p. 1367) <= **com.rti.dds.infrastructure.Domain↩ ParticipantResourceLimitsQosPolicy.max_partitions** (p. 816)

combined number of characters (including terminating 0) in **com.rti.dds.infrastructure.PartitionQosPolicy.↩ name** (p. 1367) <= **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.max_partition_↩ cumulative_characters** (p. 816)

If any of the above are not true, **com.rti.dds.subscription.Subscriber.set_qos** (p. 1744) and **com.rti.dds.↩ subscription.Subscriber.set_qos_with_profile** (p. 1744) will fail with **com.rti.dds.infrastructure.RETCODE_↩ INCONSISTENT_POLICY** (p. 1596)

8.329.2 Member Function Documentation

8.329.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 1757) overloads allow formatting the output and printing only the differences with respect to another **DataReaderQos** (p. 517) object.

```
SubscriberQos qos = new SubscriberQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
SubscriberQos baseQos = new SubscriberQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value SUBSCRIBER_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(SUBSCRIBER_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **DataReaderQos** (p. 517) and the documented default.

Returns

The string representation of the Qos.

References **SubscriberQos.toString()**.

Referenced by **SubscriberQos.toString()**.

8.329.2.2 toString() [2/4]

```
String toString (
            SubscriberQos baseQos,
            QosPrintFormat format )
```

Obtains a string representation of a **SubscriberQos** (p. 1756) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_PRINT_ALL** (p. 47) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipant.SUBSCRIBER_QOS_PRINT_ALL**.

8.329.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **SubscriberQos** (p. 1756) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **SubscriberQos.toString()**.

8.329.2.4 toString() [4/4]

```
String toString (
              SubscriberQos baseQos )
```

Obtains a string representation of a **DataReaderQos** (p. 517) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.SUBSCRIBER_QOS_PRINT_ALL** (p. 47) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **SubscriberQos.toString()**.

8.329.3 Member Data Documentation

8.329.3.1 presentation

```
final PresentationQosPolicy presentation
```

Presentation policy, **PRESENTATION** (p. 247).

8.329.3.2 partition

```
final PartitionQosPolicy partition
```

Partition policy, **PARTITION** (p. 246).

8.329.3.3 group_data

```
final GroupDataQosPolicy group_data
```

Group data policy, **GROUP_DATA** (p. 236).

8.329.3.4 entity_factory

```
final EntityFactoryQosPolicy entity_factory
```

Entity factory policy, **ENTITY_FACTORY** (p. 234).

8.329.3.5 subscriber_name

```
final  EntityNameQosPolicy subscriber_name
```

<<***extension***>> *(*p. *155)* EntityName policy, **ENTITY_NAME** (p. 234).

8.330 SubscriberSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1730) > .

Inherits AbstractNativeSequence.

Public Member Functions

- **SubscriberSeq** (Collection<?> subscribers)
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.330.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1730) > .

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.330.2 Constructor & Destructor Documentation

8.330.2.1 SubscriberSeq()

```
SubscriberSeq (
            Collection<?> subscribers )
```

Exceptions

NullPointerException	if the given collection is null

8.330.3 Member Function Documentation

8.330.3.1 getMaximum()

```
int getMaximum ( )
```

Get the current maximum number of elements that can be stored in this sequence.

The `maximum` of the sequence represents the maximum number of elements that the underlying buffer can hold. It does not represent the current number of elements.

The `maximum` is a non-negative number. It is initialized when the sequence is first created.

The `maximum` can be changed implicitly by adding an element to the sequence with **`add()`** (p. 329), or explicitly by calling com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.setMaximum.

Returns

the current maximum of the sequence.

See also

com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size()

Implements **Sequence** (p. 1665).

8.331 SubscriptionBuiltinTopicData Class Reference

Entry created when a **com.rti.dds.subscription.DataReader** (p. 450) is discovered in association with its **Subscriber** (p. 1730).

Inherits AbstractBuiltinTopicData.

Public Attributes

- final **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.
- final **BuiltinTopicKey_t participant_key**

  *DCPS key of the participant to which the **DataReader** (p. 450) belongs.*
- String **topic_name**

  *Name of the related **com.rti.dds.topic.Topic** (p. 1807).*
- String **type_name**

  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).*
- final **DurabilityQosPolicy durability**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **OwnershipQosPolicy ownership**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **DestinationOrderQosPolicy destination_order**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **UserDataQosPolicy user_data**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **TimeBasedFilterQosPolicy time_based_filter**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **PresentationQosPolicy presentation**

  *Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.*
- final **PartitionQosPolicy partition**

  *Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.*
- final **TypeConsistencyEnforcementQosPolicy type_consistency**

  *Policy of the corresponding **DataReader** (p. 450).*
- final **TopicDataQosPolicy topic_data**

  Policy of the related Topic.
- final **GroupDataQosPolicy group_data**

  *Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.*
- **TypeCode type_code**

  *<<**extension**>> (p. 155) Type code information of the corresponding Topic*
- final **BuiltinTopicKey_t subscriber_key**

  *<<**extension**>> (p. 155) DCPS key of the subscriber to which the **DataReader** (p. 450) belongs.*
- final **PropertyQosPolicy property**

  *<<**extension**>> (p. 155) Properties of the corresponding **DataReader** (p. 450).*
- final **LocatorSeq unicast_locators**

  *<<**extension**>> (p. 155) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*

- final **LocatorSeq multicast_locators**

  <<***extension***>> (p. *155) Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.*
- final **ContentFilterProperty_t content_filter_property**

  <<***extension***>> (p. *155) This field provides all the required information to enable content filtering on the Writer side.*
- final **GUID_t virtual_guid**

  <<***extension***>> (p. *155) Virtual GUID associated to the* ***DataReader*** (p. *450).*
- final **ServiceQosPolicy service**

  <<***extension***>> (p. *155) Policy of the corresponding* ***DataReader*** (p. *450).*
- final **ProtocolVersion_t rtps_protocol_version**

  <<***extension***>> (p. *155) Version number of the RTPS wire protocol used.*
- final **VendorId_t rtps_vendor_id**

  <<***extension***>> (p. *155) ID of vendor implementing the RTPS wire protocol.*
- final **ProductVersion_t product_version**

  <<***extension***>> (p. *155) This is a vendor specific parameter. It gives the current version of RTI Connext*
- final **DataRepresentationQosPolicy representation**

  Data representation policy of the corresponding ***DataReader*** (p. *450).*
- boolean **disable_positive_acks**

  <<***extension***>> (p. *155) This is a vendor specific parameter. Determines whether the corresponding* ***DataReader*** *(p. 450) sends positive acknowledgments for reliability.*
- final **EntityNameQosPolicy subscription_name**

  <<***extension***>> (p. *155) The subscription name and role name.*
- final **EndpointTrustProtectionInfo trust_protection_info**

  <<***extension***>> (p. *155) Trust plugins protection information associated with the discovered* ***DataReader*** (p. *450).*
- final **EndpointTrustAlgorithmInfo trust_algorithm_info**

  <<***extension***>> (p. *155) Trust Plugins algorithms associated with the discovered* ***DataReader*** (p. *450).*
- final **DataTagQosPolicy data_tags**

  Tags of the corresponding ***DataReader*** (p. *450).*

8.331.1 Detailed Description

Entry created when a **com.rti.dds.subscription.DataReader** (p. 450) is discovered in association with its **Subscriber** (p. 1730).

Data associated with the built-in topic **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicDataType↩ Support.SUBSCRIPTION_TOPIC_NAME** (p. 164). It contains QoS policies and additional information that apply to the remote **com.rti.dds.subscription.DataReader** (p. 450) the related **com.rti.dds.subscription.Subscriber** (p. 1730).

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_↩ NAME (p. 164)

builtin.SubscriptionBuiltinTopicDataDataReader (p. 1771)

8.331.2 Member Data Documentation

8.331.2.1 key

```
final  BuiltinTopicKey_t key
```

DCPS key to distinguish entries.

8.331.2.2 participant_key

```
final  BuiltinTopicKey_t participant_key
```

DCPS key of the participant to which the **DataReader** (p. 450) belongs.

8.331.2.3 topic_name

```
String topic_name
```

Name of the related **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.331.2.4 type_name

```
String type_name
```

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.331.2.5 durability

```
final  DurabilityQosPolicy durability
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.6 deadline

```
final  DeadlineQosPolicy deadline
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.7 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.8 liveliness

```
final LivelinessQosPolicy liveliness
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.9 reliability

```
final ReliabilityQosPolicy reliability
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.10 ownership

```
final OwnershipQosPolicy ownership
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.11 destination_order

```
final DestinationOrderQosPolicy destination_order
```

Policy of the corresponding **DataReader** (p. 450).

Warning

Only the field **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 637) is propagated during discovery. The other fields always contain their default values.

8.331.2.12 user_data

```
final  UserDataQosPolicy user_data
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.13 time_based_filter

```
final  TimeBasedFilterQosPolicy time_based_filter
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.14 presentation

```
final  PresentationQosPolicy presentation
```

Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.

8.331.2.15 partition

```
final  PartitionQosPolicy partition
```

Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.

8.331.2.16 type_consistency

```
final  TypeConsistencyEnforcementQosPolicy type_consistency
```

Policy of the corresponding **DataReader** (p. 450).

8.331.2.17 topic_data

```
final  TopicDataQosPolicy topic_data
```

Policy of the related Topic.

8.331.2.18 group_data

```
final  GroupDataQosPolicy group_data
```

Policy of the **Subscriber** (p. 1730) to which the **DataReader** (p. 450) belongs.

8.331.2.19 type_code

```
 TypeCode type_code
```

<<***extension***>> *(p. 155)* Type code information of the corresponding Topic

8.331.2.20 subscriber_key

```
final  BuiltinTopicKey_t subscriber_key
```

<<***extension***>> *(p. 155)* DCPS key of the subscriber to which the **DataReader** (p. 450) belongs.

8.331.2.21 property

```
final  PropertyQosPolicy property
```

<<***extension***>> *(p. 155)* Properties of the corresponding **DataReader** (p. 450).

8.331.2.22 unicast_locators

```
final  LocatorSeq unicast_locators
```

<<***extension***>> *(p. 155)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.331.2.23 multicast_locators

```
final  LocatorSeq multicast_locators
```

<<***extension***>> *(p. 155)* Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.331.2.24 content_filter_property

```
final  ContentFilterProperty_t content_filter_property
```

<<***extension***>> *(p. 155)* This field provides all the required information to enable content filtering on the Writer side.

8.331.2.25 virtual_guid

```
final  GUID_t virtual_guid
```

<<***extension***>> *(p. 155)* Virtual GUID associated to the **DataReader** (p. 450).

See also

com.rti.dds.infrastructure.GUID_t (p. 1132)

8.331.2.26 service

```
final  ServiceQosPolicy service
```

<<***extension***>> *(p. 155)* Policy of the corresponding **DataReader** (p. 450).

8.331.2.27 rtps_protocol_version

```
final  ProtocolVersion_t rtps_protocol_version
```

<<***extension***>> *(p. 155)* Version number of the RTPS wire protocol used.

8.331.2.28 rtps_vendor_id

```
final VendorId_t rtps_vendor_id
```

<<***extension***>> *(*p. *155)* ID of vendor implementing the RTPS wire protocol.

8.331.2.29 product_version

```
final ProductVersion_t product_version
```

<<***extension***>> *(*p. *155)* This is a vendor specific parameter. It gives the current version of RTI Connext

8.331.2.30 representation

```
final DataRepresentationQosPolicy representation
```

Data representation policy of the corresponding **DataReader** (p. 450).

8.331.2.31 disable_positive_acks

```
boolean disable_positive_acks
```

<<***extension***>> *(*p. *155)* This is a vendor specific parameter. Determines whether the corresponding **DataReader** (p. 450) sends positive acknowledgments for reliability.

8.331.2.32 subscription_name

```
final EntityNameQosPolicy subscription_name
```

<<***extension***>> *(*p. *155)* The subscription name and role name.

This member contains the name and the role name of the discovered subscription.

8.331.2.33 trust_protection_info

```
final EndpointTrustProtectionInfo trust_protection_info
```

<<***extension***>> *(*p. *155)* Trust plugins protection information associated with the discovered **DataReader** (p. 450).

e

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_protection_info contains information about how RTPS wire serialization, discovery, and liveliness interact with the loaded Trust Plugins. Two endpoints will not match if their trust_protection_info is incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.331.2.34 trust_algorithm_info

```
final EndpointTrustAlgorithmInfo trust_algorithm_info
```

<<***extension***>> *(*p. *155)* Trust Plugins algorithms associated with the discovered **DataReader** (p. 450).

e

Trust Plugins is a generic abstraction that represents any plugin intended to do transformation, interception, and validation of exchanged data and metadata.

trust_algorithm_info contains information about what algorithms the loaded Trust Plugins are running. Two endpoints will not match if their trust_algorithm_info are incompatible.

The meaning of the contents of this field may vary depending on what Trust Plugins the endpoint is using. For information about how this field interacts with the RTI Security Plugins, please refer to the `RTI Security Plugins User's Manual`.

8.331.2.35 data_tags

```
final DataTagQosPolicy data_tags
```

Tags of the corresponding **DataReader** (p. 450).

8.332 SubscriptionBuiltinTopicDataDataReader Class Reference

Instantiates **DataReader** (p. 450) < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) >.

Inherits DataReaderImpl.

8.332.1 Detailed Description

Instantiates **DataReader** (p. 450) < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) >.

com.rti.dds.subscription.DataReader (p. 450) of topic **com.rti.dds.subscription.builtin.SubscriptionBuiltin↩ TopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME** (p. 164) used for accessing **com.rti.dds.subscription.↩ builtin.SubscriptionBuiltinTopicData** (p. 1762) of the remote **com.rti.dds.subscription.DataReader** (p. 450) and the associated **com.rti.dds.subscription.Subscriber** (p. 1730).

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooDataReader** (p. 1067)

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_↩ NAME (p. 164)

8.333 SubscriptionBuiltinTopicDataSeq Class Reference

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.subscription.↩ builtin.SubscriptionBuiltinTopicData** (p. 1762) >.

Inherits AbstractBuiltinTopicDataSeq.

8.333.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.subscription.↩ builtin.SubscriptionBuiltinTopicData** (p. 1762) >.

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

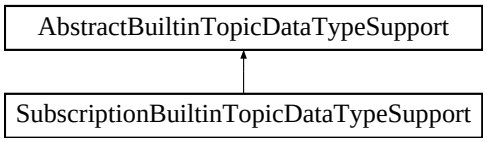
See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

8.334 SubscriptionBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) > .

Inheritance diagram for SubscriptionBuiltinTopicDataTypeSupport:



Static Public Attributes

- static final String **SUBSCRIPTION_TOPIC_NAME** = DDS_SUBSCRIPTION_TOPIC_NAME()
  Subscription topic name.

Additional Inherited Members

8.334.1 Detailed Description

Instantiates `TypeSupport` < **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762) > .

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooTypeSupport** (p. 1118)

See also

com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData (p. 1762)

8.335 SubscriptionMatchedStatus Class Reference

com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS

Inherits Status.

Public Attributes

- int **total_count** = 0

  *The total cumulative number of times that this **com.rti.dds.subscription.DataReader** (p. 450) discovered a "match" with a **com.rti.dds.publication.DataWriter** (p. 553).*
- int **total_count_change** = 0

  The changes in total_count since the last time the listener was called or the status was read.
- int **current_count** = 0

  *The current number of DataWriters with which the **com.rti.dds.subscription.DataReader** (p. 450) is matched.*
- int **current_count_peak** = 0

  *<<**extension**>> (p. 155) Greatest number of DataWriters that matched this **DataReader** (p. 450) simultaneously.*
- int **current_count_change** = 0

  The change in current_count since the last time the listener was called or the status was read.
- final **InstanceHandle_t** **last_publication_handle** = new **InstanceHandle_t**()

  *This InstanceHandle can be used to look up which remote **com.rti.dds.publication.DataWriter** (p. 553) was the last to cause this **DataReader** (p. 450)'s status to change, using **com.rti.dds.subscription.DataReader.get_matched_↩ publication_data** (p. 466).*

8.335.1 Detailed Description

com.rti.dds.infrastructure.StatusKind.StatusKind.SUBSCRIPTION_MATCHED_STATUS

A "match" happens when the **com.rti.dds.subscription.DataReader** (p. 450) finds a **com.rti.dds.publication.Data↩ Writer** (p. 553) with the same **com.rti.dds.topic.Topic** (p. 1807), same or compatible data type, and an offered QoS that is compatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 450). (For information on compatible data types, see the `Extensible Types Guide`.)

This status is also changed (and the listener, if any, called) when a match is ended. A local **com.rti.dds.subscription.↩ DataReader** (p. 450) will become "unmatched" from a remote **com.rti.dds.publication.DataWriter** (p. 553) when that **com.rti.dds.publication.DataWriter** (p. 553) goes away for any of the following reasons:

- The **com.rti.dds.domain.DomainParticipant** (p. 670) containing the matched **com.rti.dds.publication.Data↩ Writer** (p. 553) has lost liveliness.
- This **DataReader** (p. 450) or the matched DataWriter has changed QoS such that the entities are now incompatible.
- The matched DataWriter has been deleted.

This status may reflect changes from multiple match or unmatch events, and the **com.rti.dds.subscription.↩ SubscriptionMatchedStatus.current_count_change** (p. 1775) can be used to determine the number of changes since the listener was called back or the status was checked.

Note: A DataWriter's loss of liveliness (which is determined by **com.rti.dds.infrastructure.LivelinessQosPolicyKind** (p. 1247)) does not trigger an unmatch event. So a DataWriter may still match even though its liveliness is lost.

8.335.2 Member Data Documentation

8.335.2.1 total_count

```
int total_count = 0
```

The total cumulative number of times that this **com.rti.dds.subscription.DataReader** (p. 450) discovered a "match" with a **com.rti.dds.publication.DataWriter** (p. 553).

This number increases whenever a new match is discovered. It does not decrease when an existing match goes away for any of the reasons described in **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773).

8.335.2.2 total_count_change

```
int total_count_change = 0
```

The changes in total_count since the last time the listener was called or the status was read.

Note that this number will never be negative (because it's the total number of times the **DataReader** (p. 450) ever matched with a DataWriter).

8.335.2.3 current_count

```
int current_count = 0
```

The current number of DataWriters with which the **com.rti.dds.subscription.DataReader** (p. 450) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away for any of the reasons described in **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773).

8.335.2.4 current_count_peak

```
int current_count_peak = 0
```

<<***extension***>> *(p. 155)* Greatest number of DataWriters that matched this **DataReader** (p. 450) simultaneously.

That is, there was no moment in time when more than this many DataWriters matched this **DataReader** (p. 450). (As a result, total_count can be higher than current_count_peak.)

8.335.2.5 current_count_change

```
int current_count_change = 0
```

The change in current_count since the last time the listener was called or the status was read.

Note that a negative current_count_change means that one or more DataWriters have become unmatched for one or more of the reasons described in **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773).

8.335.2.6 last_publication_handle

```
final InstanceHandle_t last_publication_handle = new InstanceHandle_t()
```

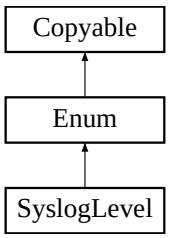
This InstanceHandle can be used to look up which remote **com.rti.dds.publication.DataWriter** (p. 553) was the last to cause this **DataReader** (p. 450)'s status to change, using **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466).

If the DataWriter no longer matches this **DataReader** (p. 450) due to any of the reasons in **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1773) except incompatible QoS, then the DataWriter has been purged from this **DataReader** (p. 450)'s DomainParticipant discovery database. (See the "Discovery Overview" section of the User's Manual.) In that case, the **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 466) method will not be able to return information about the DataWriter. The only way to get information about the lost DataWriter is if you cached the information previously.

8.336 SyslogLevel Class Reference

Syslog level category assigned to RTI Connext log messages. See Syslog Level and Verbosity Mapping, in the Core Libraries User's Manual, for more information.

Inheritance diagram for SyslogLevel:



Static Public Attributes

- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY**
  System is unusable.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_ALERT**
  Should be corrected immediately.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL**
  Critical conditions.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_ERROR**
  Error conditions.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_WARNING**
  May indicate that an error will occur if action is not taken.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_NOTICE**
  Events that are unusual, but not error conditions.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL**
  Normal operational messages that require no action.
- static final **SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_DEBUG**
  Information useful to developers for debugging the application.

Additional Inherited Members

8.336.1 Detailed Description

Syslog level category assigned to RTI Connext log messages. See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

8.336.2 Member Data Documentation

8.336.2.1 NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY

`final` **SyslogLevel** `NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY [static]`

System is unusable.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1279).

8.336.2.2 NDDS_CONFIG_SYSLOG_LEVEL_ALERT

`final` **SyslogLevel** `NDDS_CONFIG_SYSLOG_LEVEL_ALERT [static]`

Should be corrected immediately.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1279).

8.336.2.3 NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL

`final` **SyslogLevel** `NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL [static]`

Critical conditions.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1279).

8.336.2.4 NDDS_CONFIG_SYSLOG_LEVEL_ERROR

`final` **SyslogLevel** `NDDS_CONFIG_SYSLOG_LEVEL_ERROR [static]`

Error conditions.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1279).

8.336.2.5 NDDS_CONFIG_SYSLOG_LEVEL_WARNING

```
final SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_WARNING [static]
```

May indicate that an error will occur if action is not taken.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_WARNING** (p. 1279).

8.336.2.6 NDDS_CONFIG_SYSLOG_LEVEL_NOTICE

```
final SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_NOTICE [static]
```

Events that are unusual, but not error conditions.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL** (p. 1280).

8.336.2.7 NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL

```
final SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL [static]
```

Normal operational messages that require no action.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL** (p. 1280).

8.336.2.8 NDDS_CONFIG_SYSLOG_LEVEL_DEBUG

```
final SyslogLevel NDDS_CONFIG_SYSLOG_LEVEL_DEBUG [static]
```

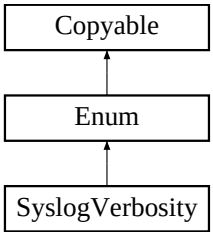
Information useful to developers for debugging the application.

Maps to **com.rti.ndds.config.LogLevel.NDDS_CONFIG_LOG_LEVEL_DEBUG** (p. 1280).

8.337 SyslogVerbosity Class Reference

The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **com.rti.ndds.config.LogVerbosity** (p. 1285). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

Inheritance diagram for SyslogVerbosity:



Static Public Attributes

- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT**

  No messages will be logged. (lowest verbosity)
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY**

  Emergency, critical, alert, and error messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT**

  Emergency, critical, and alert messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL**

  Emergency, critical, alert, and error messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR**

  Emergency, critical, alert, and error messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING**

  Emergency, critical, alert, error, and warning messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE**

  Emergency, critical, alert, error, warning, notice and informational messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL**

  Emergency, critical, alert, error, warning, notice, and informational messages will be logged.
- static final **SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG**

  All messages will be logged.

Additional Inherited Members

8.337.1 Detailed Description

The Syslog verbosities at which RTI Connext diagnostic information is logged. These Syslog verbosities are mapped to **com.rti.ndds.config.LogVerbosity** (p. 1285). See `Syslog Level and Verbosity Mapping`, in the Core Libraries User's Manual, for more information.

8.337.2 Member Data Documentation

8.337.2.1 NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT [static]
```

No messages will be logged. (lowest verbosity)

Equivalent to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_SILENT** (p. 1286).

8.337.2.2 NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY [static]
```

Emergency, critical, alert, and error messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _ERROR** (p. 1286) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency messages.)

8.337.2.3 NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT [static]
```

Emergency, critical, and alert messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _ERROR** (p. 1286) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical and alert messages.)

8.337.2.4 NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL [static]
```

Emergency, critical, alert, and error messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _ERROR** (p. 1286) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency and critical messages.)

8.337.2.5 NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR [static]
```

Emergency, critical, alert, and error messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _ERROR** (p. 1286) when interpreted by RTI Connext.

8.337.2.6 NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING

```
final SyslogVerbosity NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING [static]
```

Emergency, critical, alert, error, and warning messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _WARNING** (p. 1286) when interpreted by RTI Connext.

8.337.2.7 NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE

final **SyslogVerbosity** NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE [static]

Emergency, critical, alert, error, warning, notice and informational messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _STATUS_REMOTE** (p. 1286) when interpreted by RTI Connext. (That's why this level actually logs more than just emergency, critical, alert, error, warning and notice messages.)

8.337.2.8 NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL

final **SyslogVerbosity** NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL [static]

Emergency, critical, alert, error, warning, notice, and informational messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _STATUS_REMOTE** (p. 1286) when interpreted by RTI Connext.

8.337.2.9 NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG

final **SyslogVerbosity** NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG [static]

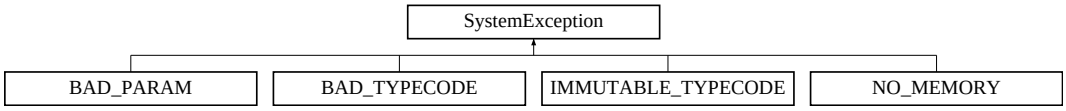
All messages will be logged.

This Syslog verbosity level is translated to **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY↩ _STATUS_ALL** (p. 1286) when interpreted by RTI Connext.

8.338 SystemException Class Reference

System exception.

Inheritance diagram for SystemException:



8.338.1 Detailed Description

System exception.

This class is based on a similar class in CORBA.

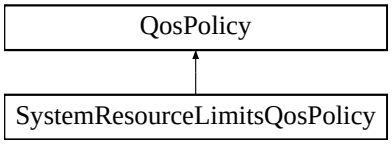
See also

http://java.sun.com/javase/6/docs/api/org/omg/CORBA/SystemException.html

8.339 SystemResourceLimitsQosPolicy Class Reference

*<<**extension**>> (p. 155)* Configures **com.rti.dds.domain.DomainParticipant** (p. 670)-independent resources used by RTI Connext. Mainly used to change the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 670) entities that can be created within a single process (address space).

Inheritance diagram for SystemResourceLimitsQosPolicy:



Public Attributes

- int **max_objects_per_thread**

  *The maximum number of objects that can be stored per thread for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).*
- int **initial_objects_per_thread**

  *The number of objects per thread for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761) for which infrastructure will initially be allocated.*

8.339.1 Detailed Description

*<<**extension**>> (p. 155)* Configures **com.rti.dds.domain.DomainParticipant** (p. 670)-independent resources used by RTI Connext. Mainly used to change the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 670) entities that can be created within a single process (address space).

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 761)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.339.2 Usage

Within a single process (or address space for some supported real-time operating systems), applications may create and use multiple **com.rti.dds.domain.DomainParticipant** (p. 670) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 670) entities that can be created in a single process/address space. These values cannot be changed after a DomainParticipant has been created and they cannot be set in a QoS XML file.

8.339.3 Member Data Documentation

8.339.3.1 max_objects_per_thread

```
int max_objects_per_thread
```

The maximum number of objects that can be stored per thread for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761).

This value is the upper bound on the number of objects that can be stored per thread. When a **com.rti.dds.domain.↩ DomainParticipantFactory** (p. 761) is created, infrastructure will be created to manage the number of objects specified by initial_objects_per_thread. As more objects are required by the application, the infrastructure will be automatically grown to accommodate up to max_objects_per_thread objects. Leave this property set to the default value to allow the infrastructure to grow as needed. If you wish to strictly control memory allocation, set max_objects_per_thread to a smaller value, but note that this runs the risk of a runtime error and reduced application functionality if your limit is reached.

[default] 261120

[range] [1, 261120]

8.339.3.2 initial_objects_per_thread

```
int initial_objects_per_thread
```

The number of objects per thread for a **com.rti.dds.domain.DomainParticipantFactory** (p. 761) for which infrastructure will initially be allocated.

The infrastructure for managing thread-specific objects will initially be sized according to this value. The infrastructure will grow automatically, up to a maximum of max_objects_per_thread, as required by the application at runtime. If you are certain that more than the default value of initial_objects_per_thread will be required for your application and you wish to reduce the number of memory allocations performed while your application reaches steady state, you may set this value to a larger number. To improve the efficiency of memory allocation, RTI Connext may initially size the infrastructure to a larger value than the value of initial_objects_per_thread. The infrastructure will never be sized less than initial_objects_per_thread or greater than max_objects_per_thread. When the infrastructure for managing thread-specific objects is created or increased, a log message stating "Allowed number of thread specific objects is now " will be produced at the local log level.

[default] 1024

[range] [1, 261120]; must be less than or equal to max_objects_per_thread

8.340 Tag Class Reference

Tags are name/value pair objects.

Inherits Struct.

Public Member Functions

- **Tag** ()
  Constructor.
- **Tag** (**Tag** src)
  Constructor.
- **Tag** (String **name**, String **value**)
  Constructor.

Public Attributes

- String **name**
  Tag *(*p. *1783) name.*
- String **value**
  Tag *(*p. *1783) value.*

8.340.1 Detailed Description

Tags are name/value pair objects.

8.340.2 Constructor & Destructor Documentation

8.340.2.1 Tag() [1/3]

```
Tag ( )
```

Constructor.

8.340.2.2 Tag() [2/3]

```
Tag (
            Tag src )
```

Constructor.

Parameters

src	<<***in***>> *(*p. *156)* **Tag** (p. 1783) used to initialize the new tag.

References **Tag.name**, and **Tag.value**.

8.340.2.3 Tag() [3/3]

```
Tag (
            String name,
            String value )
```

Constructor.

Parameters

name	<<***in***>> *(*p. *156)* **Tag** (p. 1783) name.
value	<<***in***>> *(*p. *156)* **Tag** (p. 1783) value.

References **Tag.name**, and **Tag.value**.

8.340.3 Member Data Documentation

8.340.3.1 name

```
String name
```

Tag (p. 1783) name.

Referenced by **DataTagQosPolicyHelper.lookup_tag()**, and **Tag.Tag()**.

8.340.3.2 value

```
String value
```

Tag (p. 1783) value.

Referenced by **DataTagQosPolicyHelper.assert_tag()**, and **Tag.Tag()**.

8.341 TagSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.Tag** (p. 1783) >

Inherits ArraySequence.

8.341.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.Tag** (p. 1783) >

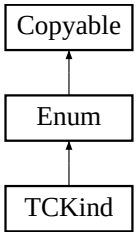
See also

com.rti.dds.infrastructure.Tag (p. 1783)

8.342 TCKind Class Reference

Enumeration type for **com.rti.dds.typecode.TypeCode** (p. 1873) kinds.

Inheritance diagram for TCKind:



Static Public Attributes

- static final **TCKind TK_NULL**
  Indicates that a type code does not describe anything.
- static final **TCKind TK_SHORT**
  short type.
- static final **TCKind TK_LONG**
  long type.
- static final **TCKind TK_USHORT**
  unsigned short type.
- static final **TCKind TK_ULONG**
  unsigned long type.
- static final **TCKind TK_FLOAT**
  float type.
- static final **TCKind TK_DOUBLE**
  double type.
- static final **TCKind TK_BOOLEAN**
  boolean type.
- static final **TCKind TK_CHAR**
  char type.
- static final **TCKind TK_OCTET**
  octet type.
- static final **TCKind TK_STRUCT**

struct type.

- static final **TCKind TK_UNION**

  union type.
- static final **TCKind TK_ENUM**

  enumerated type.
- static final **TCKind TK_STRING**

  string type.
- static final **TCKind TK_SEQUENCE**

  sequence type.
- static final **TCKind TK_ARRAY**

  array type.
- static final **TCKind TK_ALIAS**

  alias (typedef) type.
- static final **TCKind TK_LONGLONG**

  long long type.
- static final **TCKind TK_ULONGLONG**

  unsigned long long type.
- static final **TCKind TK_LONGDOUBLE**

  long double type.
- static final **TCKind TK_WCHAR**

  wide char type.
- static final **TCKind TK_WSTRING**

  wide string type.
- static final **TCKind TK_VALUE**

  value type.

Additional Inherited Members

8.342.1 Detailed Description

Enumeration type for **com.rti.dds.typecode.TypeCode** (p. 1873) kinds.

Type code kinds are modeled as values of this type.

8.342.2 Member Data Documentation

8.342.2.1 TK_NULL

```
final TCKind TK_NULL [static]
```

Indicates that a type code does not describe anything.

Referenced by **TypeCode.concrete_base_type()**, **TypeCodeFactory.create_value_tc()**, **DynamicDataMember↩ Info.DynamicDataMemberInfo()**, and **TypeCodeFactory.get_primitive_tc()**.

8.342.2.2 TK_SHORT

```
final TCKind TK_SHORT [static]
```

short type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.3 TK_LONG

```
final TCKind TK_LONG [static]
```

long type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.4 TK_USHORT

```
final TCKind TK_USHORT [static]
```

unsigned short type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.5 TK_ULONG

```
final TCKind TK_ULONG [static]
```

unsigned long type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.6 TK_FLOAT

```
final TCKind TK_FLOAT [static]
```

float type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.7 TK_DOUBLE

```
final TCKind TK_DOUBLE [static]
```

double type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.8 TK_BOOLEAN

```
final TCKind TK_BOOLEAN [static]
```

boolean type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.9 TK_CHAR

```
final TCKind TK_CHAR [static]
```

char type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.10 TK_OCTET

```
final TCKind TK_OCTET [static]
```

octet type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.11 TK_STRUCT

```
final TCKind TK_STRUCT [static]
```

struct type.

Referenced by **TypeCode.concrete_base_type()**, **TypeCode.equal()**, **TypeCode.find_member_by_name()**, **TypeCode.is_member_bitfield()**, **TypeCode.is_member_key()**, **TypeCode.is_member_pointer()**, **TypeCode.is↩ _member_required()**, **TypeCode.member_bitfield_bits()**, **TypeCode.member_count()**, **TypeCode.member_id()**, **TypeCode.member_name()**, **TypeCode.member_type()**, **TypeCode.member_visibility()**, and **TypeCode.type_↩ modifier()**.

8.342.2.12 TK_UNION

```
final TCKind TK_UNION [static]
```

union type.

Referenced by **TypeCode.default_index()**, **TypeCode.discriminator_type()**, **TypeCode.equal()**, **TypeCode.find_member_by_name()**, **TypeCode.is_member_pointer()**, **TypeCode.is_member_required()**, **TypeCode.member_count()**, **TypeCode.member_id()**, **TypeCode.member_label()**, **TypeCode.member_label_count()**, **TypeCode.member_name()**, and **TypeCode.member_type()**.

8.342.2.13 TK_ENUM

```
final TCKind TK_ENUM [static]
```

enumerated type.

Referenced by **TypeCode.add_member_to_enum()**, **TypeCodeFactory.create_enum_tc()**, **TypeCode.equal()**, **TypeCode.find_member_by_name()**, **TypeCode.member_count()**, **TypeCode.member_name()**, and **TypeCode.member_ordinal()**.

8.342.2.14 TK_STRING

```
final TCKind TK_STRING [static]
```

string type.

Referenced by **TypeCodeFactory.create_string_tc()**, **TypeCode.equal()**, and **TypeCode.length()**.

8.342.2.15 TK_SEQUENCE

```
final TCKind TK_SEQUENCE [static]
```

sequence type.

Referenced by **TypeCode.content_type()**, **TypeCode.equal()**, **DynamicData.get_char_seq()**, **TypeCode.length()**, and **DynamicData.set_char_seq()**.

8.342.2.16 TK_ARRAY

```
final TCKind TK_ARRAY [static]
```

array type.

Referenced by **TypeCode.array_dimension()**, **TypeCode.array_dimension_count()**, **TypeCode.content_type()**, **TypeCode.element_count()**, **TypeCode.equal()**, **DynamicData.get_char_array()**, **TypeCode.length()**, and **DynamicData.set_char_array()**.

8.342.2.17 TK_ALIAS

```
final TCKind TK_ALIAS [static]
```

alias (typedef) type.

Referenced by **TypeCode.content_type()**, **TypeCode.equal()**, and **TypeCode.is_alias_pointer()**.

8.342.2.18 TK_LONGLONG

```
final TCKind TK_LONGLONG [static]
```

long long type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.19 TK_ULONGLONG

```
final TCKind TK_ULONGLONG [static]
```

unsigned long long type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.20 TK_LONGDOUBLE

```
final TCKind TK_LONGDOUBLE [static]
```

long double type.

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.342.2.21 TK_WCHAR

```
final TCKind TK_WCHAR [static]
```

wide char type.

Referenced by **DynamicData.get_char()**, **DynamicData.get_char_array()**, **DynamicData.get_char_seq()**, **TypeCodeFactory.get_primitive_tc()**, **DynamicData.set_char()**, **DynamicData.set_char_array()**, and **DynamicData.set_char_seq()**.

8.342.2.22 TK_WSTRING

```
final TCKind TK_WSTRING [static]
```

wide string type.

Referenced by **TypeCodeFactory.create_wstring_tc()**, **TypeCode.equal()**, **DynamicData.get_string()**, **TypeCode.length()**, and **DynamicData.set_string()**.

8.342.2.23 TK_VALUE

```
final TCKind TK_VALUE [static]
```

value type.

Referenced by **TypeCode.concrete_base_type()**, **TypeCode.equal()**, **TypeCode.find_member_by_name()**, **TypeCode.is_member_bitfield()**, **TypeCode.is_member_key()**, **TypeCode.is_member_pointer()**, **TypeCode.is_member_required()**, **TypeCode.member_bitfield_bits()**, **TypeCode.member_count()**, **TypeCode.member_id()**, **TypeCode.member_name()**, **TypeCode.member_type()**, **TypeCode.member_visibility()**, and **TypeCode.type_modifier()**.

8.343 ThreadSettings_t Class Reference

The properties of a thread of execution.

Inherits Struct.

Public Attributes

- int **mask**

  Describes the type of thread.
- int **priority**

  Thread priority.
- int **stack_size**

  The thread stack-size.
- final **IntSeq cpu_list** = new **IntSeq**()

  The list of processors on which the thread(s) may run.
- **ThreadSettingsCpuRotationKind cpu_rotation**

  Determines how processor affinity is applied to multiple threads.

8.343.1 Detailed Description

The properties of a thread of execution.

QoS:

com.rti.dds.infrastructure.EventQosPolicy (p. 1044) **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 445) **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523) **com.rti.dds.infrastructure.Asynchronous↩ PublisherQosPolicy** (p. 339)

8.343.2 Member Data Documentation

8.343.2.1 mask

```
int mask
```

Describes the type of thread.

The meaning of each bit of the mask are defined by **com.rti.dds.infrastructure.ThreadSettingsKind** (p. 1796).

[default] 0, use default options of the OS

8.343.2.2 priority

```
int priority
```

Thread priority.

Important: The interpretation of numeric thread priority values is different based on whether the priority is specified in your application code or in an XML QoS profile document. This is because QoS profile documents are intended to be usable across programming languages.

[range] java.lang.Thread.MIN_PRIORITY to MAX_PRIORITY if set programmatically in Java code **[range]** Platform-dependent - Consult `Platform Notes` for additional details. if set in a QoS profile document

8.343.2.3 stack_size

```
int stack_size
```

The thread stack-size.

[range] Platform-dependent. Consult `Platform Notes` for additional details.

8.343.2.4 cpu_list

```
final IntSeq cpu_list = new IntSeq()
```

The list of processors on which the thread(s) may run.

A sequence of integers that represent the set of processors on which the thread(s) controlled by this QoS may run. An empty sequence (the default) means the middleware wlll make no CPU affinity adjustments.

Note: This feature is currently only supported on a subset of architectures (see the `Platform Notes`). The API may change as more architectures are added in future releases.

This value is only relevant to the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523). It is ignored within other QoS policies that include **com.rti.dds.infrastructure.ThreadSettings_t** (p. 1792).

See also

Controlling CPU Core Affinity for RTI Threads (p. 1795)

[default] Empty sequence

8.343.2.5 cpu_rotation

```
ThreadSettingsCpuRotationKind cpu_rotation
```

Initial value:

```
=
        ThreadSettingsCpuRotationKind.THREAD_SETTINGS_CPU_NO_ROTATION
```

Determines how processor affinity is applied to multiple threads.

This value is only relevant to the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523). It is ignored within other QoS policies that include **com.rti.dds.infrastructure.ThreadSettings_t** (p. 1792).

See also

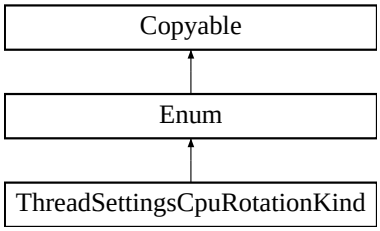
Controlling CPU Core Affinity for RTI Threads (p. 1795)

Note: This feature is currently only supported on a subset of architectures (see the `Platform Notes`). The API may change as more architectures are added in future releases.

8.344 ThreadSettingsCpuRotationKind Class Reference

Determines how **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794) affects processor affinity for thread-related QoS policies that apply to multiple threads.

Inheritance diagram for ThreadSettingsCpuRotationKind:



Static Public Attributes

- static final **ThreadSettingsCpuRotationKind THREAD_SETTINGS_CPU_NO_ROTATION**
  Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.
- static final **ThreadSettingsCpuRotationKind THREAD_SETTINGS_CPU_RR_ROTATION**
  Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

Additional Inherited Members

8.344.1 Detailed Description

Determines how **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794) affects processor affinity for thread-related QoS policies that apply to multiple threads.

8.344.2 Controlling CPU Core Affinity for RTI Threads

Most thread-related QoS settings apply to a single thread (such as for the **com.rti.dds.infrastructure.EventQosPolicy** (p. 1044), **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 445), and **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy** (p. 339)). However, the thread settings in the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523) control every receive thread created. In this case, there are several schemes to map M threads to N processors; the rotation kind controls which scheme is used.

Controlling CPU Core Affinity is only relevant to the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1523). It is ignored within other QoS policies that include **com.rti.dds.infrastructure.ThreadSettings_t** (p. 1792).

If **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794) is empty, the rotation is irrelevant since no affinity adjustment will occur. Suppose instead that **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **com.rti.dds.infrastructure.ThreadSettings_t.cpu_rotation** (p. 1794) is **com.rti.dds.infrastructure.ThreadSettingsCpuRotationKind.THREAD_SETTINGS_CPU_NO_ROTATION** (p. 1796), threads A, B and C will have the same processor affinities (0-1), and the OS will control thread scheduling within this bound. It is common to denote CPU affinities as a bitmask, where set bits represent

allowed processors to run on. This mask is printed in hex, so a CPU core affinity of 0-1 can be represented by the mask 0x3.

If **com.rti.dds.infrastructure.ThreadSettings_t.cpu_rotation** (p. 1794) is **com.rti.dds.infrastructure.Thread↩ SettingsCpuRotationKind.THREAD_SETTINGS_CPU_RR_ROTATION** (p. 1796), each thread will be assigned in round-robin fashion to one of the processors in **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1794); perhaps thread A to 0, B to 1, and C to 0. Note that the order in which internal middleware threads spawn is unspecified.

Not all of these options may be relevant for all operating systems. Refer to the Platform Notes for further information.

8.344.3 Member Data Documentation

8.344.3.1 THREAD_SETTINGS_CPU_NO_ROTATION

```
final ThreadSettingsCpuRotationKind THREAD_SETTINGS_CPU_NO_ROTATION [static]
```

Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.

8.344.3.2 THREAD_SETTINGS_CPU_RR_ROTATION

```
final ThreadSettingsCpuRotationKind THREAD_SETTINGS_CPU_RR_ROTATION [static]
```

Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

8.345 ThreadSettingsKind Class Reference

A collection of flags used to configure threads of execution.

Static Public Attributes

- static final int **THREAD_SETTINGS_KIND_MASK_DEFAULT**

  The mask of default thread options.
- static final int **THREAD_SETTINGS_FLOATING_POINT**

  Code executed within the thread may perform floating point operations. Currently applicable only for VxWorks platforms, where user callbacks use floating-point operations.
- static final int **THREAD_SETTINGS_STDIO**

  Code executed within the thread may access standard I/O. Currently applicable only for VxWorks platforms, where user callbacks do standard I/O operations.
- static final int **THREAD_SETTINGS_REALTIME_PRIORITY**

  The thread will be scheduled on a FIFO basis.
- static final int **THREAD_SETTINGS_PRIORITY_ENFORCE**

  Strictly enforce this thread's priority.
- static final int **THREAD_SETTINGS_CANCEL_ASYNCHRONOUS**

  Allows the thread to be cancelled without first reaching a cancellable state or cancellation point.

8.345.1 Detailed Description

A collection of flags used to configure threads of execution.

Not all of these options may be relevant for all operating systems. Consult `Platform Notes` for additional details.

See also

com.rti.dds.infrastructure.ThreadSettingsKindMask

8.345.2 Member Data Documentation

8.345.2.1 THREAD_SETTINGS_FLOATING_POINT

```
final int THREAD_SETTINGS_FLOATING_POINT  [static]
```

Code executed within the thread may perform floating point operations. Currently applicable only for VxWorks platforms, where user callbacks use floating-point operations.

8.345.2.2 THREAD_SETTINGS_STDIO

```
final int THREAD_SETTINGS_STDIO  [static]
```

Code executed within the thread may access standard I/O. Currently applicable only for VxWorks platforms, where user callbacks do standard I/O operations.

8.345.2.3 THREAD_SETTINGS_REALTIME_PRIORITY

```
final int THREAD_SETTINGS_REALTIME_PRIORITY  [static]
```

The thread will be scheduled on a FIFO basis.

8.345.2.4 THREAD_SETTINGS_PRIORITY_ENFORCE

```
final int THREAD_SETTINGS_PRIORITY_ENFORCE  [static]
```

Strictly enforce this thread's priority.

8.345.2.5 THREAD_SETTINGS_CANCEL_ASYNCHRONOUS

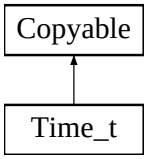
```
final int THREAD_SETTINGS_CANCEL_ASYNCHRONOUS  [static]
```

Allows the thread to be cancelled without first reaching a cancellable state or cancellation point.

8.346 Time_t Class Reference

Type for *time* representation.

Inheritance diagram for Time_t:



Public Member Functions

- **Time_t** (**Time_t** time)
  Copy constructor.
- **Time_t** (long **sec**, int **nanosec**)
  Constructor.
- **Time_t** ()
  Construct a new (invalid) ***Time_t*** *(*p. *1798).*
- boolean **is_invalid** ()
- boolean **is_zero** ()
  Check if time is zero.
- Object **copy_from** (Object src)
  Copy value of a data type from source.

Static Public Member Functions

- static **Time_t from_micros** (long micros)
  Creates a new time object from a time expressed in microseconds.
- static **Time_t from_nanos** (long nanos)
  Creates a new time object from a time expressed in nanoseconds.
- static **Time_t from_millis** (long millis)
  Creates a new time object from a time expressed in milliseconds.
- static **Time_t from_seconds** (long seconds)
  Creates a new time object from a time expressed in seconds.

Public Attributes

- long **sec**

  seconds
- int **nanosec**

  nanoseconds

Static Public Attributes

- static final long **TIME_INVALID_SEC**

  A sentinel indicating an invalid second of time.
- static final int **TIME_INVALID_NSEC**

  A sentinel indicating an invalid nano-second of time.
- static final **Time_t** **TIME_INVALID**

  A sentinel indicating an invalid time.
- static final **Time_t** **TIME_MAX**

  The maximum value of time.

8.346.1 Detailed Description

Type for *time* representation.

A **com.rti.dds.infrastructure.Time_t** (p. 1798) represents a moment in time.

8.346.2 Constructor & Destructor Documentation

8.346.2.1 Time_t() [1/3]

```
Time_t (
            Time_t time )
```

Copy constructor.

Parameters

time	The instance to copy. It must not be null.

References **Time_t.nanosec**, and **Time_t.sec**.

8.346.2.2 Time_t() [2/3]

```
Time_t (
            long sec,
            int nanosec )
```

Constructor.

Parameters

sec	must be >=0
nanosec	must be >=0 and < 1000000000

References **Time_t.nanosec**, and **Time_t.sec**.

8.346.2.3 Time_t() [3/3]

```
Time_t ( )
```

Construct a new (invalid) **Time_t** (p. 1798).

The primary purpose of this constructor is to satisfy the `java.io.Serializable` interface. The resulting object will have invalid seconds and nanoseconds.

References **Time_t.TIME_INVALID_NSEC**, and **Time_t.TIME_INVALID_SEC**.

Referenced by **Time_t.copy_from()**, **Time_t.from_micros()**, **Time_t.from_millis()**, **Time_t.from_nanos()**, and **Time_t.from_seconds()**.

8.346.3 Member Function Documentation

8.346.3.1 from_micros()

```
static Time_t from_micros (
            long micros )  [static]
```

Creates a new time object from a time expressed in microseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Time_t.TIME_MAX** (p. 1803).

References **Time_t.nanosec**, **Time_t.TIME_MAX**, and **Time_t.Time_t()**.

8.346.3.2 from_nanos()

```
static  Time_t from_nanos (
             long nanos )  [static]
```

Creates a new time object from a time expressed in nanoseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Time_t.TIME_MAX** (p. 1803).

References **Time_t.nanosec**, **Time_t.TIME_MAX**, and **Time_t.Time_t()**.

8.346.3.3 from_millis()

```
static  Time_t from_millis (
             long millis )  [static]
```

Creates a new time object from a time expressed in milliseconds.

In case of an overflow this function returns **com.rti.dds.infrastructure.Time_t.TIME_MAX** (p. 1803).

References **Time_t.nanosec**, **Time_t.TIME_MAX**, and **Time_t.Time_t()**.

8.346.3.4 from_seconds()

```
static  Time_t from_seconds (
             long seconds )  [static]
```

Creates a new time object from a time expressed in seconds.

References **Time_t.Time_t()**.

8.346.3.5 is_invalid()

```
boolean is_invalid ( )
```

Returns

com.rti.dds.infrastructure.true if the given time is not valid (i.e. is negative)

References **Time_t.nanosec**, and **Time_t.sec**.

Referenced by **DynamicDataWriter.register_instance_w_timestamp()**.

8.346.3.6 is_zero()

```
boolean is_zero ( )
```

Check if time is zero.

Returns

com.rti.dds.infrastructure.true if the given time is equal to com.rti.dds.infrastructure.Time_t.ZERO or com.rti.dds.↩ infrastructure.false otherwise.

References **Time_t.nanosec**, and **Time_t.sec**.

8.346.3.7 copy_from()

```
Object copy_from (
             Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

References **Time_t.Time_t()**.

8.346.4 Member Data Documentation

8.346.4.1 TIME_INVALID_SEC

```
final long TIME_INVALID_SEC  [static]
```

A sentinel indicating an invalid second of time.

Referenced by **Time_t.Time_t()**.

8.346.4.2 TIME_INVALID_NSEC

```
final int TIME_INVALID_NSEC  [static]
```

A sentinel indicating an invalid nano-second of time.

Referenced by **Time_t.Time_t()**.

8.346.4.3 TIME_INVALID

```
final  Time_t TIME_INVALID  [static]
```

A sentinel indicating an invalid time.

8.346.4.4 TIME_MAX

```
final  Time_t TIME_MAX  [static]
```

The maximum value of time.

Referenced by **Time_t.from_micros()**, **Time_t.from_millis()**, and **Time_t.from_nanos()**.

8.346.4.5 sec

```
long sec
```

seconds

Referenced by **SampleInfo.copy_from()**, **WriteParams_t.copy_from()**, **DynamicDataWriter.dispose_w_timestamp()**, **Time_t.is_invalid()**, **Time_t.is_zero()**, **DynamicDataWriter.register_instance_w_timestamp()**, **Time_t.Time_t()**, **DynamicDataWriter.unregister_instance_w_timestamp()**, **DynamicDataWriter.write_w_timestamp()**, and **WriteParams_t.WriteParams_t()**.

8.346.4.6 nanosec

```
int nanosec
```

nanoseconds

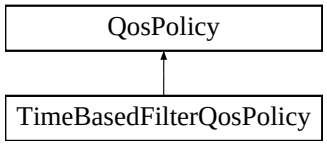
[range] [0,1000000000)

Referenced by **SampleInfo.copy_from()**, **WriteParams_t.copy_from()**, **DynamicDataWriter.dispose_w_timestamp()**, **Time_t.from_micros()**, **Time_t.from_millis()**, **Time_t.from_nanos()**, **Time_t.is_invalid()**, **Time_t.is_zero()**, **DynamicDataWriter.register_instance_w_timestamp()**, **Time_t.Time_t()**, **DynamicDataWriter.unregister_instance_w_timestamp()**, **DynamicDataWriter.write_w_timestamp()**, and **WriteParams_t.WriteParams_t()**.

8.347 TimeBasedFilterQosPolicy Class Reference

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.

Inheritance diagram for TimeBasedFilterQosPolicy:



Public Attributes

- final **Duration_t minimum_separation**

  The minimum separation duration between subsequent samples.

8.347.1 Detailed Description

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 450) to specify that it is interested only in (potentially) a subset of the values of the data.

The filter states that the **com.rti.dds.subscription.DataReader** (p. 450) does not want to receive more than one value each `minimum_separation`, regardless of how fast the changes occur.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **YES** (p. 256)

8.347.2 Usage

You can use this QoS policy to reduce the amount of data received by a **com.rti.dds.subscription.DataReader** (p. 450). **com.rti.dds.publication.DataWriter** (p. 553) entities may send data faster than needed by a **com.rti.dds.↩ subscription.DataReader** (p. 450). For example, a **com.rti.dds.subscription.DataReader** (p. 450) of sensor data that is displayed to a human operator in a GUI application does not need to receive data updates faster than a user can reasonably perceive changes in data values. This is often measured in tenths (0.1) of a second up to several seconds. However, a **com.rti.dds.publication.DataWriter** (p. 553) of sensor information may have other **com.rti.dds.↩ subscription.DataReader** (p. 450) entities that are processing the sensor information to control parts of the system and thus need new data updates in measures of hundredths (0.01) or thousandths (0.001) of a second.

With this QoS policy, different **com.rti.dds.subscription.DataReader** (p. 450) entities can set their own time-based filters, so that data published faster than the period set by a each **com.rti.dds.subscription.DataReader** (p. 450) will not be delivered to that **com.rti.dds.subscription.DataReader** (p. 450).

The **TIME_BASED_FILTER** (p. 267) also applies to each instance separately; that is, the constraint is that the **com.rti.↩ dds.subscription.DataReader** (p. 450) does not want to see more than one sample of each instance per `minimum↩ _separation` period.

This QoS policy allows you to optimize resource usage (CPU and possibly network bandwidth) by only delivering the required amount of data to each **com.rti.dds.subscription.DataReader** (p. 450), accommodating the fact that, for rapidly-changing data, different subscribers may have different requirements and constraints as to how frequently they need or can handle being notified of the most current values. As such, it can also be used to protect applications that are running on a heterogeneous network where some nodes are capable of generating data much faster than others can consume it.

For best effort data delivery, if the data type is unkeyed and the **com.rti.dds.publication.DataWriter** (p. 553) has an infinite **com.rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1246), RTI Connext will only send as many packets to a **com.rti.dds.subscription.DataReader** (p. 450) as required by the TIME_BASED_FILTER, no matter how fast **com.rti.ndds.example.FooDataWriter.write** (p. 1105) is called.

For multicast data delivery to multiple DataReaders, the one with the lowest `minimum_separation` determines the DataWriter's send rate. For example, if a **com.rti.dds.publication.DataWriter** (p. 553) sends over multicast to two DataReaders, one with `minimum_separation` of 2 seconds and one with `minimum_separation` of 1 second, the DataWriter will send every 1 second.

In configurations where RTI Connext must send all the data published by the **com.rti.dds.publication.DataWriter** (p. 553) (for example, when the **com.rti.dds.publication.DataWriter** (p. 553) is reliable, when the data type is keyed, or when the **com.rti.dds.publication.DataWriter** (p. 553) has a finite **com.rti.dds.infrastructure.LivelinessQos↩ Policy.lease_duration** (p. 1246)), only the data that passes the TIME_BASED_FILTER will be stored in the receive queue of the **com.rti.dds.subscription.DataReader** (p. 450). Extra data will be accepted but dropped. Note that filtering is only applied on alive samples (that is, samples that have not been disposed/unregistered).

8.347.3 Consistency

It is inconsistent for a **com.rti.dds.subscription.DataReader** (p. 450) to have a `minimum_separation` longer than its **DEADLINE** (p. 217) `period`.

However, it is important to be aware of certain edge cases that can occur when your publication rate, minimum separation, and deadline period align and that can cause missed deadlines that you may not expect. For example, suppose that you nominally publish samples every second but that this rate can vary somewhat over time. You declare a minimum

separation of 1 second to filter out rapid updates and set a deadline of two seconds so that you will be aware if the rate falls too low. Even if your update rate never wavers, you can still miss deadlines! Here's why:

Suppose you publish the first sample at time t=0 seconds. You then publish your next sample at t=1 seconds. Depending on how your operating system schedules the time-based filter execution relative to the publication, this second sample may be filtered. You then publish your third sample at t=2 seconds, and depending on how your OS schedules this publication in relation to the deadline check, you could miss the deadline.

This scenario demonstrates a couple of rules of thumb:

- Beware of setting your `minimum_separation` to a value very close to your publication rate: you may filter more data than you intend to.
- Beware of setting your `minimum_separation` to a value that is too close to your deadline period relative to your publication rate. You may miss deadlines.

See **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 632) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 267) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 237) and **RELIABILITY** (p. 258) QoS. The **TIME_BASED_FILTER** (p. 267) specifies the samples that are of interest to the **com.rti.dds.subscription.DataReader** (p. 450). The **HISTORY** (p. 237) and **RELIABILITY** (p. 258) QoS affect the behavior of the middleware with respect to the samples that have been determined to be of interest to the **com.rti.dds.subscription.DataReader** (p. 450); that is, they apply *after* the **TIME_BASED_FILTER** (p. 267) has been applied.

In the case where the reliability QoS kind is **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532), in steady-state – defined as the situation where the **com.rti.dds.publication.DataWriter** (p. 553) does not write new samples for a period "long" compared to the `minimum_separation` – the system should guarantee delivery of the last sample to the **com.rti.dds.subscription.DataReader** (p. 450).

See also

DeadlineQosPolicy (p. 632)

HistoryQosPolicy (p. 1144)

ReliabilityQosPolicy (p. 1526)

8.347.4 Member Data Documentation

8.347.4.1 minimum_separation

```
final Duration_t minimum_separation
```

The minimum separation duration between subsequent samples.

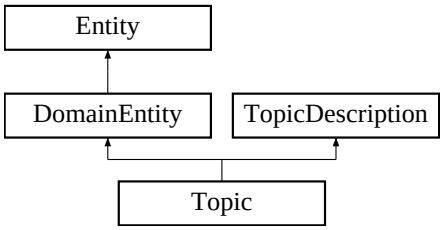
[default] 0 (meaning the **com.rti.dds.subscription.DataReader** (p. 450) is potentially interested in all values)

[range] [0,1 year], $<$ **com.rti.dds.infrastructure.DeadlineQosPolicy.period** (p. 634)

8.348 Topic Interface Reference

<<***interface***>> *(*p. *156)* The most basic description of the data to be published and subscribed.

Inheritance diagram for Topic:



Public Member Functions

- void **get_inconsistent_topic_status** (**InconsistentTopicStatus** status)

  Allows the application to retrieve the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status of a ***com.rti.dds.topic.Topic*** *(*p. *1807).*
- void **set_qos** (**TopicQos** qos)

  Set the topic QoS.
- void **set_qos_with_profile** (String library_name, String profile_name)

  <<***extension***>> *(*p. *155) Change the QoS of this topic using the input XML QoS profile.*
- void **get_qos** (**TopicQos** qos)

  Get the topic QoS.
- void **set_listener** (**TopicListener** l, int mask)

  Set the topic listener.
- **TopicListener get_listener** ()

  Get the topic listener.

8.348.1 Detailed Description

<<***interface***>> *(*p. *156)* The most basic description of the data to be published and subscribed.

QoS:

com.rti.dds.topic.TopicQos (p. 1824)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS, **com.rti.dds.topic.↩ InconsistentTopicStatus** (p. 1149)

Listener:

com.rti.dds.topic.TopicListener (p. 1822)

A **com.rti.dds.topic.Topic** (p. 1807) is identified by its name, which must be unique in the whole domain. In addition (by virtue of extending **com.rti.dds.topic.TopicDescription** (p. 1820)) it fully specifies the type of the data that can be communicated when publishing or subscribing to the **com.rti.dds.topic.Topic** (p. 1807).

com.rti.dds.topic.Topic (p. 1807) is the only **com.rti.dds.topic.TopicDescription** (p. 1820) that can be used for publications and therefore associated with a **com.rti.dds.publication.DataWriter** (p. 553).

The following operations may be called even if the **com.rti.dds.topic.Topic** (p. 1807) is not enabled. Other operations will fail with the value **com.rti.dds.infrastructure.RETCODE_NOT_ENABLED** (p. 1597) if called on a disabled **com.↩ rti.dds.topic.Topic** (p. 1807):

- **com.rti.dds.infrastructure.Entity.enable** (p. 1032)
- **com.rti.dds.topic.Topic.set_qos** (p. 1809), **com.rti.dds.topic.Topic.get_qos** (p. 1810)
- **com.rti.dds.topic.Topic.set_qos_with_profile** (p. 1809)
- **com.rti.dds.topic.Topic.set_listener** (p. 1811), **com.rti.dds.topic.Topic.get_listener** (p. 1811)
- **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 1034), **com.rti.dds.infrastructure.Entity.get_↩ status_changes** (p. 1034)
- **com.rti.dds.topic.Topic.get_inconsistent_topic_status** (p. 1808)

See also

Operations Allowed in Listener Callbacks (p. 1238)

8.348.2 Member Function Documentation

8.348.2.1 get_inconsistent_topic_status()

```
void get_inconsistent_topic_status (
              InconsistentTopicStatus status )
```

Allows the application to retrieve the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_↩ STATUS status of a **com.rti.dds.topic.Topic** (p. 1807).

Retrieve the current **com.rti.dds.topic.InconsistentTopicStatus** (p. 1149)

Parameters

status	<<***inout***>> *(p. 156)* Status to be retrieved. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

com.rti.dds.topic.InconsistentTopicStatus (p. 1149)

8.348.2.2 set_qos()

```
void set_qos (
              TopicQos qos )
```

Set the topic QoS.

The **com.rti.dds.topic.TopicQos.topic_data** (p. 1827) and **com.rti.dds.topic.TopicQos.deadline** (p. 1828), **com.↩ rti.dds.topic.TopicQos.latency_budget** (p. 1828), **com.rti.dds.topic.TopicQos.transport_priority** (p. 1829) and **com.rti.dds.topic.TopicQos.lifespan** (p. 1829) can be changed. The other policies are immutable.

Parameters

qos	<<***in***>> *(p. 156)* Set of policies to be applied to **com.rti.dds.topic.Topic** (p. 1807).

Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.topic.Topic** (p. 1807) is enabled. The special value **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_DEFAULT** (p. 45) can be used to indicate that the QoS of the **com.rti.dds.topic.Topic** (p. 1807) should be changed to match the current default **com.rti.dds.topic.↩ TopicQos** (p. 1824) set in the **com.rti.dds.domain.DomainParticipant** (p. 670). Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) if immutable policy is changed, or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) if policies are inconsistent

See also

com.rti.dds.topic.TopicQos (p. 1824) for rules on consistency among QoS

set_qos (abstract) (p. 1030)

Operations Allowed in Listener Callbacks (p. 1238)

8.348.2.3 set_qos_with_profile()

```
void set_qos_with_profile (
            String library_name,
            String profile_name )
```

<<***extension***>> *(p. 155)* Change the QoS of this topic using the input XML QoS profile.

The **com.rti.dds.topic.TopicQos.topic_data** (p. 1827) and **com.rti.dds.topic.TopicQos.deadline** (p. 1828), **com.↩ rti.dds.topic.TopicQos.latency_budget** (p. 1828), **com.rti.dds.topic.TopicQos.transport_priority** (p. 1829) and **com.rti.dds.topic.TopicQos.lifespan** (p. 1829) can be changed. The other policies are immutable.

Parameters

library_name	<<***in***>> *(p. 156)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the default library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 716)).
profile_name	<<***in***>> *(p. 156)* XML QoS Profile name. If profile_name is null RTI Connext will use the default profile (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 717)).

Exceptions

One	of the **Standard Return Codes** (p. 261), **com.rti.dds.infrastructure.RETCODE_IMMUTABLE_POLICY** (p. 1596) if immutable policy is changed, or **com.rti.dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) if policies are inconsistent

See also

com.rti.dds.topic.TopicQos (p. 1824) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1238)

8.348.2.4 get_qos()

```
void get_qos (
             TopicQos qos )
```

Get the topic QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

qos	<<***inout***>> *(p. 156)* QoS to be filled up. Cannot be NULL.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

get_qos (abstract) (p. 1031)

8.348.2.5 set_listener()

```
void set_listener (
             TopicListener l,
            int mask )
```

Set the topic listener.

Parameters

l	<<***in***>> *(p. 156)* Listener to be installed on entity.
mask	<<***in***>> *(p. 156)* Changes of communication status to be invoked on the listener. See com.rti.dds.infrastructure.StatusMask.

Exceptions

One	of the **Standard Return Codes** (p. 261)

See also

set_listener (abstract) (p. 1031)

8.348.2.6 get_listener()

```
TopicListener get_listener ( )
```

Get the topic listener.

Returns

Existing listener attached to the **com.rti.dds.topic.Topic** (p. 1807).

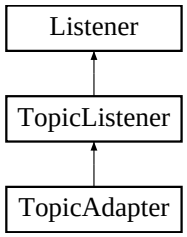
See also

get_listener (abstract) (p. 1032)

8.349 TopicAdapter Class Reference

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Inheritance diagram for TopicAdapter:



Public Member Functions

- void **on_inconsistent_topic** (**Topic** topic, **InconsistentTopicStatus** status)

  Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.

8.349.1 Detailed Description

<<***extension***>> *(p. 155)* A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)

Clients who do not wish to implement all listener methods can subclass this class and override only the methods of interest.

8.349.2 Member Function Documentation

8.349.2.1 on_inconsistent_topic()

```
void on_inconsistent_topic (
             Topic topic,
             InconsistentTopicStatus status )
```

Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1807) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1807) of the same topic name.

Parameters

topic	<<***out***>> *(p. 156)* Locally created **com.rti.dds.topic.Topic** (p. 1807) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1807)

Implements **TopicListener** (p. 1823).

8.350 TopicBuiltinTopicData Class Reference

Entry created when a **Topic** (p. 1807) object discovered.

Inherits AbstractBuiltinTopicData.

Public Attributes

- final **BuiltinTopicKey_t key**
  DCPS key to distinguish entries.
- String **name**
  *Name of the **com.rti.dds.topic.Topic** (p. 1807).*
- String **type_name**
  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).*
- final **DurabilityQosPolicy durability**
  *durability policy of the corresponding **Topic** (p. 1807)*
- final **DurabilityServiceQosPolicy durability_service**
  *durability service policy of the corresponding **Topic** (p. 1807)*
- final **DeadlineQosPolicy deadline**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **LatencyBudgetQosPolicy latency_budget**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **LivelinessQosPolicy liveliness**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **ReliabilityQosPolicy reliability**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **TransportPriorityQosPolicy transport_priority**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **LifespanQosPolicy lifespan**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **DestinationOrderQosPolicy destination_order**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **HistoryQosPolicy history**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **ResourceLimitsQosPolicy resource_limits**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **OwnershipQosPolicy ownership**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **TopicDataQosPolicy topic_data**
  *Policy of the corresponding **Topic** (p. 1807).*
- final **DataRepresentationQosPolicy representation**
  *Data representation policy of the corresponding **Topic** (p. 1807).*

8.350.1 Detailed Description

Entry created when a **Topic** (p. 1807) object discovered.

Data associated with the built-in topic **com.rti.dds.topic.builtin.TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME** (p. 165). It contains QoS policies and additional information that apply to the remote **com.rti.dds.topic.Topic** (p. 1807).

Note: The **builtin.TopicBuiltinTopicData** (p. 1813) built-in topic is meant to convey information about discovered Topics. This **Topic** (p. 1807)'s samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762)). Therefore **TopicBuiltinTopicData** (p. 1813) DataReaders will not receive any data.

See also

com.rti.dds.topic.builtin.TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME (p. 165)

builtin.TopicBuiltinTopicDataDataReader (p. 1817)

8.350.2 Member Data Documentation

8.350.2.1 key

```
final BuiltinTopicKey_t key
```

DCPS key to distinguish entries.

8.350.2.2 name

```
String name
```

Name of the **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.350.2.3 type_name

```
String type_name
```

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1807).

The length of this string is limited to 255 characters.

8.350.2.4 durability

```
final DurabilityQosPolicy durability
```

durability policy of the corresponding **Topic** (p. 1807)

8.350.2.5 durability_service

```
final DurabilityServiceQosPolicy durability_service
```

durability service policy of the corresponding **Topic** (p. 1807)

8.350.2.6 deadline

```
final DeadlineQosPolicy deadline
```

Policy of the corresponding **Topic** (p. 1807).

8.350.2.7 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Policy of the corresponding **Topic** (p. 1807).

8.350.2.8 liveliness

```
final LivelinessQosPolicy liveliness
```

Policy of the corresponding **Topic** (p. 1807).

8.350.2.9 reliability

```
final ReliabilityQosPolicy reliability
```

Policy of the corresponding **Topic** (p. 1807).

8.350.2.10 transport_priority

final **TransportPriorityQosPolicy** transport_priority

Policy of the corresponding **Topic** (p. 1807).

8.350.2.11 lifespan

final **LifespanQosPolicy** lifespan

Policy of the corresponding **Topic** (p. 1807).

8.350.2.12 destination_order

final **DestinationOrderQosPolicy** destination_order

Policy of the corresponding **Topic** (p. 1807).

8.350.2.13 history

final **HistoryQosPolicy** history

Policy of the corresponding **Topic** (p. 1807).

8.350.2.14 resource_limits

final **ResourceLimitsQosPolicy** resource_limits

Policy of the corresponding **Topic** (p. 1807).

8.350.2.15 ownership

final **OwnershipQosPolicy** ownership

Policy of the corresponding **Topic** (p. 1807).

8.350.2.16 topic_data

```
final TopicDataQosPolicy topic_data
```

Policy of the corresponding **Topic** (p. 1807).

8.350.2.17 representation

```
final DataRepresentationQosPolicy representation
```

Data representation policy of the corresponding **Topic** (p. 1807).

8.351 TopicBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **builtin.TopicBuiltinTopicData** (p. 1813) > .

Inherits DataReaderImpl.

8.351.1 Detailed Description

Instantiates `DataReader` < **builtin.TopicBuiltinTopicData** (p. 1813) > .

com.rti.dds.subscription.DataReader (p. 450) of topic **com.rti.dds.topic.builtin.TopicBuiltinTopicDataType↩ Support.TOPIC_TOPIC_NAME** (p. 165) used for accessing **builtin.TopicBuiltinTopicData** (p. 1813) of the remote **com.rti.dds.topic.Topic** (p. 1807).

Note: The **builtin.TopicBuiltinTopicData** (p. 1813) built-in topic is meant to convey information about discovered Topics. This **Topic** (p. 1807)'s samples are not propagated in a separate packet on the wire. Instead, the data is sent as part of the information carried by other built-in topics (**com.rti.dds.publication.builtin.PublicationBuiltinTopicData** (p. 1452) and **com.rti.dds.subscription.builtin.SubscriptionBuiltinTopicData** (p. 1762)). Therefore **TopicBuiltin↩ TopicData** (p. 1813) DataReaders will not receive any data.

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooDataReader** (p. 1067)

See also

builtin.TopicBuiltinTopicData (p. 1813)

com.rti.dds.topic.builtin.TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME (p. 165)

8.352 TopicBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **builtin.TopicBuiltin↩ TopicData** (p. 1813) > .

Inherits AbstractBuiltinTopicDataSeq.

8.352.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < **builtin.TopicBuiltin↩ TopicData** (p. 1813) > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

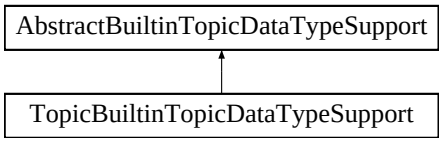
See also

builtin.TopicBuiltinTopicData (p. 1813)

8.353 TopicBuiltinTopicDataTypeSupport Class Reference

Instantiates **`TypeSupport`** (p. `1941`) < **builtin.TopicBuiltinTopicData** (p. 1813) > .

Inheritance diagram for TopicBuiltinTopicDataTypeSupport:



Static Public Attributes

- static final String **TOPIC_TOPIC_NAME** = DDS_TOPIC_TOPIC_NAME()
  Topic *(*p. *1807) topic name.*

Additional Inherited Members

8.353.1 Detailed Description

Instantiates **`TypeSupport`** (p. `1941`) < **builtin.TopicBuiltinTopicData** (p. 1813) > .

Instantiates:

<<***generic***>> *(*p. *156)* **com.rti.ndds.example.FooTypeSupport** (p. 1118)

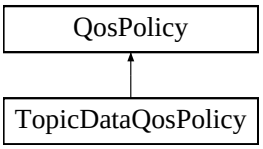
See also

builtin.TopicBuiltinTopicData (p. 1813)

8.354 TopicDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Inheritance diagram for TopicDataQosPolicy:



Public Attributes

- final **ByteSeq value**
  a sequence of octets

8.354.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Entity:

com.rti.dds.topic.Topic (p. 1807)

Properties:

RxO (p. 256) = NO
Changeable (p. 256) = **YES** (p. 256)

See also

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.354.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **com.rti.dds.topic.Topic** (p. 1807) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This extra data is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with **com.rti.dds.subscription.DataReaderListener** (p. 497), **com.rti.dds.publication.DataWriterListener** (p. 589), or operations such as **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 724), this QoS policy can assist an application in defining and enforcing its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.topic_data_max_length** (p. 815).

8.354.3 Member Data Documentation

8.354.3.1 value

`final` **`ByteSeq`** `value`

a sequence of octets

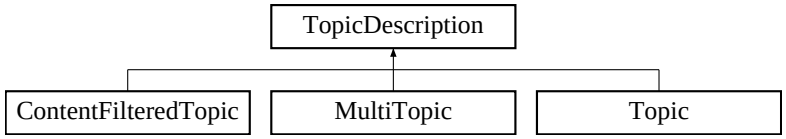
[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

8.355 TopicDescription Interface Reference

com.rti.dds.topic.Topic (p. 1807) entity and associated elements

Inheritance diagram for TopicDescription:



Public Member Functions

- String **get_type_name** ()

  Get the associated `type_name`*.*
- String **get_name** ()

  Get the name used to create this ***com.rti.dds.topic.TopicDescription*** *(*p. *1820) .*
- **DomainParticipant get_participant** ()

  Get the ***com.rti.dds.domain.DomainParticipant*** *(*p. *670) to which the* ***com.rti.dds.topic.TopicDescription*** *(*p. *1820) belongs.*

8.355.1 Detailed Description

com.rti.dds.topic.Topic (p. 1807) entity and associated elements

<<***interface***>> *(*p. *156)* Base class for **com.rti.dds.topic.Topic** (p. 1807), **com.rti.dds.topic.ContentFilteredTopic** (p. 436), and **com.rti.dds.topic.MultiTopic** (p. 1320).

com.rti.dds.topic.TopicDescription (p. 1820) represents the fact that both publications and subscriptions are tied to a single data-type. Its attribute `type_name` defines a unique resulting type for the publication or the subscription and therefore creates an implicit association with a **com.rti.dds.topic.TypeSupport** (p. 1941).

com.rti.dds.topic.TopicDescription (p. 1820) has also a `name` that allows it to be retrieved locally.

See also

com.rti.dds.topic.TypeSupport (p. 1941), **com.rti.ndds.example.FooTypeSupport** (p. 1118)

8.355.2 Member Function Documentation

8.355.2.1 get_type_name()

```
String get_type_name ( )
```

Get the associated `type_name`.

The type name defines a locally unique type for the publication or the subscription.

The `type_name` corresponds to a unique string used to register a type via the **com.rti.ndds.example.FooType↩ Support.register_type** (p. 1119) method.

Thus, the `type_name` implies an association with a corresponding **com.rti.dds.topic.TypeSupport** (p. 1941) and this **com.rti.dds.topic.TopicDescription** (p. 1820).

Returns

the type name. The returned type name is valid until the **com.rti.dds.topic.TopicDescription** (p. 1820) is deleted.

Postcondition

The result is non-NULL.

See also

com.rti.dds.topic.TypeSupport (p. 1941), **com.rti.ndds.example.FooTypeSupport** (p. 1118)

8.355.2.2 get_name()

```
String get_name ( )
```

Get the name used to create this **com.rti.dds.topic.TopicDescription** (p. 1820) .

Returns

the name used to create this **com.rti.dds.topic.TopicDescription** (p. 1820). The returned topic name is valid until the **com.rti.dds.topic.TopicDescription** (p. 1820) is deleted.

Postcondition

The result is non-NULL.

8.355.2.3 get_participant()

```
DomainParticipant get_participant ( )
```

Get the **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.topic.TopicDescription** (p. 1820) belongs.

Returns

The **com.rti.dds.domain.DomainParticipant** (p. 670) to which the **com.rti.dds.topic.TopicDescription** (p. 1820) belongs.

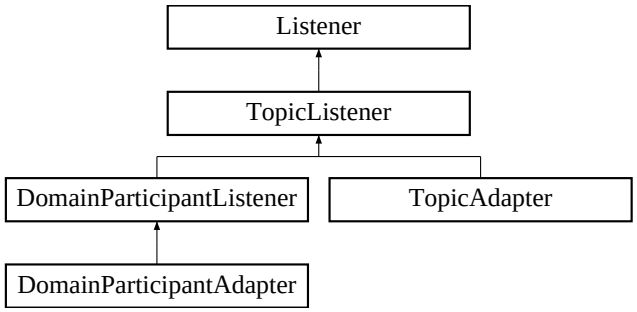
Postcondition

The result is non-NULL.

8.356 TopicListener Interface Reference

<<***interface***>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.topic.Topic** (p. 1807) entities.

Inheritance diagram for TopicListener:



Public Member Functions

- void **on_inconsistent_topic** (**Topic** topic, **InconsistentTopicStatus** status)

  Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.

8.356.1 Detailed Description

<<***interface***>> *(p. 156)* **com.rti.dds.infrastructure.Listener** (p. 1236) for **com.rti.dds.topic.Topic** (p. 1807) entities.

Entity:

com.rti.dds.topic.Topic (p. 1807)

Status:

com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS, **com.rti.dds.topic.InconsistentTopicStatus** (p. 1149)

This is the interface that can be implemented by an application-provided class and then registered with the **com.rti.dds.topic.Topic** (p. 1807) such that the application can be notified by RTI Connext of relevant status changes.

See also

Status Kinds (p. 262)

com.rti.dds.infrastructure.Listener (p. 1236)

com.rti.dds.topic.Topic.set_listener (p. 1811)

Operations Allowed in Listener Callbacks (p. 1238)

8.356.2 Member Function Documentation

8.356.2.1 on_inconsistent_topic()

```
void on_inconsistent_topic (
             Topic topic,
             InconsistentTopicStatus status )
```

Handle the com.rti.dds.infrastructure.StatusKind.StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1807) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1807) of the same topic name.

Parameters

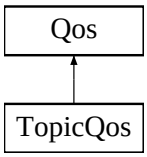
topic	<<***out***>> *(p. 156)* Locally created **com.rti.dds.topic.Topic** (p. 1807) that triggers the listener callback
status	<<***out***>> *(p. 156)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1807)

Implemented in **DomainParticipantAdapter** (p. 751), and **TopicAdapter** (p. 1812).

8.357 TopicQos Class Reference

QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1807) entity.

Inheritance diagram for TopicQos:



Public Member Functions

- String **toString** ()

  Overrides the builtin Object.toString method.
- String **toString** (**TopicQos** baseQos, **QosPrintFormat** format)

  *Obtains a string representation of a **TopicQos** (p. 1824) object.*
- String **toString** (**QosPrintFormat** format)

  *Obtains a string representation of a **TopicQos** (p. 1824) object.*
- String **toString** (**TopicQos** baseQos)

  *Obtains a string representation of a **TopicQos** (p. 1824) object.*

Public Attributes

- final **TopicDataQosPolicy topic_data**

  ***Topic** (p. 1807) data policy, **TOPIC_DATA** (p. 268).*
- final **DurabilityQosPolicy durability**

  *Durability policy, **DURABILITY** (p. 230).*
- final **DurabilityServiceQosPolicy durability_service**

  *DurabilityService policy, **DURABILITY_SERVICE** (p. 232).*
- final **DeadlineQosPolicy deadline**

  *Deadline policy, **DEADLINE** (p. 217).*
- final **LatencyBudgetQosPolicy latency_budget**

  *Latency budget policy, **LATENCY_BUDGET** (p. 238).*
- final **LivelinessQosPolicy liveliness**

  *Liveliness policy, **LIVELINESS** (p. 239).*
- final **ReliabilityQosPolicy reliability**

  *Reliability policy, **RELIABILITY** (p. 258).*
- final **DestinationOrderQosPolicy destination_order**

  *Destination order policy, **DESTINATION_ORDER** (p. 218).*
- final **HistoryQosPolicy history**

  *History policy, **HISTORY** (p. 237).*
- final **ResourceLimitsQosPolicy resource_limits**

  *Resource limits policy, **RESOURCE_LIMITS** (p. 259).*
- final **TransportPriorityQosPolicy transport_priority**

Transport priority policy, ***TRANSPORT_PRIORITY*** *(p. 274).*

- final **LifespanQosPolicy lifespan**

  Lifespan policy, ***LIFESPAN*** *(p. 238).*

- final **OwnershipQosPolicy ownership**

  Ownership policy, ***OWNERSHIP*** *(p. 244).*

- final **DataRepresentationQosPolicy representation**

  Data representation policy, ***DATA_REPRESENTATION*** *(p. 212).*

8.357.1 Detailed Description

QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1807) entity.

You must set certain members in a consistent manner:

length of **com.rti.dds.topic.TopicQos.topic_data** (p. 1827) .value <= **com.rti.dds.domain.DomainParticipant↩ Qos.resource_limits** (p. 800) .topic_data_max_length

If any of the above are not true, **com.rti.dds.topic.Topic.set_qos** (p. 1809), **com.rti.dds.topic.Topic.set_qos_with_↩ profile** (p. 1809) and **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 679) will fail with **com.rti.↩ dds.infrastructure.RETCODE_INCONSISTENT_POLICY** (p. 1596) and **com.rti.dds.domain.DomainParticipant.↩ create_topic** (p. 706) will return NULL.

Entity:

com.rti.dds.topic.Topic (p. 1807)

See also

QoS Policies (p. 250) allowed ranges within each Qos.

8.357.2 Member Function Documentation

8.357.2.1 toString() [1/4]

```
String toString ( )
```

Overrides the builtin Object.toString method.

The various **toString()** (p. 1825) overloads allow formatting the output and printing only the differences with respect to another **TopicQos** (p. 1824) object.

```
TopicQos qos = new TopicQos();
String theString = new String();
// The most basic version of the API simply overrides the builtin
// Object.toString method. Only the differences with respect to the
// documented default are printed to the string. The string is formatted
// according to the default values for QosPrintFormat.
theString = qos.toString();
// This overload allows us to specify a base profile. Only the differences
// with respect to this base profile are printed to the string. If the two
// Qos objects are equal, the resultant string will be empty.
TopicQos baseQos = new TopicQos(); // ...;
theString = qos.toString(baseQos);
// It is also possible to supply a custom format at this point
QosPrintFormat printFormat = new QosPrintFormat(); // ...;
theString = qos.toString(baseQos, format);
// The sentinel value TOPIC_QOS_PRINT_ALL can be used as
// the base in order to print the entire qos object
theString = qos.toString(TOPIC_QOS_PRINT_ALL);
```

This overload uses the default print format and only prints the differences between the supplied **TopicQos** (p. 1824) and the documented default.

Returns

The string representation of the Qos.

References **TopicQos.toString()**.

Referenced by **TopicQos.toString()**.

8.357.2.2 toString() [2/4]

```
String toString (
              TopicQos baseQos,
              QosPrintFormat format )
```

Obtains a string representation of a **TopicQos** (p. 1824) object.

Parameters

format	The print format used to format the output.
baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_PRINT_ALL** (p. 45) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **DomainParticipant.TOPIC_QOS_PRINT_ALL**.

8.357.2.3 toString() [3/4]

```
String toString (
              QosPrintFormat format )
```

Obtains a string representation of a **TopicQos** (p. 1824) object.

Parameters

format	The print format used to format the output.

This overload prints the differences between the qos and the documented. default. The output string is formatted using the supplied **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **TopicQos.toString()**.

8.357.2.4 toString() [4/4]

```
String toString (
              TopicQos baseQos )
```

Obtains a string representation of a **TopicQos** (p. 1824) object.

Parameters

baseQos	Only the differences between baseQos and the Qos object are included in the output string. If you want to print everything within the Qos, use the **com.rti.dds.domain.DomainParticipant.TOPIC_QOS_PRINT_ALL** (p. 45) sentinel value.

This overload prints the differences between the qos and the supplied baseQos. The output string is formatted using the default value for **com.rti.dds.infrastructure.QosPrintFormat** (p. 1507).

Returns

The string representation of the Qos.

References **TopicQos.toString()**.

8.357.3 Member Data Documentation

8.357.3.1 topic_data

```
final  TopicDataQosPolicy topic_data
```

Topic (p. 1807) data policy, **TOPIC_DATA** (p. 268).

8.357.3.2 durability

```
final DurabilityQosPolicy durability
```

Durability policy, **DURABILITY** (p. 230).

8.357.3.3 durability_service

```
final DurabilityServiceQosPolicy durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 232).

8.357.3.4 deadline

```
final DeadlineQosPolicy deadline
```

Deadline policy, **DEADLINE** (p. 217).

8.357.3.5 latency_budget

```
final LatencyBudgetQosPolicy latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 238).

8.357.3.6 liveliness

```
final LivelinessQosPolicy liveliness
```

Liveliness policy, **LIVELINESS** (p. 239).

8.357.3.7 reliability

```
final ReliabilityQosPolicy reliability
```

Reliability policy, **RELIABILITY** (p. 258).

8.357.3.8 destination_order

`final` **DestinationOrderQosPolicy** `destination_order`

Destination order policy, **DESTINATION_ORDER** (p. 218).

8.357.3.9 history

`final` **HistoryQosPolicy** `history`

History policy, **HISTORY** (p. 237).

8.357.3.10 resource_limits

`final` **ResourceLimitsQosPolicy** `resource_limits`

Resource limits policy, **RESOURCE_LIMITS** (p. 259).

8.357.3.11 transport_priority

`final` **TransportPriorityQosPolicy** `transport_priority`

Transport priority policy, **TRANSPORT_PRIORITY** (p. 274).

8.357.3.12 lifespan

`final` **LifespanQosPolicy** `lifespan`

Lifespan policy, **LIFESPAN** (p. 238).

8.357.3.13 ownership

`final` **OwnershipQosPolicy** `ownership`

Ownership policy, **OWNERSHIP** (p. 244).

8.357.3.14 representation

```
final DataRepresentationQosPolicy representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 212).

8.358 TopicQuery Interface Reference

<<***extension***>> *(p. 155)* Allows a **com.rti.dds.subscription.DataReader** (p. 450) to query the sample cache of its matching **com.rti.dds.publication.DataWriter** (p. 553).

Public Member Functions

- **GUID_t get_guid** ()

  *Get the **TopicQuery** (p. 1830)'s GUID.*

8.358.1 Detailed Description

<<***extension***>> *(p. 155)* Allows a **com.rti.dds.subscription.DataReader** (p. 450) to query the sample cache of its matching **com.rti.dds.publication.DataWriter** (p. 553).

8.358.2 Member Function Documentation

8.358.2.1 get_guid()

```
GUID_t get_guid ( )
```

Get the **TopicQuery** (p. 1830)'s GUID.

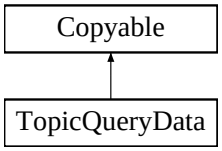
Exceptions

One	of the **Standard Return Codes** (p. 261)

8.359 TopicQueryData Class Reference

<<***extension***>> *(p. 155)* Provides information about a **com.rti.dds.subscription.TopicQuery** (p. 1830)

Inheritance diagram for TopicQueryData:



Public Member Functions

- Object **copy_from** (Object src)

Public Attributes

- **TopicQuerySelection topic_query_selection** = new **TopicQuerySelection**()

  The data selection.
- String **topic_name** = ""

  *The topic name of the **com.rti.dds.subscription.DataReader** (p. 450).*
- **GUID_t original_related_reader_guid** = new **GUID_t**()

  *Identifies the **com.rti.dds.subscription.DataReader** (p. 450) that created the **com.rti.dds.subscription.TopicQuery** (p. 1830).*

8.359.1 Detailed Description

<<***extension***>> *(p. 155)* Provides information about a **com.rti.dds.subscription.TopicQuery** (p. 1830)

Contains the information about a **TopicQuery** (p. 1830) that can be retrieved using **com.rti.dds.subscription.Topic↩ QueryHelper.topic_query_data_from_service_request** (p. 1835).

See also

com.rti.dds.subscription.TopicQueryHelper.topic_query_data_from_service_request (p. 1835)

Author

erin (12/18/2015)

8.359.2 Member Function Documentation

8.359.2.1 copy_from()

```
Object copy_from (
             Object src )
```

This is the implementation of the `Copyable` interface. This method will perform a deep copy of `src` This method could be placed into `TopicQueryDataTypeSupport` rather than here by using the `-noCopyable` option to rtiddsgen.

Parameters

src	The Object which contains the data to be copied.

Returns

Returns `this`.

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not the same type as `this`.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

References **TopicQuerySelection.copy_from()**, **TopicQueryData.original_related_reader_guid**, **TopicQueryData.topic_name**, and **TopicQueryData.topic_query_selection**.

8.359.3 Member Data Documentation

8.359.3.1 topic_query_selection

```
TopicQuerySelection topic_query_selection = new TopicQuerySelection()
```

The data selection.

Referenced by **TopicQueryData.copy_from()**.

8.359.3.2 topic_name

```
String topic_name = ""
```

The topic name of the **com.rti.dds.subscription.DataReader** (p. 450).

Referenced by **TopicQueryData.copy_from()**.

8.359.3.3 original_related_reader_guid

```
GUID_t original_related_reader_guid = new GUID_t()
```

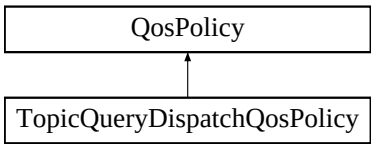
Identifies the **com.rti.dds.subscription.DataReader** (p. 450) that created the **com.rti.dds.subscription.TopicQuery** (p. 1830).

Referenced by **TopicQueryData.copy_from()**.

8.360 TopicQueryDispatchQosPolicy Class Reference

Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish samples in response to a **com.rti.↩ dds.subscription.TopicQuery** (p. 1830).

Inheritance diagram for TopicQueryDispatchQosPolicy:



Public Attributes

- boolean **enable** = false

  Allows this writer to dispatch TopicQueries.
- int **samples_per_period** = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED**

  Sets the maximum number of samples to publish in each publication_period.
- final **Duration_t** **publication_period** = new **Duration_t**(1, 0)

  Sets the periodic interval at which samples are published.

8.360.1 Detailed Description

Configures the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish samples in response to a **com.rti.↩ dds.subscription.TopicQuery** (p. 1830).

Enables the ability of a **com.rti.dds.publication.DataWriter** (p. 553) to publish historical samples upon reception of a **com.rti.dds.subscription.TopicQuery** (p. 1830) and how often they are published.

Since a TopicQuery selects previously written samples, the DataWriter must have a **com.rti.dds.infrastructure.↩ DurabilityQosPolicy.kind** (p. 833) different from com.rti.dds.infrastructure.DurabilityQosPolicyKind.DurabilityQos↩ PolicyKind.VOLATILE_DURABILITY_QOS. Also, **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1528) must be set to **com.rti.dds.infrastructure.ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1532).

Only samples that fall within the **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834) for an instance are evaluated against the TopicQuery filter. While the DataWriter is waiting for acknowledgements from one

or more DataReaders, there may temporarily be more than **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834) samples per instance in the DataWriter's queue if the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1147) is set to a higher value than **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834). Those additional samples past **com.rti.dds.infrastructure.DurabilityQosPolicy.writer_depth** (p. 834) are not eligible to be sent in response to the TopicQuery.

A TopicQuery may select multiple samples at once. The writer will publish them periodically, independently from newly written samples. **com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy.publication_period** (p. 1835) configures the frequency of that period and **com.rti.dds.infrastructure.TopicQueryDispatchQosPolicy.samples_per_period** (p. 1834) configures the maximum number of samples to publish each period.

If the DataWriter blocks during the publication of one of these samples, it will stop and try again the next period. (See **com.rti.ndds.example.FooDataWriter.write** (p. 1105) for the conditions that may cause the write operation to block.)

All the DataWriters that belong to a single **com.rti.dds.publication.Publisher** (p. 1466) and enable TopicQueries share the same event thread, but each DataWriter schedules separate events. To configure that thread, see **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.topic_query_publication_thread** (p. 342).

If the DataWriter is dispatching more than one TopicQuery at the same time, the configuration of this periodic event applies to all of them. For example, if a DataWriter receives two TopicQueries around the same time, the period is 1 second, the number of samples per period is 10, the first TopicQuery selects 5 samples, and the second one selects 8, the DataWriter will immediately attempt to publish all 5 for the first TopicQuery and 5 for the second one. After one second, it will publish the remaining 3 samples.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

See also

com.rti.dds.publication.Publisher (p. 1466)

8.360.2 Member Data Documentation

8.360.2.1 enable

```
boolean enable = false
```

Allows this writer to dispatch TopicQueries.

When set to true, this writer can receive and dispatch TopicQueries.

[default] com.rti.dds.infrastructure.false

8.360.2.2 samples_per_period

```
int samples_per_period =  ResourceLimitsQosPolicy.LENGTH_UNLIMITED
```

Sets the maximum number of samples to publish in each publication_period.

[default] com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 259)

[range] [1, 100000000] or **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 259)

8.360.2.3 publication_period

```
final  Duration_t publication_period = new  Duration_t(1, 0)
```

Sets the periodic interval at which samples are published.

[default] 1 second

[range] [0,1 year]

8.361 TopicQueryHelper Class Reference

Helpers to provide utility operations related to **com.rti.dds.subscription.TopicQuery** (p. 1830).

Static Public Member Functions

- static void **topic_query_data_from_service_request** (**TopicQueryData** dst, **ServiceRequest** service↩ Request)

  *Retrieves the **com.rti.dds.subscription.TopicQueryData** (p. 1830) data contained in the specified **com.rti.dds.topic.↩ builtin.ServiceRequest** (p. 1675).*

8.361.1 Detailed Description

Helpers to provide utility operations related to **com.rti.dds.subscription.TopicQuery** (p. 1830).

8.361.2 Member Function Documentation

8.361.2.1 topic_query_data_from_service_request()

```
static void topic_query_data_from_service_request (
              TopicQueryData dst,
              ServiceRequest serviceRequest )  [static]
```

Retrieves the **com.rti.dds.subscription.TopicQueryData** (p. 1830) data contained in the specified **com.rti.dds.↩ topic.builtin.ServiceRequest** (p. 1675).

This operation will extract the content from the request body of the **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) and place it into the specified **com.rti.dds.subscription.TopicQueryData** (p. 1830) object.

The specified **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) must be a valid sample associated with the service id **com.rti.dds.topic.builtin.ServiceRequest.TOPIC_QUERY_SERVICE_ID** (p. 167). Otherwise this operation will return com.rti.dds.infrastructure.false.

This operation can be called within the context of a **com.rti.dds.publication.DataWriterListener::on_service_↩ request_accepted** (p. 596) to retrieve the **com.rti.dds.subscription.TopicQueryData** (p. 1830) of a **com.rti.dds.↩ topic.builtin.ServiceRequest** (p. 1675) that has been received with service id **com.rti.dds.topic.builtin.Service↩ Request.TOPIC_QUERY_SERVICE_ID** (p. 167)

Parameters

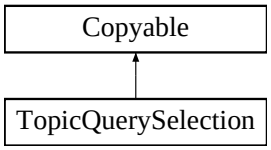
dst	*<<**inout**>> (p. 156)* A **com.rti.dds.subscription.TopicQueryData** (p. 1830) object where the content from the service request is extracted. Cannot be NULL.
serviceRequest	*<<**in**>> (p. 156)* Input **com.rti.dds.topic.builtin.ServiceRequest** (p. 1675) that contains the **com.rti.dds.subscription.TopicQueryData** (p. 1830) as part of its request body. Cannot be NULL.

References **ServiceRequest.request_body**.

8.362 TopicQuerySelection Class Reference

*<<**extension**>> (p. 155)* Specifies the data query that defines a **com.rti.dds.subscription.TopicQuery** (p. 1830)

Inheritance diagram for TopicQuerySelection:



Public Member Functions

- Object **copy_from** (Object src)

Public Attributes

- String **filter_class_name** = null

  The name of the filter to use.
- String **filter_expression** = null

  The filter expression.
- **StringSeq filter_parameters** = new **StringSeq**()

  The query parameters.
- **TopicQuerySelectionKind kind**

  Indicates whether the sample selection is limited to cached samples or not.

8.362.1 Detailed Description

<<***extension***>> *(*p. *155)* Specifies the data query that defines a **com.rti.dds.subscription.TopicQuery** (p. 1830)

The query format is similar to the expression and parameters of a **com.rti.dds.subscription.QueryCondition** (p. 1510) or a **com.rti.dds.topic.ContentFilteredTopic** (p. 436), as described in **com.rti.dds.domain.DomainParticipant.↩ create_contentfilteredtopic_with_filter** (p. 711).

There are two special queries:

- **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 87)
- **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 87)

See also

Queries and Filters Syntax (p. 104)

8.362.2 Member Function Documentation

8.362.2.1 copy_from()

```
Object copy_from (
            Object src )
```

This is the implementation of the `Copyable` interface. This method will perform a deep copy of `src` This method could be placed into `TopicQuerySelectionTypeSupport` rather than here by using the `-noCopyable` option to rtiddsgen.

Parameters

src	The Object which contains the data to be copied.

Returns

Returns `this`.

Exceptions

NullPointerException	If `src` is null.
ClassCastException	If `src` is not the same type as `this`.

See also

com.rti.dds.infrastructure.Copyable::copy_from (p. 445)(java.lang.Object)

Implements **Copyable** (p. 445).

References **StringSeq.copy_from()**, **TopicQuerySelection.filter_class_name**, **TopicQuerySelection.filter_↩ expression**, **TopicQuerySelection.filter_parameters**, and **TopicQuerySelection.kind**.

Referenced by **TopicQueryData.copy_from()**.

8.362.3 Member Data Documentation

8.362.3.1 filter_class_name

```
String filter_class_name = null
```

The name of the filter to use.

Name of content filter to use. Must be one of the built-in filter names or previously registered with **com.rti.dds.domain.↩ DomainParticipant.register_contentfilter** (p. 740) on the same **com.rti.dds.domain.DomainParticipant** (p. 670).

Built-in filter names are **com.rti.dds.domain.DomainParticipant.SQLFILTER_NAME** (p. 50) and **com.rti.dds.↩ domain.DomainParticipant.STRINGMATCHFILTER_NAME** (p. 50).

[default] See **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_USE_READER_CONTENT_↩ FILTER** (p. 87)

Referenced by **TopicQuerySelection.copy_from()**.

8.362.3.2 filter_expression

```
String filter_expression = null
```

The filter expression.

The expression to filter samples in the DataWriters. It follows the format described in **Queries and Filters Syntax** (p. 104).

[default] See **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_USE_READER_CONTENT_↩ FILTER** (p. 87)

See also

Queries and Filters Syntax (p. 104)

Referenced by **TopicQuerySelection.copy_from()**.

8.362.3.3 filter_parameters

```
StringSeq filter_parameters = new  StringSeq()
```

The query parameters.

[default] See **com.rti.dds.subscription.DataReader.TOPIC_QUERY_SELECTION_USE_READER_CONTENT_↩ FILTER** (p. 87)

See also

com.rti.dds.domain.DomainParticipant.create_contentfilteredtopic_with_filter (p. 711).

Referenced by **TopicQuerySelection.copy_from()**.

8.362.3.4 kind

```
TopicQuerySelectionKind kind
```

Initial value:

```
=
            TopicQuerySelectionKind.HISTORY_SNAPSHOT
```

Indicates whether the sample selection is limited to cached samples or not.

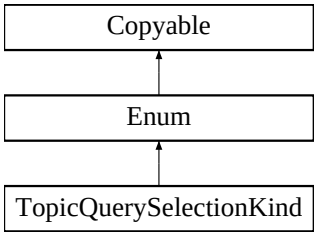
[default] com.rti.dds.subscription.TopicQuerySelectionKind.HISTORY_SNAPSHOT (p. 1840)

Referenced by **TopicQuerySelection.copy_from()**.

8.363 TopicQuerySelectionKind Class Reference

Kinds of **TopicQuerySelection** (p. 1836).

Inheritance diagram for TopicQuerySelectionKind:



Static Public Attributes

- static final **TopicQuerySelectionKind HISTORY_SNAPSHOT**

  *Indicates that the **com.rti.dds.subscription.TopicQuery** (p. 1830) may only select samples that were in the DataWriter cache upon reception.*
- static final **TopicQuerySelectionKind CONTINUOUS**

  *Indicates that the **com.rti.dds.subscription.TopicQuery** (p. 1830) may continue selecting samples published after its reception.*

Additional Inherited Members

8.363.1 Detailed Description

Kinds of **TopicQuerySelection** (p. 1836).

See also

com.rti.dds.subscription.TopicQuerySelection (p. 1836)

8.363.2 Member Data Documentation

8.363.2.1 HISTORY_SNAPSHOT

```
final TopicQuerySelectionKind HISTORY_SNAPSHOT [static]
```

Indicates that the **com.rti.dds.subscription.TopicQuery** (p. 1830) may only select samples that were in the DataWriter cache upon reception.

[default]

8.363.2.2 CONTINUOUS

```
final  TopicQuerySelectionKind CONTINUOUS  [static]
```

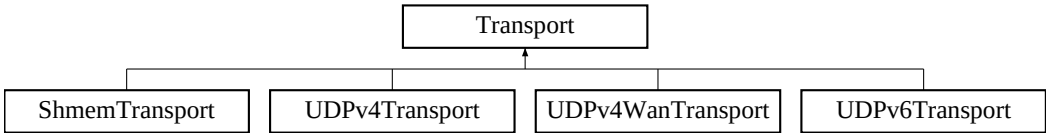
Indicates that the **com.rti.dds.subscription.TopicQuery** (p. 1830) may continue selecting samples published after its reception.

The subscribing application must explicitly delete the **TopicQuery** (p. 1830) (see **com.rti.dds.subscription.DataReader.delete_topic_query** (p. 473)) to signal the DataWriters to stop publishing samples for it.

8.364 Transport Interface Reference

RTI Connext's abstract pluggable transport interface.

Inheritance diagram for Transport:



Classes

- class **Property_t**

  *Base configuration structure that must be inherited by derived **Transport** (p. 1841) Plugin classes.*

8.364.1 Detailed Description

RTI Connext's abstract pluggable transport interface.

8.365 TransportBuiltinKind Class Reference

Built-in transport kind.

Static Public Attributes

- static final int **UDPv4**

  *Built-in UDPv4 transport, **com.rti.ndds.transport.UDPv4Transport** (p. 1943).*
- static final String **UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport: "builtin.udpv4".
- static final int **SHMEM**

  *Built-in shared memory transport, **com.rti.ndds.transport.ShmemTransport** (p. 1681).*
- static final String **SHMEM_ALIAS**

  Alias name for the shared memory built-in transport: "builtin.shmem".
- static final int **UDPv6**

  *Built-in UDPv6 transport, **com.rti.ndds.transport.UDPv6Transport** (p. 1952).*
- static final String **UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport: "builtin.udpv6".
- static final int **UDPv4_WAN**

  *Built-in UDPv4 asymmetric transport, **com.rti.ndds.transport.UDPv4WanTransport** (p. 1948).*
- static final String **UDPv4_WAN_ALIAS**

  Alias name for the UDPv4 asymmetric built-in transport: "builtin.udpv4_wan".
- static final int **MASK_NONE**

  *None of the built-in transports will be registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.*
- static final int **MASK_DEFAULT**

  *The default value of **com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask** (p. 1844).*
- static final int **MASK_ALL**

  *All the available built-in transports are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.*

8.365.1 Detailed Description

Built-in transport kind.

See also

com.rti.dds.infrastructure.TransportBuiltinKindMask

8.365.2 Member Data Documentation

8.365.2.1 UDPv4

```
final int UDPv4  [static]
```

Built-in UDPv4 transport, **com.rti.ndds.transport.UDPv4Transport** (p. 1943).

Referenced by **TransportSupport.get_builtin_transport_property()**, and **TransportSupport.set_builtin_↩ transport_property()**.

8.365.2.2 SHMEM

```
final int SHMEM  [static]
```

Built-in shared memory transport, **com.rti.ndds.transport.ShmemTransport** (p. 1681).

Referenced by **TransportSupport.get_builtin_transport_property()**, and **TransportSupport.set_builtin_↩ transport_property()**.

8.365.2.3 UDPv6

```
final int UDPv6  [static]
```

Built-in UDPv6 transport, **com.rti.ndds.transport.UDPv6Transport** (p. 1952).

Referenced by **TransportSupport.get_builtin_transport_property()**, and **TransportSupport.set_builtin_↩ transport_property()**.

8.365.2.4 UDPv4_WAN

```
final int UDPv4_WAN  [static]
```

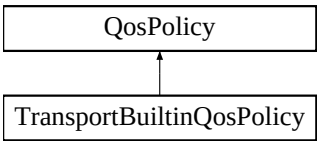
Built-in UDPv4 asymmetric transport, **com.rti.ndds.transport.UDPv4WanTransport** (p. 1948).

Referenced by **TransportSupport.get_builtin_transport_property()**, and **TransportSupport.set_builtin_↩ transport_property()**.

8.366 TransportBuiltinQosPolicy Class Reference

Specifies which built-in transports are used.

Inheritance diagram for TransportBuiltinQosPolicy:



Public Attributes

- int **mask**

  *Specifies the built-in transports that are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.*

8.366.1 Detailed Description

Specifies which built-in transports are used.

Three different transport plug-ins are built into the core RTI Connext libraries (for most supported target platforms): UDPv4, shared memory, and UDPv6.

This QoS policy allows you to control which of these built-in transport plug-ins are used by a **com.rti.dds.domain.↩ DomainParticipant** (p. 670). By default, only the UDPv4 and shared memory plug-ins are enabled (although on some embedded platforms, the shared memory plug-in is not available). In some cases, users will disable the shared memory transport when they do not want applications to use shared memory to communicate when running on the same node.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.366.2 Member Data Documentation

8.366.2.1 mask

```
int mask
```

Specifies the built-in transports that are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.

RTI Connext provides several built-in transports. Only those that are specified with this mask are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 670) is enabled.

[default] com.rti.dds.infrastructure.TransportBuiltinKind.MASK_DEFAULT (p. 271)

8.367 TransportInfo_t Class Reference

Contains the class_id and message_size_max of an installed transport.

Inherits Struct.

Public Member Functions

- **TransportInfo_t** ()

  Constructor with default values.
- **TransportInfo_t** (**TransportInfo_t** src)

  Copy constructor.

Public Attributes

- int **class_id**

  The class_id identifies the transport associated with the message_size_max.
- int **message_size_max**

  The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

8.367.1 Detailed Description

Contains the class_id and message_size_max of an installed transport.

8.367.2 Constructor & Destructor Documentation

8.367.2.1 TransportInfo_t() [1/2]

```
TransportInfo_t ( )
```

Constructor with default values.

8.367.2.2 TransportInfo_t() [2/2]

```
TransportInfo_t (
            TransportInfo_t src )
```

Copy constructor.

References **TransportInfo_t.class_id**, and **TransportInfo_t.message_size_max**.

8.367.3 Member Data Documentation

8.367.3.1 class_id

```
int class_id
```

The class_id identifies the transport associated with the message_size_max.

Referenced by **TransportInfo_t.TransportInfo_t()**.

8.367.3.2 message_size_max

```
int message_size_max
```

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

Referenced by **TransportInfo_t.TransportInfo_t()**.

8.368 TransportInfoSeq Class Reference

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.infrastructure.↩ TransportInfo_t** (p. 1845) > .

Inherits ArraySequence.

8.368.1 Detailed Description

Instantiates com.rti.dds.infrastructure.com.rti.dds.util.Sequence < **com.rti.dds.infrastructure.↩ TransportInfo_t** (p. 1845) > .

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.TransportInfo_t (p. 1845)

8.369 TransportMulticastMapping_t Class Reference

Type representing a list of multicast mapping elements.

Inherits Struct.

Public Member Functions

- **TransportMulticastMapping_t** ()

  Constructor.
- **TransportMulticastMapping_t** (**TransportMulticastMapping_t** src)

  Copy constructor.

Public Attributes

- String **addresses**

  A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive multicast traffic for the entity with a topic that matches the ***com.rti.dds.infrastructure.TransportMulticastMapping_t.topic_expression*** *(*p. 1848*).*
- String **topic_expression**

  A regular expression that will be used to map topic names to corresponding multicast receive addresses.
- **TransportMulticastMappingFunction_t mapping_function**

  Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.

8.369.1 Detailed Description

Type representing a list of multicast mapping elements.

A multicast mapping element specifies a string containing a list of IP addresses, a topic expression and a mapping function.

QoS:

com.rti.dds.infrastructure.TransportMulticastMappingQosPolicy (p. 1851)

8.369.2 Constructor & Destructor Documentation

8.369.2.1 TransportMulticastMapping_t() [1/2]

```
TransportMulticastMapping_t ( )
```

Constructor.

References **TransportMulticastMapping_t.addresses**, **TransportMulticastMapping_t.mapping_function**, and **TransportMulticastMapping_t.topic_expression**.

8.369.2.2 TransportMulticastMapping_t() [2/2]

```
TransportMulticastMapping_t (
            TransportMulticastMapping_t src )
```

Copy constructor.

References **TransportMulticastMapping_t.addresses**, **TransportMulticastMapping_t.mapping_function**, and **TransportMulticastMapping_t.topic_expression**.

8.369.3 Member Data Documentation

8.369.3.1 addresses

```
String addresses
```

A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive *multicast* traffic for the entity with a topic that matches the **com.rti.dds.infrastructure.TransportMulticastMapping_t.topic_expression** (p. 1848).

The string must contain IPv4 or IPv6 addresses separated by commas. For example: "239.255.100.1,239.255.100.↩ 2,239.255.100.3"

You may specify ranges of addresses by enclosing the start address and the end address in square brackets. For example: "[239.255.100.1,239.255.100.3]"

You may combine the two approaches. For example:

"239.255.200.1,[239.255.100.1,239.255.100.3], 239.255.200.3"

IPv4 addresses must be specified in Dot-decimal notation.

IPv6 addresses must be specified using 8 groups of 16-bit hexadecimal values separated by colons. For example: FF00:0000:0000:0000:0202:B3FF:FE1E:8329

Leading zeroes can be skipped. For example: "FF00:0:0:0:202:B3FF:FE1E:8329"

You may replace a consecutive number of zeroes with a double colon, but only once within an address. For example: "FF00::202:B3FF:FE1E:8329"

[default] NULL

Referenced by **TransportMulticastMapping_t.TransportMulticastMapping_t()**.

8.369.3.2 topic_expression

```
String topic_expression
```

A regular expression that will be used to map topic names to corresponding multicast receive addresses.

A topic name must match the expression before a corresponding address is assigned.

[default] NULL

Referenced by **TransportMulticastMapping_t.TransportMulticastMapping_t()**.

8.369.3.3 mapping_function

```
TransportMulticastMappingFunction_t mapping_function
```

Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.

This function is optional. If not specified, the middleware will use a hash function to perform the mapping.

Referenced by **TransportMulticastMapping_t.TransportMulticastMapping_t()**.

8.370 TransportMulticastMappingFunction_t Class Reference

Type representing an external mapping function.

Inherits Struct.

Public Member Functions

- **TransportMulticastMappingFunction_t** ()
  Constructor.
- **TransportMulticastMappingFunction_t** (**TransportMulticastMappingFunction_t** src)
  Copy constructor.

Public Attributes

- String **dll**
  Specifies a dynamic library that contains a mapping function.
- String **function_name**
  Specifies the name of a mapping function.

8.370.1 Detailed Description

Type representing an external mapping function.

A mapping function is defined by a dynamic library name and a function name.

QoS:

com.rti.dds.infrastructure.TransportMulticastMappingQosPolicy (p. 1851)

8.370.2 Constructor & Destructor Documentation

8.370.2.1 TransportMulticastMappingFunction_t() [1/2]

```
TransportMulticastMappingFunction_t ( )
```

Constructor.

References **TransportMulticastMappingFunction_t.dll**, and **TransportMulticastMappingFunction_t.function_↩name**.

8.370.2.2 TransportMulticastMappingFunction_t() [2/2]

```
TransportMulticastMappingFunction_t (
            TransportMulticastMappingFunction_t src )
```

Copy constructor.

References **TransportMulticastMappingFunction_t.dll**, and **TransportMulticastMappingFunction_t.function_↩name**.

8.370.3 Member Data Documentation

8.370.3.1 dll

`String dll`

Specifies a dynamic library that contains a mapping function.

A relative or absolute path can be specified.

If the name is specified as "foo", the library name on Linux systems will be libfoo.so; on Windows systems it will be foo.dll.

[default] NULL

Referenced by **TransportMulticastMappingFunction_t.TransportMulticastMappingFunction_t()**.

8.370.3.2 function_name

`String function_name`

Specifies the name of a mapping function.

This function must be implemented in the library specified in **com.rti.dds.infrastructure.TransportMulticast↩ MappingFunction_t.dll** (p. 1850).

The function must implement the following interface:

int function(const char∗ topic_name, int numberOfAddresses);

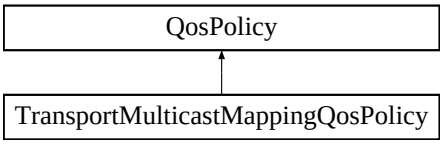
The function must return an integer that indicates the *index* of the address to use for the given `topic_name`. For example, if the first address in the list should be used, it must return 0; if the second address in the list should be used, it must return 1, etc.

Referenced by **TransportMulticastMappingFunction_t.TransportMulticastMappingFunction_t()**.

8.371 TransportMulticastMappingQosPolicy Class Reference

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

Inheritance diagram for TransportMulticastMappingQosPolicy:



Public Attributes

- final **TransportMulticastMappingSeq value**

  A sequence of multicast communications settings.

8.371.1 Detailed Description

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

By default, a **com.rti.dds.publication.DataWriter** (p. 553) will send individually addressed packets for each **com.rti.↩ dds.subscription.DataReader** (p. 450) that subscribes to the topic of the DataWriter – this is known as unicast delivery. Thus, as many copies of the data will be sent over the network as there are DataReaders for the data. The network bandwidth used by a DataWriter will thus increase linearly with the number of DataReaders.

Multicast addressing (on UDP/IP transports) allows multiple DataReaders to receive the *same* network packet. By using multicast, a **com.rti.dds.publication.DataWriter** (p. 553) can send a single network packet that is received by all subscribing applications. Thus the network bandwidth usage will be constant, independent of the number of Data↩ Readers.

Coordinating the multicast address specified by DataReaders can help optimize network bandwidth usage in systems where there are multiple DataReaders for the same **com.rti.dds.topic.Topic** (p. 1807).

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.371.2 Member Data Documentation

8.371.2.1 value

`final` **`TransportMulticastMappingSeq`** `value`

A sequence of multicast communications settings.

An empty sequence means that multicast is not used by the entity.

The RTPS wire protocol currently limits the maximum number of multicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **com.rti.dds.infrastructure.↩ PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipantQos** (p. 795).

[default] Empty sequence.

8.372 TransportMulticastMappingSeq Class Reference

Declares IDL sequence< **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) >

Inherits ArraySequence.

8.372.1 Detailed Description

Declares IDL sequence< **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

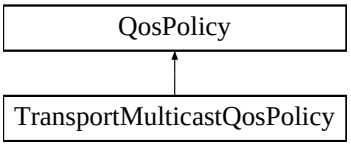
See also

com.rti.dds.infrastructure.TransportMulticastSettings_t (p. 1856)

8.373 TransportMulticastQosPolicy Class Reference

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

Inheritance diagram for TransportMulticastQosPolicy:



Public Attributes

- final **TransportMulticastSettingsSeq value**

  A sequence of multicast communications settings.
- **TransportMulticastQosPolicyKind kind**

  A value that specifies a way to determine how to obtain the multicast address.

8.373.1 Detailed Description

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 450) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **com.rti.dds.domain.DomainParticipant** (p. 670) level) transports with which to receive the multicast data.

By default, a **com.rti.dds.publication.DataWriter** (p. 553) will send individually addressed packets for each **com.rti.dds.subscription.DataReader** (p. 450) that subscribes to the topic of the DataWriter – this is known as unicast delivery. Thus, as many copies of the data will be sent over the network as there are DataReaders for the data. The network bandwidth used by a DataWriter will thus increase linearly with the number of DataReaders.

Multicast addressing (on UDP/IP transports) allows multiple DataReaders to receive the *same* network packet. By using multicast, a **com.rti.dds.publication.DataWriter** (p. 553) can send a single network packet that is received by all subscribing applications. Thus the network bandwidth usage will be constant, independent of the number of DataReaders.

Coordinating the multicast address specified by DataReaders can help optimize network bandwidth usage in systems where there are multiple DataReaders for the same **com.rti.dds.topic.Topic** (p. 1807).

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.373.2 Member Data Documentation

8.373.2.1 value

```
final TransportMulticastSettingsSeq value
```

A sequence of multicast communications settings.

An empty sequence means that multicast is not used by the entity.

The RTPS wire protocol currently limits the maximum number of multicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipantQos** (p. 795).

[default] Empty sequence.

8.373.2.2 kind

```
TransportMulticastQosPolicyKind kind
```

A value that specifies a way to determine how to obtain the multicast address.

This field can be set to one of the following two values: **com.rti.dds.infrastructure.TransportMulticastQosPolicy↩ Kind.AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 273) or **com.rti.dds.infrastructure.TransportMulticast↩ QosPolicyKind.UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** (p. 274).

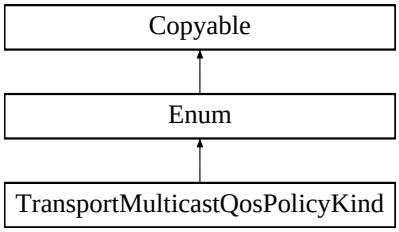
- If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_↩ MULTICAST_QOS** (p. 273), the behavior will depend on the **com.rti.dds.infrastructure.TransportMulticast↩ QosPolicy.value** (p. 1854):
  - If **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1854) does not have any elements, then multicast will not be used.
  - If **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1854) first element has an empty address, then the address will be obtained from **com.rti.dds.infrastructure.TransportMulticastMapping↩ QosPolicy** (p. 1851).
  - If none of the elements in **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1854) is empty, and at least one element has a valid address, then that address will be used.
- If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.UNICAST_ONLY_TRANSPORT↩ _MULTICAST_QOS** (p. 274), then multicast will not be used.

[default] com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_MULTICAST↩ _QOS (p. 273)

8.374 TransportMulticastQosPolicyKind Class Reference

Transport Multicast Policy Kind.

Inheritance diagram for TransportMulticastQosPolicyKind:



Static Public Attributes

- static final **TransportMulticastQosPolicyKind AUTOMATIC_TRANSPORT_MULTICAST_QOS**

  Transport Multicast Policy Kind.
- static final **TransportMulticastQosPolicyKind UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** = new **TransportMulticastQosPolicyKind**("UNICAST_ONLY_TRANSPORT_MULTICAST_QOS", 1)

  Transport Multicast Policy Kind.

Additional Inherited Members

8.374.1 Detailed Description

Transport Multicast Policy Kind.

See also

com.rti.dds.infrastructure.TransportMulticastQosPolicy (p. 1853)

8.375 TransportMulticastSettings_t Class Reference

Type representing a list of multicast locators.

Inherits Struct.

Public Member Functions

- **TransportMulticastSettings_t** ()
  Constructor with default values.
- **TransportMulticastSettings_t** (**TransportMulticastSettings_t** src)
  Copy constructor.

Public Attributes

- final **StringSeq transports**
  A sequence of transport aliases that specifies the transports on which to receive multicast traffic for the entity.
- InetAddress **receive_address**
  The multicast group address on which the entity can receive data.
- int **receive_port**
  The multicast port on which the entity can receive data.

8.375.1 Detailed Description

Type representing a list of multicast locators.

A multicast locator specifies a transport class, a multicast address, and a multicast port number on which messages can be received by an entity.

QoS:

com.rti.dds.infrastructure.TransportMulticastQosPolicy (p. 1853)

8.375.2 Constructor & Destructor Documentation

8.375.2.1 TransportMulticastSettings_t() [1/2]

```
TransportMulticastSettings_t ( )
```

Constructor with default values.

8.375.2.2 TransportMulticastSettings_t() [2/2]

```
TransportMulticastSettings_t (
            TransportMulticastSettings_t src )
```

Copy constructor.

References **TransportMulticastSettings_t.receive_address**, **TransportMulticastSettings_t.receive_port**, and **TransportMulticastSettings_t.transports**.

8.375.3 Member Data Documentation

8.375.3.1 transports

```
final  StringSeq transports
```

A sequence of transport aliases that specifies the transports on which to receive *multicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias in this sequence are used to subscribe to the multicast group addresses. Thus, this list of aliases sub-selects from the transport s available to the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 269).

[default] Empty sequence; i.e. all the transports available to the entity.

[range] Any sequence of non-null, non-empty strings.

Referenced by **TransportMulticastSettings_t.TransportMulticastSettings_t()**.

8.375.3.2 receive_address

```
InetAddress receive_address
```

The multicast group address on which the entity can receive data.

Must must be an address in the proper format (see **Address Format** (p. 224)).

[default] NONE/INVALID. Required to specify a multicast group address to join.

[range] A valid IPv4 or IPv6 multicast address.

See also

Address Format (p. 224)

Referenced by **TransportMulticastSettings_t.TransportMulticastSettings_t()**.

8.375.3.3 receive_port

```
int receive_port
```

The multicast port on which the entity can receive data.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id` (see **com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id** (p. 1990)).

[range] [0,0xffffffff]

Referenced by **TransportMulticastSettings_t.TransportMulticastSettings_t()**.

8.376 TransportMulticastSettingsSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) `>`

Inherits ArraySequence.

8.376.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1856) `>`

Instantiates:

<<***generic***>> *(p. 156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

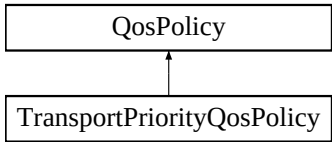
See also

com.rti.dds.infrastructure.TransportMulticastSettings_t (p. 1856)

8.377 TransportPriorityQosPolicy Class Reference

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Inheritance diagram for TransportPriorityQosPolicy:



Public Attributes

- int **value**

  This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.

8.377.1 Detailed Description

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

The Transport Priority QoS policy is optional and only supported on certain OSs and transports. It allows you to specify on a per-DataWriter or DataReader basis that the data sent by that endpoint is of a different priority.

The DDS specification does not indicate how a DDS implementation should treat data of different priorities. It is often difficult or impossible for DDS implementations to treat data of higher priority differently than data of lower priority, especially when data is being sent (delivered to a physical transport) directly by the thread that called **com.rti.ndds.↩ example.FooDataWriter.write** (p. 1105). Also, many physical network transports themselves do not have a end-user controllable level of data packet priority.

Entity:

com.rti.dds.publication.DataWriter (p. 553), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.↩ topic.Topic** (p. 1807)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.377.2 Usage

In RTI Connext, for the IP-based transports (UDPv4, UDPv6, Real-Time WAN, and TCP), the value set in the Transport Priority QoS policy can be used to set the differentiated services field (DS field) bits of the IPv4 and IPv6 headers for datagrams sent by a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). It is platform-dependent on how and whether setting the DS field has an effect. Some platforms may require external permissions in order to set the DS field.

The transport priority value is not provided as is to the transports, but transformed according to the following fields: transport_priority_mask (for example, see **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mask** (p. 1415)), transport_priority_mapping_low (for example, see **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_low** (p. 1415)), and transport_priority_mapping_high (for example, see **com.rti.ndds.transport.UDPv4Transport.Property_t.transport_priority_mapping_high** (p. 1415)). If you want the priority value to be exactly equal to the DS value, then the only change you need to make is to set transport_priority_mask to `0xff` (and keep the transport_priority_mapping_low and transport_priority_mapping_high defaults).

It is incorrect to assume that using the Transport Priority QoS policy will have any effect at all on the end-to-end delivery of data from a **com.rti.dds.publication.DataWriter** (p. 553) and a **com.rti.dds.subscription.DataReader** (p. 450). All network elements, including switches and routers, must have the capability and be enabled to actually use the DS field to treat higher priority packets differently. Thus the ability to use the Transport Priority QoS policy must be designed and configured at a *system* level; just turning it on in an application may have no effect at all at a transport level.

For additional details on how to set the DS field in IP-based transports, see the "TRANSPORT_PRIORITY QosPolicy" section of the `User's Manual`.

8.377.3 Member Data Documentation

8.377.3.1 value

```
int value
```

This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.

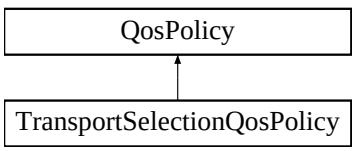
You may choose any value within the range of a 32-bit signed integer; higher values indicate higher priority. However, any further interpretation of this policy is specific to a particular transport and a particular DDS implementation. For example, a particular transport is permitted to treat a range of priority values as equivalent to one another.

[default] 0

8.378 TransportSelectionQosPolicy Class Reference

Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) may use to send or receive data.

Inheritance diagram for TransportSelectionQosPolicy:



Public Attributes

- final **StringSeq enabled_transports**

  A sequence of transport aliases that specifies the transport instances available for use by the entity.

8.378.1 Detailed Description

Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.Data↩ Reader** (p. 450) may use to send or receive data.

An application may be simultaneously connected to many different physical transports, e.g., Ethernet, Infiniband, shared memory, VME backplane, and wireless. By default, RTI Connext will use up to 16 transports to deliver data from a DataWriter to a DataReader.

This QoS policy can be used to both limit and control which of the application's available transports may be used by a **com.rti.dds.publication.DataWriter** (p. 553) to send data or by a **com.rti.dds.subscription.DataReader** (p. 450) to receive data.

Entity:

com.rti.dds.subscription.DataReader (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.378.2 Member Data Documentation

8.378.2.1 enabled_transports

```
final  StringSeq enabled_transports
```

A sequence of transport aliases that specifies the transport instances available for use by the entity.

Of the transport instances installed with the **com.rti.dds.domain.DomainParticipant** (p. 670), only those with aliases matching an alias in this sequence are available to the entity.

Thus, this list of aliases sub-selects from the transports available to the **com.rti.dds.domain.DomainParticipant** (p. 670).

An empty sequence is a special value that specifies all the transports installed with the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 269). These alias names are case sensitive and should be written in lowercase.

[default] Empty sequence; i.e. all the transports installed with and available to the **com.rti.dds.domain.Domain↩ Participant** (p. 670).

[range] A sequence of non-null, non-empty strings.

See also

com.rti.dds.domain.DomainParticipantQos.transport_builtin (p. 800).

8.379 TransportSupport Class Reference

<<***interface***>> *(p. 156)* The utility class used to configure RTI Connext pluggable transports.

Static Public Member Functions

- static void **get_builtin_transport_property** (**DomainParticipant** participant_in, Transport.Property_t builtin_↩ transport_property_inout)

  Get the properties used to create a builtin transport plugin.
- static void **set_builtin_transport_property** (**DomainParticipant** participant_in, Transport.Property_t builtin_↩ transport_property_in)

  Set the properties used to create a builtin transport plugin.

8.379.1 Detailed Description

<<***interface***>> *(p. 156)* The utility class used to configure RTI Connext pluggable transports.

8.379.2 Member Function Documentation

8.379.2.1 get_builtin_transport_property()

```
static void get_builtin_transport_property (
              DomainParticipant participant_in,
             Transport.Property_t builtin_transport_property_inout )  [static]
```

Get the properties used to create a builtin transport plugin.

Retrieves the properties that will be used to create a builtin transport plugin.

Precondition

The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_↩ _transport_kind_in`.

Parameters

participant_↩ _in	<<***in***>> *(p. 156)* A valid non-null **com.rti.dds.domain.DomainParticipant** (p. 670)

Parameters

builtin_transport_property_inout	<<***inout***>> *(*p. *156)* The storage area where the retrieved property will be output. The specific type required by the `builtin_transport_kind_in` must be used.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

References **TransportBuiltinKind.SHMEM**, **TransportBuiltinKind.UDPv4**, **TransportBuiltinKind.UDPv4_WAN**, and **TransportBuiltinKind.UDPv6**.

8.379.2.2 set_builtin_transport_property()

```
static void set_builtin_transport_property (
              DomainParticipant participant_in,
             Transport.Property_t builtin_transport_property_in )  [static]
```

Set the properties used to create a builtin transport plugin.

Specifies the properties that will be used to create a builtin transport plugin.

If the builtin transport is already registered when this operation is called, these property changes will *not* have any effect. Builtin transport properties should always be set before the transport is registered. See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

Precondition

A disabled **com.rti.dds.domain.DomainParticipant** (p. 670). The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_transport_kind_in`.

Parameters

participant_in	<<***in***>> *(*p. *156)* A valid non-null **com.rti.dds.domain.DomainParticipant** (p. 670) that has not been enabled.

Parameters

builtin_transport_property_in	<<***inout***>> *(*p. *156)* The new transport property that will be used to the create the builtin transport plugin. The specific type required by the `builtin_transport_kind_in` must be used.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

See also

com.rti.ndds.transport.TransportSupport.get_builtin_transport_property() (p. 1862)

References **TransportBuiltinKind.SHMEM**, **TransportBuiltinKind.UDPv4**, **TransportBuiltinKind.UDPv4_WAN**, and **TransportBuiltinKind.UDPv6**.

8.380 TransportUdpWanCommPortsMappingInfo_t Class Reference

Type for storing UDP WAN communication ports.

Inherits Struct.

Public Member Functions

- **TransportUdpWanCommPortsMappingInfo_t** ()
  Constructor with default values.
- **TransportUdpWanCommPortsMappingInfo_t** (**TransportUdpWanCommPortsMappingInfo_t** src)
  Copy constructor.

Public Attributes

- int **rtps_port**
  RTPS port.
- int **host_port**
  Host port.
- int **public_port**
  Public port.

8.380.1 Detailed Description

Type for storing UDP WAN communication ports.

8.380.2 Constructor & Destructor Documentation

8.380.2.1 TransportUdpWanCommPortsMappingInfo_t() [1/2]

```
TransportUdpWanCommPortsMappingInfo_t ( )
```

Constructor with default values.

8.380.2.2 TransportUdpWanCommPortsMappingInfo_t() [2/2]

```
TransportUdpWanCommPortsMappingInfo_t (
            TransportUdpWanCommPortsMappingInfo_t src )
```

Copy constructor.

References **TransportUdpWanCommPortsMappingInfo_t.host_port**, **TransportUdpWanCommPortsMapping↩Info_t.public_port**, and **TransportUdpWanCommPortsMappingInfo_t.rtps_port**.

8.380.3 Member Data Documentation

8.380.3.1 rtps_port

```
int rtps_port
```

RTPS port.

Referenced by **TransportUdpWanCommPortsMappingInfo_t.TransportUdpWanCommPortsMappingInfo_t()**.

8.380.3.2 host_port

```
int host_port
```

Host port.

Referenced by **TransportUdpWanCommPortsMappingInfo_t.TransportUdpWanCommPortsMappingInfo_t()**.

8.380.3.3 public_port

```
int public_port
```

Public port.

Referenced by **TransportUdpWanCommPortsMappingInfo_t.TransportUdpWanCommPortsMappingInfo_t()**.

8.381 TransportUdpWanCommPortsMappingInfoSeq Class Reference

List of < **com.rti.dds.infrastructure.TransportUdpWanCommPortsMappingInfo_t** (p. 1864) > .

Inherits ArraySequence.

Public Member Functions

- void **push_to_nativel** (long native_sequence, long native_length)

8.381.1 Detailed Description

List of < **com.rti.dds.infrastructure.TransportUdpWanCommPortsMappingInfo_t** (p. 1864) > .

See also

com.rti.dds.infrastructure.TransportUdpWanCommPortsMappingInfo_t (p. 1864)

8.381.2 Member Function Documentation

8.381.2.1 push_to_nativel()

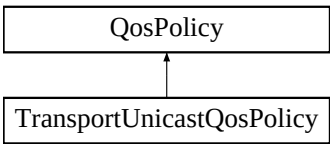
```
void push_to_nativeI (
             long native_sequence,
             long native_length )
```

The native method will return NULL if the given index is out of range. That should only happen if there's an NDDS bug, so throw an IllegalStateException instead of a **RETCODE_ERROR** (p. 1595) exception.

8.382 TransportUnicastQosPolicy Class Reference

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 1029) to receive data.

Inheritance diagram for TransportUnicastQosPolicy:



Public Attributes

- final **TransportUnicastSettingsSeq value**
  A sequence of unicast communication settings.

8.382.1 Detailed Description

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 1029) to receive data.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.382.2 Usage

RTI Connext may send data to a variety of Entities, not just DataReaders. For example, reliable DataWriters may receive ACK/NACK packets from reliable DataReaders.

During discovery, each **com.rti.dds.infrastructure.Entity** (p. 1029) announces to remote applications a list of (up to 16) unicast addresses to which the remote application should send data (either user data packets or reliable protocol metadata such as ACK/NACKs and heartbeats). Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipantQos** (p. 795).

By default, the list of addresses is populated automatically with values obtained from the enabled transport plug-ins allowed to be used by the **Entity** (p. 1029) (see **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843) and **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1860)). Also, the associated ports are automatically determined (see **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622)).

Use this QoS policy to manually set the receive address list for an **Entity** (p. 1029). You may optionally set a port to use a non-default receive port as well. Only the first 16 addresses will be used.

RTI Connext will create a receive thread for every unique port number that it encounters (on a per transport basis).

- For a **com.rti.dds.domain.DomainParticipant** (p. 670), this QoS policy sets the default list of addresses used by other applications to send user data for local DataReaders.

- For a **com.rti.dds.subscription.DataReader** (p. 450), if set, then other applications will use the specified list of addresses to send user data (and reliable protocol packets for reliable DataReaders). Otherwise, if not set, the other applications will use the addresses set by the **com.rti.dds.domain.DomainParticipant** (p. 670).

- For a reliable **com.rti.dds.publication.DataWriter** (p. 553), if set, then other applications will use the specified list of addresses to send reliable protocol packets (ACKS/NACKS) on the behalf of reliable DataReaders. Otherwise, if not set, the other applications will use the addresses set by the **com.rti.dds.domain.DomainParticipant** (p. 670).

8.382.3 Member Data Documentation

8.382.3.1 value

`final` **`TransportUnicastSettingsSeq`** `value`

A sequence of unicast communication settings.

An empty sequence means that applicable defaults specified by elsewhere (e.g. **com.rti.dds.domain.Domain↩ ParticipantQos.default_unicast** (p. 800)) should be used.

The RTPS wire protocol currently limits the maximum number of unicast locators to

1. Note that this is a hard limit that cannot be increased. However, this limit can be decreased by configuring the 'dds.domain_participant.max_announced_locator_list_size' property in the **com.rti.dds.infrastructure.↩ PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipantQos** (p. 795).

[default] Empty sequence.

See also

com.rti.dds.domain.DomainParticipantQos.default_unicast (p. 800)

8.383 TransportUnicastSettings_t Class Reference

Type representing a list of unicast locators.

Inherits Struct.

Public Member Functions

- **TransportUnicastSettings_t** ()

  Constructor.
- **TransportUnicastSettings_t** (**TransportUnicastSettings_t** src)

  Copy constructor.

Public Attributes

- final **StringSeq transports**

  A sequence of transport aliases that specifies the unicast interfaces on which to receive unicast traffic for the entity.
- int **receive_port**

  The unicast port on which the entity can receive data.

8.383.1 Detailed Description

Type representing a list of unicast locators.

A unicast locator specifies a transport class, a unicast address, and a unicast port number on which messages can be received by an entity.

QoS:

com.rti.dds.infrastructure.TransportUnicastQosPolicy (p. 1867)

8.383.2 Constructor & Destructor Documentation

8.383.2.1 TransportUnicastSettings_t() [1/2]

```
TransportUnicastSettings_t ( )
```

Constructor.

8.383.2.2 TransportUnicastSettings_t() [2/2]

```
TransportUnicastSettings_t (
            TransportUnicastSettings_t src )
```

Copy constructor.

References **TransportUnicastSettings_t.receive_port**, and **TransportUnicastSettings_t.transports**.

8.383.3 Member Data Documentation

8.383.3.1 transports

```
final StringSeq transports
```

A sequence of transport aliases that specifies the unicast interfaces on which to receive *unicast* traffic for the entity.

Of the transport instances available to the entity, only those with aliases matching an alias on this sequence are used to determine the unicast interfaces used by the entity.

Thus, this list of aliases sub-selects from the transports available to the entity.

Each unicast interface on a transport results in a unicast locator for the entity.

An empty sequence is a special value that specifies all the transports available to the entity.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 269).

[default] Empty sequence; i.e. all the transports available to the entity.
[range] Any sequence of non-null, non-empty strings.

Referenced by **TransportUnicastSettings_t.TransportUnicastSettings_t()**.

8.383.3.2 receive_port

```
int receive_port
```

The unicast port on which the entity can receive data.

Must be an *unused* unicast port on the system.

[default] 0, which implies that the actual port number is determined by a formula as a function of the `domain_id`, and the **com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id** (p. 1990).

[range] [0,0xffffffff]

See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id (p. 1990).

Referenced by **TransportUnicastSettings_t.TransportUnicastSettings_t()**.

8.384 TransportUnicastSettingsSeq Class Reference

Declares IDL sequence< **com.rti.dds.infrastructure.TransportUnicastSettings_t** (p. 1868) >

Inherits ArraySequence.

8.384.1 Detailed Description

Declares IDL sequence< **com.rti.dds.infrastructure.TransportUnicastSettings_t** (p. 1868) >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.TransportUnicastSettings_t (p. 1868)

8.385 TrustAlgorithmRequirements Class Reference

Type to describe Trust Plugins algorithm requirements for an entity.

Inherits Struct.

Public Member Functions

- **TrustAlgorithmRequirements** ()
  Create an instance with the default Trust Algorithm Requirements.
- **TrustAlgorithmRequirements** (int **supported_mask**, int **required_mask**)
  Create an instance with the Trust Algorithm Requirements passed as input.

Public Attributes

- int **supported_mask**
  Trust Plugins algorithms that an entity supports.
- int **required_mask**
  Trust Plugins algorithms that an entity uses.

8.385.1 Detailed Description

Type to describe Trust Plugins algorithm requirements for an entity.

8.385.2 Constructor & Destructor Documentation

8.385.2.1 TrustAlgorithmRequirements() [1/2]

```
TrustAlgorithmRequirements ( )
```

Create an instance with the default Trust Algorithm Requirements.

The meaning of this field may vary depending on what Trust Plugins the entity is using.

8.385.2.2 TrustAlgorithmRequirements() [2/2]

```
TrustAlgorithmRequirements (
            int supported_mask,
            int required_mask )
```

Create an instance with the Trust Algorithm Requirements passed as input.

The meaning of this field may vary depending on what Trust Plugins the entity is using.

References **TrustAlgorithmRequirements.required_mask**, and **TrustAlgorithmRequirements.supported_mask**.

8.385.3 Member Data Documentation

8.385.3.1 supported_mask

```
int supported_mask
```

Trust Plugins algorithms that an entity supports.

Referenced by **TrustAlgorithmRequirements.TrustAlgorithmRequirements()**.

8.385.3.2 required_mask

```
int required_mask
```

Trust Plugins algorithms that an entity uses.

Referenced by **TrustAlgorithmRequirements.TrustAlgorithmRequirements()**.

8.386 TypeCode Class Reference

The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.

Inherits Serializable.

Public Member Functions

- **TCKind kind** ()

  *Gets the **com.rti.dds.typecode.TCKind** (p. 1786) value of a type code.*
- **ExtensibilityKind extensibility_kind** ()

  *Gets the **com.rti.dds.typecode.ExtensibilityKind** (p. 1047) value of a type code.*
- boolean **equal** (**TypeCode** tc)

  *Compares two **com.rti.dds.typecode.TypeCode** (p. 1873) objects for equality.*
- boolean **equals** (Object tc)

  *Compares two **com.rti.dds.typecode.TypeCode** (p. 1873) objects for equality.*
- boolean **assignable** (**TypeCode** from)

  *Checks if this **com.rti.dds.typecode.TypeCode** (p. 1873) is assignable from the input **com.rti.dds.typecode.TypeCode** (p. 1873).*
- int **length** () throws BadKind

  Returns the number of elements in the type described by this type code.
- String **name** () throws BadKind

  *Retrieves the simple name identifying this **com.rti.dds.typecode.TypeCode** (p. 1873) object within its enclosing scope.*
- boolean **is_alias_pointer** () throws BadKind

  Function that tells if an alias is a pointer or not.
- short **type_modifier** () throws BadKind

  *Returns a constant indicating the modifier of the value type that this **com.rti.dds.typecode.TypeCode** (p. 1873) object describes.*
- **TypeCode concrete_base_type** () throws BadKind

  *Returns the **com.rti.dds.typecode.TypeCode** (p. 1873) that describes the concrete base type of the value type that this **com.rti.dds.typecode.TypeCode** (p. 1873) object describes.*
- **TypeCode content_type** () throws BadKind

  *Returns the **com.rti.dds.typecode.TypeCode** (p. 1873) object representing the type for the members of the object described by this **com.rti.dds.typecode.TypeCode** (p. 1873) object.*
- int **array_dimension_count** () throws BadKind

  This function returns the number of dimensions of an array type code.
- int **array_dimension** (int index) throws BadKind,Bounds

  This function returns the index-th dimension of an array type code.
- int **element_count** () throws BadKind

  The number of elements in an array.
- int **member_count** () throws BadKind

  Returns the number of members of the type code.
- String **member_name** (int index) throws BadKind,Bounds

  Returns the name of a type code member identified by the given index.
- **TypeCode member_type** (int index) throws BadKind,Bounds

*Retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the type of the member identified by the given index.*

- int **member_id** (int index) throws BadKind,Bounds

  *Returns the ID of the **TypeCode** (p. 1873) member identified by the given index.*
- int **member_label_count** (int index) throws BadKind,Bounds

  Returns the number of labels associated to the index-th union member.
- int **member_label** (int member_index, int label_index) throws BadKind,Bounds

  Return the label_index-th label associated to the member_index-th member.
- int **member_ordinal** (int index) throws BadKind,Bounds

  Returns the ordinal that corresponds to the index-th enum value.
- boolean **is_member_key** (int index) throws BadKind,Bounds

  Function that tells if a member is a key or not.
- boolean **is_member_required** (int index) throws BadKind,Bounds

  Indicates whether a given member of a type is required to be present in every sample of that type.
- boolean **is_member_pointer** (int index) throws BadKind,Bounds

  Function that tells if a member is a pointer or not.
- boolean **is_member_bitfield** (int index) throws BadKind,Bounds

  Function that tells if a member is a bitfield or not.
- short **member_bitfield_bits** (int index) throws BadKind,Bounds

  Returns the number of bits of a bitfield member.
- short **member_visibility** (int index) throws BadKind,Bounds

  Returns the constant that indicates the visibility of the index-th member.
- **TypeCode** **discriminator_type** () throws BadKind

  Returns the discriminator type code.
- int **default_index** () throws BadKind

  Returns the index of the default member, or -1 if there is no default member.
- int **find_member_by_id** (int id) throws BadKind

  Get the index of the member of the given ID.
- int **find_member_by_name** (String **name**) throws BadKind

  Get the index of the member of the given name.
- void **print_IDL** (int indent)

  *Prints a **com.rti.dds.typecode.TypeCode** (p. 1873) in IDL notation.*
- void **print_IDL** (int indent, Writer out) throws IOException

  *Prints a **com.rti.dds.typecode.TypeCode** (p. 1873) in a pseudo-IDL notation.*
- void **print_complete_IDL** (Writer writer) throws IOException
- int **add_member** (String **name**, int id, **TypeCode** tc, byte member_flags) throws BadKind,BadMemberName,↩ BadMemberId

  *Add a new member to this **com.rti.dds.typecode.TypeCode** (p. 1873).*
- synchronized int **add_member** (String **name**, int id, **TypeCode** tc, byte member_flags, short visibility, boolean is_pointer, short bits) throws BadKind,BadMemberName,BadMemberId,BAD_PARAM,IllegalStateException

  *Add a new member to this **com.rti.dds.typecode.TypeCode** (p. 1873).*
- synchronized int **add_member_to_enum** (String **name**, int ordinal) throws BadKind,BadMemberName

  *Add a new enumerated constant to this enum **com.rti.dds.typecode.TypeCode** (p. 1873).*
- synchronized int **add_member_to_union** (String **name**, int id, int labels_in[ ], **TypeCode** tc, boolean is_pointer) throws BadKind,BadMemberName, BadMemberId

  *Add a new member to a union **com.rti.dds.typecode.TypeCode** (p. 1873).*
- String **signature** () throws RETCODE_ERROR

  *Gets an MD5 signature of the **com.rti.dds.typecode.TypeCode** (p. 1873).*

- final long **cdr_serialized_sample_min_size** (short representation_id) throws UserException, SystemException

  Gets the minimum serialized size of samples of this type.
- final long **cdr_serialized_sample_min_size** () throws UserException, SystemException

  Gets the minimum serialized size of samples of this type.
- final long **cdr_serialized_sample_max_size** (short representation_id) throws UserException, SystemException

  Gets the maximum serialized size of samples of this type.
- final long **cdr_serialized_sample_max_size** () throws UserException, SystemException

  Gets the maximum serialized size of samples of this type.
- final long **cdr_serialized_sample_key_max_size** (short representation_id) throws UserException, System↩ Exception

  Gets the maximum serialized size of sample keys of this type.
- final long **cdr_serialized_sample_key_max_size** () throws UserException, SystemException

  Gets the maximum serialized size of sample keys of this type.
- long **get_type_object_serialized_size** ()

  Gets the serialized size of the TypeObject created from this ***com.rti.dds.typecode.TypeCode*** *(*p. *1873).*

Static Public Attributes

- static final **TypeCode TC_NULL**

  Basic null type.
- static final **TypeCode TC_SHORT**

  Basic 16-bit signed integer type.
- static final **TypeCode TC_LONG**

  Basic 32-bit signed integer type.
- static final **TypeCode TC_USHORT**

  Basic unsigned 16-bit integer type.
- static final **TypeCode TC_ULONG**

  Basic unsigned 32-bit integer type.
- static final **TypeCode TC_FLOAT**

  Basic 32-bit floating point type.
- static final **TypeCode TC_DOUBLE**

  Basic 64-bit floating point type.
- static final **TypeCode TC_BOOLEAN**

  Basic Boolean type.
- static final **TypeCode TC_CHAR**

  Basic single-byte character type.
- static final **TypeCode TC_OCTET**

  Basic octet/byte type.
- static final **TypeCode TC_LONGLONG**

  Basic 64-bit integer type.
- static final **TypeCode TC_ULONGLONG**

  Basic unsigned 64-bit integer type.
- static final **TypeCode TC_LONGDOUBLE**

  Basic 128-bit floating point type.
- static final **TypeCode TC_WCHAR**

  Basic four-byte character type.

- static final int **MEMBER_ID_INVALID**

  *A sentinel indicating an invalid **com.rti.dds.typecode.TypeCode** (p. 1873) member ID.*
- static final int **MAX_MEMBER_ID**

  Maximum value for member id. ID.
- static final int **INDEX_INVALID**

  *A sentinel indicating an invalid **com.rti.dds.typecode.TypeCode** (p. 1873) member index.*
- static final byte **NONKEY_MEMBER**

  A flag indicating that a type member is optional and not part of the key.
- static final byte **KEY_MEMBER**

  A flag indicating that a type member is part of the key for that type, and therefore required.
- static final byte **NONKEY_REQUIRED_MEMBER**

  A flag indicating that a type member is not part of the key but is nevertheless required.
- static final short **NOT_BITFIELD**

  Indicates that a member of a type is not a bitfield.

8.386.1 Detailed Description

The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with rtiddsgen (see the `Code Generator User's Manual`) or to modify types you define yourself at runtime.

You create **com.rti.dds.typecode.TypeCode** (p. 1873) objects using the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) singleton. Then you can use the methods on *this* class to inspect and modify the data type definition.

This class is based on a similar class from CORBA.

MT Safety:

SAFE for read-only access, UNSAFE for modification. Modifying a single **com.rti.dds.typecode.TypeCode** (p. 1873) object concurrently from multiple threads is *unsafe*. Modifying a **com.rti.dds.typecode.TypeCode** (p. 1873) from a single thread while concurrently reading the state of that **com.rti.dds.typecode.TypeCode** (p. 1873) from another thread is also *unsafe*. However, reading the state of a **com.rti.dds.typecode.TypeCode** (p. 1873) concurrently from multiple threads, without any modification, is *safe*.

See also

`http://java.sun.com/javase/6/docs/api/org/omg/CORBA/TypeCode.html`

8.386.2 Member Function Documentation

8.386.2.1 kind()

```
TCKind kind ( )
```

Gets the **com.rti.dds.typecode.TCKind** (p. 1786) value of a type code.

Retrieves the kind of this **com.rti.dds.typecode.TypeCode** (p. 1873) object. The kind of a type code determines which **com.rti.dds.typecode.TypeCode** (p. 1873) methods may legally be invoked on it.

MT Safety:

SAFE.

Returns

The type code kind.

Referenced by **TypeCode.concrete_base_type()**, **TypeCode.equal()**, **DynamicData.get_char()**, **DynamicData.get_char_array()**, **DynamicData.get_char_seq()**, **DynamicData.get_string()**, **DynamicData.set_char()**, **DynamicData.set_char_array()**, **DynamicData.set_char_seq()**, and **DynamicData.set_string()**.

8.386.2.2 extensibility_kind()

```
ExtensibilityKind extensibility_kind ( )
```

Gets the **com.rti.dds.typecode.ExtensibilityKind** (p. 1047) value of a type code.

Retrieves the extensibility kind of this **com.rti.dds.typecode.TypeCode** (p. 1873) object.

In some cases, it is desirable for types to evolve without breaking interoperability with deployed components already using those types. For example:

- A new set of applications to be integrated into an existing system may want to introduce additional fields into a structure. These new fields can be safely ignored by already deployed applications, but applications that do understand the new fields can benefit from their presence.
- A new set of applications to be integrated into an existing system may want to increase the maximum size of some sequence or string in a Type. Existing applications can receive data samples from these new applications as long as the actual number of elements (or length of the strings) in the received data sample does not exceed what the receiving applications expects. If a received data sample exceeds the limits expected by the receiving application, then the sample can be safely ignored (filtered out) by the receiver.

In order to support use cases such as these, the type system introduces the concept of appendable and mutable types.

- A type may be final, indicating that the range of its possible data values is strictly defined. In particular, it is not possible to add elements to members of collection or aggregated types while maintaining type assignability.

- A type may be appendable, indicating that two types, where one contains all of the elements/members of the other plus additional elements/members appended to the end, may remain assignable.
- A type may be mutable, indicating that two types may differ from one another in the additional, removal, and/or transposition of elements/members while remaining assignable.

The extensibility of **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), and **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) can be change using the built-in "Extensibility" annotation when the type is declared.

IDL examples:

```
@final

struct MyType {

    long member_1;

}
@appendable

struct MyType {

    long member_1;

}
@mutable

struct MyType {

    long member_1;

}
```

XML example:

```
<struct name="MyType" extensibility="final">

    <member name="member_1" type="long"/>

</struct>
<struct name="MyType" extensibility="appendable">

    <member name="member_1" type="long"/>

</struct>
<struct name="MyType" extensibility="mutable">

    <member name="member_1" type="long"/>

</struct>
```

XSD example:

```
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>

<!-- @struct true -->

<!-- @extensibility FINAL_EXTENSIBILITY -->
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>
```

```
<!-- @struct true -->

<!-- @extensibility EXTENSIBLE_EXTENSIBILITY -->
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

    </xsd:sequence>

</xsd:complexType>

<!-- @struct true -->

<!-- @extensibility MUTABLE_EXTENSIBILITY -->
```

For TypeCodes built at run-time using the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) API, the extensibility can be provided as a parameter of the following APIs:

- **com.rti.dds.typecode.TypeCodeFactory.create_struct_tc** (p. 1923)
- **com.rti.dds.typecode.TypeCodeFactory.create_value_tc** (p. 1925)
- **com.rti.dds.typecode.TypeCodeFactory.create_union_tc** (p. 1927)
- **com.rti.dds.typecode.TypeCodeFactory.create_enum_tc** (p. 1928)

See also

com.rti.dds.typecode.ExtensibilityKind (p. 1047)

MT Safety:

SAFE.

Returns

The type code extensibility kind.

Referenced by **TypeCode.equal()**.

8.386.2.3 equal()

```
boolean equal (
               TypeCode tc )
```

Compares two **com.rti.dds.typecode.TypeCode** (p. 1873) objects for equality.

MT Safety:

SAFE.

For equality and assignability purposes, **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) are considered equivalent.

The **com.rti.dds.typecode.TypeCode** (p. 1873) of structs inheriting from other structs has a **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) kind.

For example:

```
struct MyStruct: MyBaseStruct {

    long member_1;

};
```

The code generation for the previous type will generate a **com.rti.dds.typecode.TypeCode** (p. 1873) with **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) kind.

Parameters

tc	<<***in***>> *(p. 156)* Type code that will be compared with this **com.rti.dds.typecode.TypeCode** (p. 1873).

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM	if tc is null.

Returns

com.rti.dds.infrastructure.true if the type codes are equal. Otherwise, com.rti.dds.infrastructure.false.

See also

http://java.sun.com/javase/6/docs/api/org/omg/CORBA/TypeCode.html#equal(org.omg.CORBA.TypeCode)

References **TypeCode.array_dimension()**, **TypeCode.array_dimension_count()**, **TypeCode.concrete_base_type()**, **TypeCode.content_type()**, **TypeCode.default_index()**, **TypeCode.discriminator_type()**, **TypeCode.equal()**, **TypeCode.equals()**, **TypeCode.extensibility_kind()**, **TypeCode.is_alias_pointer()**, **TypeCode.is_member_bitfield()**, **TypeCode.is_member_key()**, **TypeCode.is_member_pointer()**, **TypeCode.is_member_required()**, **TypeCode.kind()**, **TypeCode.length()**, **TypeCode.member_bitfield_bits()**, **TypeCode.member_count()**, **TypeCode.member_id()**, **TypeCode.member_label()**, **TypeCode.member_label_count()**, **TypeCode.member_name()**, **TypeCode.member_ordinal()**, **TypeCode.member_type()**, **TypeCode.member_visibility()**, **TypeCode.name()**, **TypeCode.TC_NULL**, **TCKind.TK_ALIAS**, **TCKind.TK_ARRAY**, **TCKind.TK_ENUM**, **TCKind.TK_SEQUENCE**, **TCKind.TK_STRING**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, **TCKind.TK_VALUE**, **TCKind.TK_WSTRING**, and **TypeCode.type_modifier()**.

Referenced by **TypeCode.equal()**, and **TypeCode.equals()**.

8.386.2.4 equals()

```
boolean equals (
            Object tc )
```

Compares two **com.rti.dds.typecode.TypeCode** (p. 1873) objects for equality.

MT Safety:

SAFE.

Parameters

tc	<<***in***>> *(p. 156)* Type code that will be compared with this **com.rti.dds.typecode.TypeCode** (p. 1873).

Returns

com.rti.dds.infrastructure.true if the type codes are equal. Otherwise, com.rti.dds.infrastructure.false.

See also

http://java.sun.com/javase/6/docs/api/java/lang/Object.html#equals(java.↩
lang.Object)

References **TypeCode.equal()**.

Referenced by **TypeCode.equal()**.

8.386.2.5 assignable()

```
boolean assignable (
             TypeCode from )
```

Checks if this **com.rti.dds.typecode.TypeCode** (p. 1873) is assignable from the input **com.rti.dds.typecode.Type↩
Code** (p. 1873).

MT Safety:

SAFE.

This method checks if this type code is assignable from the type code provided as a parameter.

In order to maintain the loose coupling between data producers and consumers, especially as systems change over time, it is desirable that the two be permitted to use slightly different versions of a type, and that the infrastructure perform any necessary translation. To support type evolution and inheritance the type system defines the "is-assignable-from" directed binary relationship between every pair of types in the Type System.

Intuitively, if T1 is-assignable-from T2, it means that in general it is possible, in a structural way, to set the contents of an object of type T1 to the contents of an object of T2 (or perhaps a subset of those contents) without leading to incorrect interpretations of that information.

For example:

```
struct MyBaseType {

    long member_1;

};


struct MyDerivedType: MyBaseType {

    long member_2;

};
```

The previous two types that are related to each other by an inheritance relationship are assignable.

For additional information on type assignability refer to the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

Parameters

from	<<***in***>> *(*p. *156)* From type code

Returns

com.rti.dds.infrastructure.true if the "this" is assignable from "from". Otherwise, com.rti.dds.infrastructure.false.

8.386.2.6 length()

```
int length ( ) throws  BadKind
```

Returns the number of elements in the type described by this type code.

Length is:

- The maximum length of the string for string type codes.
- The maximum length of the sequence for sequence type codes.
- The first dimension of the array for array type codes.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_↩ SEQUENCE** (p. 1790), **com.rti.dds.typecode.TCKind.TK_STRING** (p. 1790) or **com.rti.dds.typecode.↩ TCKind.TK_WSTRING** (p. 1792).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

The bound for strings and sequences, or the number of elements for arrays if no errors.

References **TCKind.TK_ARRAY**, **TCKind.TK_SEQUENCE**, **TCKind.TK_STRING**, and **TCKind.TK_WSTRING**.

Referenced by **TypeCode.equal()**.

8.386.2.7 name()

```
String name ( ) throws BadKind
```

Retrieves the simple name identifying this **com.rti.dds.typecode.TypeCode** (p. 1873) object within its enclosing scope.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790), **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), or **com.rti.dds.typecode.TCKind.TK_ALIAS** (p. 1791).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

Name of the type code if no errors.

Referenced by **TypeCode.add_member()**, **TypeCode.add_member_to_enum()**, **TypeCode.add_member_to_union()**, **TypeCode.equal()**, and **TypeCode.find_member_by_name()**.

8.386.2.8 is_alias_pointer()

```
boolean is_alias_pointer ( ) throws BadKind
```

Function that tells if an alias is a pointer or not.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ALIAS** (p. 1791).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

com.rti.dds.infrastructure.true if an alias is a pointer to the aliased type. Otherwise, com.rti.dds.infrastructure.false.

References **TCKind.TK_ALIAS**.

Referenced by **TypeCode.equal()**.

8.386.2.9 type_modifier()

```
short type_modifier ( ) throws BadKind
```

Returns a constant indicating the modifier of the value type that this **com.rti.dds.typecode.TypeCode** (p. 1873) object describes.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) or **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789).

For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), this method always returns com.rti.dds.typecode.Value↩ Modifier.VM_NONE.

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

One of the following type modifiers: com.rti.dds.typecode.ValueModifier.VM_NONE, com.rti.dds.typecode.↩ ValueModifier.VM_ABSTRACT, com.rti.dds.typecode.ValueModifier.VM_CUSTOM or com.rti.dds.typecode.↩ ValueModifier.VM_TRUNCATABLE.

References **TCKind.TK_STRUCT**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.10 concrete_base_type()

```
TypeCode concrete_base_type ( ) throws  BadKind
```

Returns the **com.rti.dds.typecode.TypeCode** (p. 1873) that describes the concrete base type of the value type that this **com.rti.dds.typecode.TypeCode** (p. 1873) object describes.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) or **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789).

MT Safety:

SAFE.

For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), this method always returns **com.rti.dds.typecode.↩ TCKind.TK_NULL** (p. 1787).

The **com.rti.dds.typecode.TypeCode** (p. 1873) of structs inheriting from other structs has a **com.rti.dds.typecode.↩ TCKind.TK_VALUE** (p. 1792) kind.

For example:

```
struct MyStruct: MyBaseStruct {

    long member_1;

};
```

The code generation for the previous type will generate a **com.rti.dds.typecode.TypeCode** (p. 1873) with **com.rti.↩ dds.typecode.TCKind.TK_VALUE** (p. 1792) kind.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

com.rti.dds.typecode.TypeCode (p. 1873) that describes the concrete base type or null if there is no a concrete base type.

References **TypeCode.kind()**, **TypeCode.TC_NULL**, **TCKind.TK_NULL**, **TCKind.TK_STRUCT**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.11 content_type()

```
TypeCode content_type ( ) throws BadKind
```

Returns the **com.rti.dds.typecode.TypeCode** (p. 1873) object representing the type for the members of the object described by this **com.rti.dds.typecode.TypeCode** (p. 1873) object.

For sequences and arrays, it returns the element type. For aliases, it returns the original type.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790), **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790) or **com.rti.dds.typecode.TCKind.TK_ALIAS** (p. 1791).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

A **com.rti.dds.typecode.TypeCode** (p. 1873) object representing the element type for sequences and arrays, and the original type for aliases.

References **TCKind.TK_ALIAS**, **TCKind.TK_ARRAY**, and **TCKind.TK_SEQUENCE**.

Referenced by **TypeCode.equal()**, **DynamicData.get_char_array()**, **DynamicData.get_char_seq()**, **Dynamic↩ Data.set_char_array()**, and **DynamicData.set_char_seq()**.

8.386.2.12 array_dimension_count()

```
int array_dimension_count ( ) throws  BadKind
```

This function returns the number of dimensions of an array type code.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

Number of dimensions if no errors.

References **TCKind.TK_ARRAY**.

Referenced by **TypeCode.equal()**.

8.386.2.13 array_dimension()

```
int array_dimension (
            int index ) throws  BadKind, Bounds
```

This function returns the index-th dimension of an array type code.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790).
Dimension index in the interval [0,(dimensions count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 156)* Dimension index in the interval [0,(dimensions count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

Requested dimension if no errors.

References **TCKind.TK_ARRAY**.

Referenced by **TypeCode.element_count()**, and **TypeCode.equal()**.

8.386.2.14 element_count()

```
int element_count ( ) throws  BadKind
```

The number of elements in an array.

This operation isn't relevant for other kinds of types.

MT Safety:

SAFE.

References **TypeCode.array_dimension()**, and **TCKind.TK_ARRAY**.

8.386.2.15 member_count()

```
int member_count ( ) throws  BadKind
```

Returns the number of members of the type code.

The method member_count can be invoked on structure, union, and enumeration **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

The number of members constituting the type described by this **com.rti.dds.typecode.TypeCode** (p. 1873) object if no errors.

References **TCKind.TK_ENUM**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**, **TypeCode.find_member_by_id()**, **TypeCode.find_member_by_name()**, **TypeCode.is_member_bitfield()**, **TypeCode.is_member_key()**, **TypeCode.is_member_pointer()**, **TypeCode.is↩_member_required()**, **TypeCode.member_bitfield_bits()**, **TypeCode.member_id()**, **TypeCode.member_label()**, **TypeCode.member_label_count()**, **TypeCode.member_name()**, **TypeCode.member_ordinal()**, **TypeCode.↩member_type()**, and **TypeCode.member_visibility()**.

8.386.2.16 member_name()

```
String member_name (
            int index ) throws  BadKind, Bounds
```

Returns the name of a type code member identified by the given index.

The method member_name can be invoked on structure, union, and enumeration **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

Name of the member if no errors.

References **TypeCode.member_count()**, **EnumMember.name**, **StructMember.name**, **UnionMember.name**, **ValueMember.name**, **TCKind.TK_ENUM**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**, and **TypeCode.find_member_by_name()**.

8.386.2.17 member_type()

```
TypeCode member_type (
            int index ) throws  BadKind, Bounds
```

Retrieves the **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the type of the member identified by the given index.

The method member_type can be invoked on structure and union type codes.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

The **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the member at the given index if no errors.

References **TypeCode.member_count()**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, **TCKind.TK_VALUE**, **StructMember.type**, **UnionMember.type**, and **ValueMember.type**.

Referenced by **TypeCode.equal()**.

8.386.2.18 member_id()

```
int member_id (
             int index ) throws  BadKind, Bounds
```

Returns the ID of the **TypeCode** (p. 1873) member identified by the given index.

This function is an RTI Connext extension to the CORBA Type Code Specification.

The method can be invoked on aggregation **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

All members of aggregated types have an integral member ID that uniquely identifies them within their defining type.

In IDL, you can specify the member ID using the built-in annotation ID. For example:

```
struct MyType {

    long member_1; //@ID 200

} //@Extensibility MUTABLE_EXTENSIBILITY
```

In XML, you can specify the member ID using the attribute 'id':

```
<struct name="MyType" extensibility="mutable" id="200">

    <member name="member_1" type="long"/>

</struct>
```

In XSD, you can specify the member ID using the built-in annotation ID

```
<xsd:complexType name="MyType">

    <xsd:sequence>

        <xsd:element name="member_1" minOccurs="1" maxOccurs="1" type="xsd:int"/>

        <!-- @id 200 -->

    </xsd:sequence>

</xsd:complexType>

<!-- @extensibility MUTABLE_EXTENSIBILITY -->
```

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), or **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

ID of the member if no errors.

References **StructMember.id**, **UnionMember.id**, **ValueMember.id**, **TypeCode.member_count()**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.19 member_label_count()

```
int member_label_count (
             int index ) throws  BadKind, Bounds
```

Returns the number of labels associated to the index-th union member.

The method can be invoked on union **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

Number of labels if no errors.

References **UnionMember.labels**, **TypeCode.member_count()**, and **TCKind.TK_UNION**.

Referenced by **TypeCode.equal()**.

8.386.2.20 member_label()

```
int member_label (
             int member_index,
             int label_index ) throws  BadKind, Bounds
```

Return the label_index-th label associated to the member_index-th member.

This method has been modified for RTI Connext from the CORBA Type code Specification.

Example:
case 1: Label index 0
case 2: Label index 1
short short_member;

The method can be invoked on union **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789).
The member_index param must be in the interval [0,(member count-1)].
The label_index param must be in the interval [0,(member labels count-1)].

MT Safety:

SAFE.

Parameters

member_index	<<***in***>> *(*p. *156)* Member index.
label_index	<<***in***>> *(*p. *156)* Label index.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

The evaluated value of the label if no errors.

References **UnionMember.labels**, **TypeCode.member_count()**, and **TCKind.TK_UNION**.

Referenced by **TypeCode.equal()**.

8.386.2.21 member_ordinal()

```
int member_ordinal (
             int index ) throws  BadKind, Bounds
```

Returns the ordinal that corresponds to the index-th enum value.

The method can be invoked on enum **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790).
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

Ordinal that corresponds to the index-th enumerator if no errors.

References **TypeCode.member_count()**, **EnumMember.ordinal**, and **TCKind.TK_ENUM**.

Referenced by **TypeCode.equal()**.

8.386.2.22 is_member_key()

```
boolean is_member_key (
             int index ) throws  BadKind, Bounds
```

Function that tells if a member is a key or not.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(p. 156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

com.rti.dds.infrastructure.true if the member is a key. Otherwise, com.rti.dds.infrastructure.false.

References **StructMember.is_key**, **ValueMember.is_key**, **TypeCode.KEY_MEMBER**, **TypeCode.member_count()**, **TCKind.TK_STRUCT**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.23 is_member_required()

```
boolean is_member_required (
            int index ) throws  BadKind, Bounds
```

Indicates whether a given member of a type is required to be present in every sample of that type.

A non-key member is required if it has not been marked as optional. All key members are required.

See also

com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)

MT Safety:

SAFE.

References **StructMember.is_optional**, **ValueMember.is_optional**, **TypeCode.KEY_MEMBER**, **TypeCode.member_count()**, **TypeCode.NONKEY_REQUIRED_MEMBER**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.24 is_member_pointer()

```
boolean is_member_pointer (
            int index ) throws  BadKind, Bounds
```

Function that tells if a member is a pointer or not.

The method is_member_pointer can be invoked on union and structs type objects

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Index of the member for which type information is begin requested.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

com.rti.dds.infrastructure.true if the member is a pointer. Otherwise, com.rti.dds.infrastructure.false.

References **StructMember.is_pointer**, **UnionMember.is_pointer**, **ValueMember.is_pointer**, **TypeCode.member_count()**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.25 is_member_bitfield()

```
boolean is_member_bitfield (
            int index ) throws  BadKind, Bounds
```

Function that tells if a member is a bitfield or not.

The method can be invoked on struct type objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

com.rti.dds.infrastructure.true if the member is a bitfield. Otherwise, com.rti.dds.infrastructure.false.

References **TypeCode.member_count()**, **TypeCode.NOT_BITFIELD**, **TCKind.TK_STRUCT**, and **TCKind.TK_↩ VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.26 member_bitfield_bits()

```
short member_bitfield_bits (
            int index ) throws  BadKind, Bounds
```

Returns the number of bits of a bitfield member.

The method can be invoked on struct type objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) or **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

index	<<***in***>> *(*p. *156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

The number of bits of the bitfield or **com.rti.dds.typecode.TypeCode.NOT_BITFIELD** (p. 1920) if the member is not a bitfield.

References **StructMember.bits**, **ValueMember.bits**, **TypeCode.member_count()**, **TCKind.TK_STRUCT**, and **TCKind.TK_VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.27 member_visibility()

```
short member_visibility (
            int index ) throws  BadKind, Bounds
```

Returns the constant that indicates the visibility of the index-th member.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) or **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789). The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), this method always returns com.rti.dds.typecode.↩ Visibility.PUBLIC_MEMBER.

Parameters

index	<<***in***>> *(p. 156)* Member index in the interval [0,(member count-1)].

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.
com.rti.dds.infrastructure.ExceptionCode_t.Bounds	- if the index parameter/s are out of range.

Returns

One of the following constants: com.rti.dds.typecode.Visibility.PRIVATE_MEMBER or com.rti.dds.typecode.↩ Visibility.PUBLIC_MEMBER.

References **ValueMember.access**, **TypeCode.member_count()**, **TCKind.TK_STRUCT**, **TCKind.TK_VALUE**, and **PUBLIC_MEMBER.VALUE**.

Referenced by **TypeCode.equal()**.

8.386.2.28 discriminator_type()

```
TypeCode discriminator_type ( ) throws BadKind
```

Returns the discriminator type code.

The method discriminator_type can be invoked only on union **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789).

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

com.rti.dds.typecode.TypeCode (p. 1873) object describing the discriminator of the union type if no errors.

References **TCKind.TK_UNION**.

Referenced by **TypeCode.equal()**.

8.386.2.29 default_index()

```
int default_index ( ) throws BadKind
```

Returns the index of the default member, or -1 if there is no default member.

The method default_index can be invoked only on union **com.rti.dds.typecode.TypeCode** (p. 1873) objects.

Precondition

self kind is **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)

MT Safety:

SAFE.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

The index of the default member, or -1 if there is no default member.

References **TCKind.TK_UNION**.

Referenced by **TypeCode.equal()**.

8.386.2.30 find_member_by_id()

```
int find_member_by_id (
             int id ) throws  BadKind
```

Get the index of the member of the given ID.

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing structs (**com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789)) and union (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)) types.

Parameters

id	<<***in***>> *(p. 156)* The member ID.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

The index of the member of the given ID or **com.rti.dds.typecode.TypeCode.INDEX_INVALID** (p. 1918) if the member is not found.

MT Safety:

SAFE.

References **TypeCode.INDEX_INVALID**, and **TypeCode.member_count()**.

8.386.2.31 find_member_by_name()

```
int find_member_by_name (
            String name ) throws  BadKind
```

Get the index of the member of the given name.

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing structs (**com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789)) and union (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)) types.

Parameters

name	<<***in***>> *(*p. *156)* The member name.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	if the method is invoked on an inappropriate kind of **TypeCode** (p. 1873) object.

Returns

The index of the member of the given name or **com.rti.dds.typecode.TypeCode.INDEX_INVALID** (p. 1918) if the member is not found.

MT Safety:

SAFE.

References **TypeCode.INDEX_INVALID**, **TypeCode.member_count()**, **TypeCode.member_name()**, **TypeCode.name()**, **TCKind.TK_ENUM**, **TCKind.TK_STRUCT**, **TCKind.TK_UNION**, and **TCKind.TK_VALUE**.

8.386.2.32 print_IDL() [1/2]

```
void print_IDL (
            int indent )
```

Prints a **com.rti.dds.typecode.TypeCode** (p. 1873) in IDL notation.

MT Safety:

SAFE.

Parameters

indent	<<***in***>> *(*p. *156)* Indent.

References **TypeCode.print_IDL()**.

Referenced by **TypeCode.print_complete_IDL()**, and **TypeCode.print_IDL()**.

8.386.2.33 print_IDL() [2/2]

```
void print_IDL (
            int indent,
            Writer out ) throws IOException
```

Prints a **com.rti.dds.typecode.TypeCode** (p. 1873) in a pseudo-IDL notation.

MT Safety:

SAFE.

Parameters

indent	<<***in***>> *(*p. *156)* Indent.
out	<<***out***>> *(*p. *156)* Output stream.

References **TypeCode.print_IDL()**.

8.386.2.34 print_complete_IDL()

```
void print_complete_IDL (
            Writer writer ) throws IOException
```

Print all IDL constructs needed by the code generator to generate code for this type.

Parameters

writer	The writer to which the IDL is to be written.

Exceptions

IOException	If there were issues writing to the writer.

References **TypeCode.print_IDL()**.

8.386.2.35 add_member() [1/2]

```
int add_member (
             String name,
             int id,
              TypeCode tc,
             byte member_flags ) throws  BadKind, BadMemberName, BadMemberId
```

Add a new member to this **com.rti.dds.typecode.TypeCode** (p. 1873).

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing structures (**com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789)), value types (**com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792)), and unions (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)). To add a constant to an enumeration, see **com.rti.dds.typecode.TypeCode.add_member_to_enum** (p. 1908).

Calling this method clones the type code passed as the `tc` parameter if the type code is not a builtin one. To delete this cloned type code, call **com.rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934).

The ability to modify a **com.rti.dds.typecode.TypeCode** (p. 1873) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 65) APIs.

Here's a simple code example that adds two fields to a data type, one an integer and another a sequence of integers.

```
// Integer:

myTypeCode.add_member(

    "myFieldName",

    TypeCode.MEMBER_ID_INVALID,

    TypeCode.TC_LONG,

    // New field is a required non-key member:

    TypeCode.NONKEY_REQUIRED_MEMBER);


// Sequence of 10 or fewer integers:

myTypeCode.add_member(

    "myFieldName",

    TypeCode.MEMBER_ID_INVALID,

    TypeCodeFactory.get_instance().create_sequence_tc(10, TypeCode.TC_LONG),

    // New field is a required non-key member:

    TypeCode.NONKEY_REQUIRED_MEMBER);
```

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(p. 156)* The name of the new member.
id	<<***in***>> *(p. 156)* Member ID or case value. • For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789), and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), this parameter is used to provide the member ID. For automatic assignment of the member ID specify **com.rti.dds.typecode.TypeCode.MEMBER_ID_INVALID** (p. 1918). • For **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), this parameter is used to provide the case value associated with the member. The member ID is assigned automatically. To explicitly assign the member ID, use the API com.rti.dds.typecode.TypeCode.add_member_to_union.
tc	<<***in***>> *(p. 156)* The type of the new member. You can get or create this **com.rti.dds.typecode.TypeCode** (p. 1873) with the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921).
member_flags	<<***in***>> *(p. 156)* Indicates whether the member is part of the key and whether it is required.

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.add_member_to_enum (p. 1908)

com.rti.dds.typecode.TypeCodeFactory (p. 1921)

com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)

References **TypeCode.add_member()**, **TypeCode.name()**, **TypeCode.NOT_BITFIELD**, and **PUBLIC_MEMBER.VALUE**.

Referenced by **TypeCode.add_member()**.

8.386.2.36 add_member() [2/2]

```
synchronized int add_member (
             String name,
             int id,
              TypeCode tc,
             byte member_flags,
             short visibility,
             boolean is_pointer,
```

```
            short bits ) throws  BadKind, BadMemberName, BadMemberId, BAD_PARAM,IllegalState↩
Exception
```

Add a new member to this **com.rti.dds.typecode.TypeCode** (p. 1873).

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing structures (**com.rti.↩ dds.typecode.TCKind.TK_STRUCT** (p. 1789)), value types (**com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792)), and unions (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)). To add a constant to an enumeration, see **com.↩ rti.dds.typecode.TypeCode.add_member_to_enum** (p. 1908).

Calling this method clones the type code passed in as the `tc` parameter if the type code is not a builtin one. To delete this cloned type code, call **com.rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934).

The ability to modify a **com.rti.dds.typecode.TypeCode** (p. 1873) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 65) APIs.

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(p. 156)* The name of the new member.
id	<<***in***>> *(p. 156)* Member ID or case value. • For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792), this parameter is used to provide the member ID. • For **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) and **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) only: For automatic assignment of the member ID, specify **com.rti.dds.typecode.TypeCode.MEMBER_ID_INVALID** (p. 1918). • For **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789), this parameter is used to provide the case value associated with the member. The member ID is assigned automatically. To explicitly assign the member ID, use the API com.rti.dds.typecode.TypeCode.add_member_to_union.
tc	<<***in***>> *(p. 156)* The type of the new member. You can get or create this **com.rti.dds.typecode.TypeCode** (p. 1873) with the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921).
member_flags	<<***in***>> *(p. 156)* Indicates whether the member is part of the key and whether it is required.
visibility	<<***in***>> *(p. 156)* Whether the new member is public or private. Non-public members are only relevant for types of kind **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792). Possible values include: • com.rti.dds.typecode.Visibility.PRIVATE_MEMBER • com.rti.dds.typecode.Visibility.PUBLIC_MEMBER
is_pointer	<<***in***>> *(p. 156)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.
bits	<<***in***>> *(p. 156)* The number of bits, if this new member is a bit field, or **com.rti.dds.typecode.TypeCode.NOT_BITFIELD** (p. 1920).

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.add_member_to_enum (p. 1908)

com.rti.dds.typecode.TypeCode.add_member_to_union

com.rti.dds.typecode.TypeCodeFactory (p. 1921)

com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)

References **TypeCode.add_member()**, and **TypeCode.name()**.

8.386.2.37 add_member_to_enum()

```
synchronized int add_member_to_enum (
            String name,
            int ordinal ) throws  BadKind, BadMemberName
```

Add a new enumerated constant to this enum **com.rti.dds.typecode.TypeCode** (p. 1873).

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing enumerations (**com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790)). To add a field to a structured type, see **com.rti.dds.typecode.TypeCode.add_member_to_enum** (p. 1908).

Modifying a **com.rti.dds.typecode.TypeCode** (p. 1873) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 65) APIs.

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(p. 156)* The name of the new member. This string must be unique within this type and must not be null.
ordinal	<<***in***>> *(p. 156)* The relative order of the new member in this enum or a custom integer value. The value must be unique within the type.

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCodeFactory (p. 1921)

References **TypeCode.INDEX_INVALID**, **TypeCode.name()**, and **TCKind.TK_ENUM**.

8.386.2.38 add_member_to_union()

```
synchronized int add_member_to_union (
             String name,
             int id,
             int labels_in[],
              TypeCode tc,
             boolean is_pointer ) throws  BadKind, BadMemberName,  BadMemberId
```

Add a new member to a union **com.rti.dds.typecode.TypeCode** (p. 1873).

This method is applicable to **com.rti.dds.typecode.TypeCode** (p. 1873) objects representing unions (**com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789)).

Modifying a **com.rti.dds.typecode.TypeCode** (p. 1873) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 65) APIs.

MT Safety:

UNSAFE.

Parameters

name	<<***in***>> *(p. 156)* The name of the new member.
id	<<***in***>> *(p. 156)* The member ID. For automatic assignment of the member ID specify **com.rti.dds.typecode.TypeCode.MEMBER_ID_INVALID** (p. 1918).
labels_in	<<***in***>> *(p. 156)* A sequence of labels or case values associated with the member .
tc	<<***in***>> *(p. 156)* The type of the new member. You can get or create this **com.rti.dds.typecode.TypeCode** (p. 1873) with the **com.rti.dds.typecode.TypeCodeFactory** (p. 1921).
is_pointer	<<***in***>> *(p. 156)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BadKind	com.rti.dds.infrastructure.ExceptionCode_t.BadMember↩ Name com.rti.dds.infrastructure.ExceptionCode_t.BadMemberId

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

com.rti.dds.typecode.TypeCode.add_member (p. 1905)
com.rti.dds.typecode.TypeCode.add_member (p. 1905)
com.rti.dds.typecode.TypeCode.add_member_to_enum (p. 1908)
com.rti.dds.typecode.TypeCode.add_member_to_union
com.rti.dds.typecode.TypeCodeFactory (p. 1921)
com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)
com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)
com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)

References **TypeCode.name()**.

8.386.2.39 signature()

```
String signature ( ) throws  RETCODE_ERROR
```

Gets an MD5 signature of the **com.rti.dds.typecode.TypeCode** (p. 1873).

MT Safety:

SAFE.

Returns

MD5 signature.

References **TypeCode.signature()**.

Referenced by **TypeCode.signature()**.

8.386.2.40 cdr_serialized_sample_min_size() [1/2]

```
final long cdr_serialized_sample_min_size (
            short representation_id ) throws  UserException,  SystemException
```

Gets the minimum serialized size of samples of this type.

Obtains the minimum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the minimum size.

Returns

The minimum size

8.386.2.41 cdr_serialized_sample_min_size() [2/2]

```
final long cdr_serialized_sample_min_size ( ) throws UserException, SystemException
```

Gets the minimum serialized size of samples of this type.

Obtains the minimum possible size in bytes of any serialized data sample of this type.

Assumes the com.rti.dds.infrastructure.DataRepresentationId_t to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Returns

The minimum size

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **TypeCode.cdr_serialized_↩ sample_min_size()**.

Referenced by **TypeCode.cdr_serialized_sample_min_size()**.

8.386.2.42 cdr_serialized_sample_max_size() [1/2]

```
final long cdr_serialized_sample_max_size (
            short representation_id ) throws UserException, SystemException
```

Gets the maximum serialized size of samples of this type.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum size.

Returns

The maximum size

8.386.2.43 cdr_serialized_sample_max_size() [2/2]

```
final long cdr_serialized_sample_max_size ( ) throws UserException, SystemException
```

Gets the maximum serialized size of samples of this type.

Obtains the maximum possible size in bytes of any serialized data sample of this type.

Assumes the com.rti.dds.infrastructure.DataRepresentationId_t to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Returns

The maximum size

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **TypeCode.cdr_serialized_↩ sample_max_size()**.

Referenced by **TypeCode.cdr_serialized_sample_max_size()**.

8.386.2.44 cdr_serialized_sample_key_max_size() [1/2]

```
final long cdr_serialized_sample_key_max_size (
            short representation_id ) throws UserException, SystemException
```

Gets the maximum serialized size of sample keys of this type.

Obtains the maximum possible size in bytes of the serialized keys of any data sample of this type

Precondition

The type is an aggregation type (struct, union)

Parameters

representation↩ _id	The serialized data representation for which we calculate the maximum key size.

Returns

The maximum key size

8.386.2.45 cdr_serialized_sample_key_max_size() [2/2]

```
final long cdr_serialized_sample_key_max_size ( ) throws UserException, SystemException
```

Gets the maximum serialized size of sample keys of this type.

Obtains the maximum possible size in bytes of the serialized keys of any data sample of this type

Assumes the com.rti.dds.infrastructure.DataRepresentationId_t to be DDS_AUTO_DATA_REPRESENTATION

Precondition

The type is an aggregation type (struct, union)

Returns

The maximum key size

References **DataRepresentationQosPolicy.AUTO_DATA_REPRESENTATION**, and **TypeCode.cdr_serialized_↩ sample_key_max_size()**.

Referenced by **TypeCode.cdr_serialized_sample_key_max_size()**.

8.386.2.46 get_type_object_serialized_size()

```
long get_type_object_serialized_size ( )
```

Gets the serialized size of the TypeObject created from this **com.rti.dds.typecode.TypeCode** (p. 1873).

The default buffer size used for storing a TypeObject is 8192 bytes. For a large TypeObject, this API can be used to determine the size that needs to be set in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQos↩ Policy.type_object_max_serialized_length** (p. 817)

MT Safety:

SAFE.

Returns

The TypeObject serialized size.

8.386.3 Member Data Documentation

8.386.3.1 TC_NULL

```
final TypeCode TC_NULL [static]
```

Basic null type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCode.concrete_base_type()**, **TypeCode.equal()**, and **TypeCodeFactory.get_primitive_tc()**.

8.386.3.2 TC_SHORT

```
final TypeCode TC_SHORT [static]
```

Basic 16-bit signed integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.3 TC_LONG

```
final TypeCode TC_LONG [static]
```

Basic 32-bit signed integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.4 TC_USHORT

```
final TypeCode TC_USHORT [static]
```

Basic unsigned 16-bit integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.5 TC_ULONG

```
final TypeCode TC_ULONG [static]
```

Basic unsigned 32-bit integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.6 TC_FLOAT

```
final TypeCode TC_FLOAT [static]
```

Basic 32-bit floating point type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.7 TC_DOUBLE

```
final TypeCode TC_DOUBLE [static]
```

Basic 64-bit floating point type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.8 TC_BOOLEAN

```
final TypeCode TC_BOOLEAN [static]
```

Basic Boolean type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.9 TC_CHAR

```
final TypeCode TC_CHAR [static]
```

Basic single-byte character type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.10 TC_OCTET

```
final TypeCode TC_OCTET [static]
```

Basic octet/byte type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.11 TC_LONGLONG

```
final TypeCode TC_LONGLONG [static]
```

Basic 64-bit integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.12 TC_ULONGLONG

```
final TypeCode TC_ULONGLONG [static]
```

Basic unsigned 64-bit integer type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.13 TC_LONGDOUBLE

```
final TypeCode TC_LONGDOUBLE [static]
```

Basic 128-bit floating point type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.14 TC_WCHAR

```
final TypeCode TC_WCHAR [static]
```

Basic four-byte character type.

See also

com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc (p. 1935)

Referenced by **TypeCodeFactory.get_primitive_tc()**.

8.386.3.15 MEMBER_ID_INVALID

```
final int MEMBER_ID_INVALID [static]
```

A sentinel indicating an invalid **com.rti.dds.typecode.TypeCode** (p. 1873) member ID.

Referenced by **StructMember.StructMember()**, **UnionMember.UnionMember()**, and **ValueMember.Value↩ Member()**.

8.386.3.16 MAX_MEMBER_ID

```
final int MAX_MEMBER_ID [static]
```

Maximum value for member id. ID.

This value was introduced with the Extensible Types specification

8.386.3.17 INDEX_INVALID

```
final int INDEX_INVALID  [static]
```

A sentinel indicating an invalid **com.rti.dds.typecode.TypeCode** (p. 1873) member index.

Referenced by **TypeCode.add_member_to_enum()**, **TypeCode.find_member_by_id()**, and **TypeCode.find_member_by_name()**.

8.386.3.18 NONKEY_MEMBER

```
final byte NONKEY_MEMBER  [static]
```

A flag indicating that a type member is optional and not part of the key.

If a type is used with the **Dynamic Data** (p. 65) facility, a **com.rti.dds.dynamicdata.DynamicData** (p. 847) sample of the type will only contain a value for a **com.rti.dds.typecode.TypeCode.NONKEY_MEMBER** (p. 1919) field if one has been explicitly set (see, for example, com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int). The middleware will *not* assume any default value.

See also

com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)
com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)
com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)
com.rti.dds.typecode.TypeCode.add_member (p. 1905)
com.rti.dds.typecode.TypeCode.add_member (p. 1905)
com.rti.dds.typecode.TypeCode.is_member_key (p. 1896)
com.rti.dds.typecode.TypeCode.is_member_required (p. 1897)
com.rti.dds.typecode.StructMember.is_key (p. 1729)
com.rti.dds.typecode.ValueMember.is_key (p. 1965)

8.386.3.19 KEY_MEMBER

```
final byte KEY_MEMBER  [static]
```

A flag indicating that a type member is part of the key for that type, and therefore required.

If a type is used with the **Dynamic Data** (p. 65) facility, all **com.rti.dds.dynamicdata.DynamicData** (p. 847) samples of the type will contain a value for all **com.rti.dds.typecode.TypeCode.KEY_MEMBER** (p. 1919) fields. If you do not set a value of the member explicitly (see, for example, com.rti.dds.dynamicdata.DynamicData.DynamicData.set_int), the middleware will assume a default "zero" value: numeric values will be set to zero; strings and sequences will be of zero length.

See also

com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER (p. 1920)

com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.is_member_key (p. 1896)

com.rti.dds.typecode.TypeCode.is_member_required (p. 1897)

com.rti.dds.typecode.StructMember.is_key (p. 1729)

com.rti.dds.typecode.ValueMember.is_key (p. 1965)

Referenced by **TypeCode.is_member_key()**, and **TypeCode.is_member_required()**.

8.386.3.20 NONKEY_REQUIRED_MEMBER

```
final byte NONKEY_REQUIRED_MEMBER  [static]
```

A flag indicating that a type member is not part of the key but is nevertheless required.

This is the most common kind of member.

If a type is used with the **Dynamic Data** (p. 65) facility, all **com.rti.dds.dynamicdata.DynamicData** (p. 847) samples of the type will contain a value for all **com.rti.dds.typecode.TypeCode.NONKEY_REQUIRED_MEMBER** (p. 1920) fields. If you do not set a value of the member explicitly (see, for example, com.rti.dds.dynamicdata.DynamicData.↩ DynamicData.set_int), the middleware will assume a default "zero" value: numeric values will be set to zero; strings and sequences will be of zero length.

See also

com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.NONKEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.KEY_MEMBER (p. 1919)

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.add_member (p. 1905)

com.rti.dds.typecode.TypeCode.is_member_key (p. 1896)

com.rti.dds.typecode.TypeCode.is_member_required (p. 1897)

com.rti.dds.typecode.StructMember.is_key (p. 1729)

com.rti.dds.typecode.ValueMember.is_key (p. 1965)

Referenced by **TypeCode.is_member_required()**.

8.386.3.21 NOT_BITFIELD

```
final short NOT_BITFIELD  [static]
```

Indicates that a member of a type is not a bitfield.

Referenced by **TypeCode.add_member()**, and **TypeCode.is_member_bitfield()**.

8.387 TypeCodeFactory Class Reference

A singleton factory for creating, copying, and deleting data type definitions dynamically.

Public Member Functions

- **TypeCode create_struct_tc** (String name, **ExtensibilityKind** extensibility_kind, **StructMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_struct_tc** (String name, **StructMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_value_tc** (String name, **ExtensibilityKind** extensibility_kind, short type_modifier, **Type↩ Code** concrete_base, **ValueMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_value_tc** (String name, short type_modifier, **TypeCode** concrete_base, **ValueMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_union_tc** (String name, **ExtensibilityKind** extensibility_kind, **TypeCode** discriminator_↩ type, int default_index, **UnionMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_union_tc** (String name, **TypeCode** discriminator_type, int default_index, **UnionMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_enum_tc** (String name, **ExtensibilityKind** extensibility_kind, **EnumMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_enum_tc** (String name, **EnumMember**[ ] members) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_alias_tc** (String name, **TypeCode** original_type, boolean is_pointer) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_ALIAS** (p. 1791) (typedef) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_string_tc** (int bound) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_STRING** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_wstring_tc** (int bound) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_WSTRING** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_sequence_tc** (int bound, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode create_array_tc** (int[ ] dimensions, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).*

- **TypeCode create_array_tc** (int length, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873) for a single-dimensional array.*
- **TypeCode clone_tc** (**TypeCode** tc)

  *Creates and returns a copy of the input **com.rti.dds.typecode.TypeCode** (p. 1873).*
- void **delete_tc** (**TypeCode** tc)

  *Deletes the input **com.rti.dds.typecode.TypeCode** (p. 1873).*
- **TypeCode get_primitive_tc** (**TCKind** kind) throws BAD_PARAM

  *Get the **com.rti.dds.typecode.TypeCode** (p. 1873) for a primitive type (integers, floating point values, etc.) identified by the given **com.rti.dds.typecode.TCKind** (p. 1786).*

Static Public Member Functions

- static final **TypeCodeFactory get_instance** ()

  Gets the singleton instance of this class.

8.387.1 Detailed Description

A singleton factory for creating, copying, and deleting data type definitions dynamically.

You can access the singleton with the **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923) method.

If you want to publish and subscribe to data of types that are not known to you at system design time, this class will be your starting point. After creating a data type definition with this class, you will modify that definition using the **com.rti.dds.typecode.TypeCode** (p. 1873) class and then register it with the **Dynamic Data** (p. 65) API.

The methods of this class fall into several categories:

Getting definitions for primitive types:
Type definitions for primitive types (e.g. integers, floating point values, etc.) are pre-defined; your application only needs to *get* them, not *create* them.

- **com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc** (p. 1935)

Creating definitions for strings, arrays, and sequences:
Type definitions for strings, arrays, and sequences (i.e. variables-size lists) must be created as you need them, because the type definition includes the maximum length of those containers.

- **com.rti.dds.typecode.TypeCodeFactory.create_string_tc** (p. 1931)
- **com.rti.dds.typecode.TypeCodeFactory.create_wstring_tc** (p. 1932)
- **com.rti.dds.typecode.TypeCodeFactory.create_array_tc** (p. 1933)
- com.rti.dds.typecode.TypeCodeFactory.TypeCodeFactory.create_array_tc
- **com.rti.dds.typecode.TypeCodeFactory.create_sequence_tc** (p. 1932)

Creating definitions for structured types:
Structured types include structures, value types, and unions.

- **com.rti.dds.typecode.TypeCodeFactory.create_struct_tc** (p. 1923)
- **com.rti.dds.typecode.TypeCodeFactory.create_value_tc** (p. 1925)
- **com.rti.dds.typecode.TypeCodeFactory.create_union_tc** (p. 1927)

Creating definitions for other types:
The type system also supports enumerations and aliases (i.e. `typedef`s in C and C++).

- **com.rti.dds.typecode.TypeCodeFactory.create_enum_tc** (p. 1928)
- **com.rti.dds.typecode.TypeCodeFactory.create_alias_tc** (p. 1931)

Deleting type definitions:
When you're finished using a type definition, you should delete it. (*Note* that you only need to delete a **com.rti.dds.typecode.TypeCode** (p. 1873) that you *created*; if you got the object from **com.rti.dds.typecode.TypeCodeFactory.get_primitive_tc** (p. 1935), you must *not* delete it.)

- **com.rti.dds.typecode.TypeCodeFactory.delete_tc** (p. 1934)

Copying type definitions:
You can also create deep copies of type definitions:

- **com.rti.dds.typecode.TypeCodeFactory.clone_tc** (p. 1934)

8.387.2 Member Function Documentation

8.387.2.1 get_instance()

```
static final TypeCodeFactory get_instance ( ) [static]
```

Gets the singleton instance of this class.

Returns

The **com.rti.dds.typecode.TypeCodeFactory** (p. 1921) instance if no errors. Otherwise, null.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

8.387.2.2 create_struct_tc() [1/2]

```
TypeCode create_struct_tc (
            String name,
             ExtensibilityKind extensibility_kind,
             StructMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the struct type. Cannot be null.
extensibility_kind	<<***in***>> *(*p. *156)* Type extensibility.
members	<<***in***>> *(*p. *156)* Initial members of the structure. This list may be empty (that is, com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size() may return zero). If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a struct.

8.387.2.3 create_struct_tc() [2/2]

```
TypeCode create_struct_tc (
            String name,
             StructMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_STRUCT** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the struct type. Cannot be null.
members	<<***in***>> *(*p. *156)* Initial members of the structure. This list may be empty (that is, com.rti.dds.infrastructure.com.rti.dds.util.Sequence.Sequence.size() may return zero). If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a struct.

References **ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY**.

8.387.2.4 create_value_tc() [1/2]

```
TypeCode create_value_tc (
            String name,
             ExtensibilityKind extensibility_kind,
            short type_modifier,
             TypeCode concrete_base,
             ValueMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	*<<**in**>> (p. 156)* Name of the value type. Cannot be null.
extensibility_kind	*<<**in**>> (p. 156)* Type extensibility.
type_modifier	*<<**in**>> (p. 156)* One of the value type modifier constants: com.rti.dds.typecode.ValueModifier.VM_NONE, com.rti.dds.typecode.ValueModifier.VM_CUSTOM, com.rti.dds.typecode.ValueModifier.VM_ABSTRACT or com.rti.dds.typecode.ValueModifier.VM_TRUNCATABLE.
concrete_base	*<<**in**>> (p. 156)* **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.
members	*<<**in**>> (p. 156)* Initial members of the value type. This list may be empty. If the list is not empty, the elements must describe valid value type members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a value.

8.387.2.5 create_value_tc() [2/2]

```
TypeCode create_value_tc (
            String name,
            short type_modifier,
             TypeCode concrete_base,
             ValueMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_VALUE** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the value type. Cannot be null.
type_modifier	<<***in***>> *(*p. *156)* One of the value type modifier constants: com.rti.dds.typecode.ValueModifier.VM_NONE, com.rti.dds.typecode.ValueModifier.VM_CUSTOM, com.rti.dds.typecode.ValueModifier.VM_ABSTRACT or com.rti.dds.typecode.ValueModifier.VM_TRUNCATABLE.
concrete_base	<<***in***>> *(*p. *156)* **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.
members	<<***in***>> *(*p. *156)* Initial members of the value type. This list may be empty. If the list is not empty, the elements must describe valid value type members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a value.

References **ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY**, and **TCKind.TK_NULL**.

8.387.2.6 create_union_tc() [1/2]

```
TypeCode create_union_tc (
            String name,
             ExtensibilityKind extensibility_kind,
             TypeCode discriminator_type,
            int default_index,
             UnionMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the union type. Cannot be null.
extensibility_kind	<<***in***>> *(*p. *156)* Type extensibility.
discriminator_type	<<***in***>> *(*p. *156)* Discriminator Type Code. Cannot be null.
default_index	<<***in***>> *(*p. *156)* Index of the default member, or -1 if there is no default member.
members	<<***in***>> *(*p. *156)* Initial members of the union. This list may be empty. If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a union.

8.387.2.7 create_union_tc() [2/2]

```
TypeCode create_union_tc (
            String name,
             TypeCode discriminator_type,
            int default_index,
             UnionMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_UNION** (p. 1789) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(p. 156)* Name of the union type. Cannot be null.
discriminator_type	<<***in***>> *(p. 156)* Discriminator Type Code. Cannot be null.
default_index	<<***in***>> *(p. 156)* Index of the default member, or -1 if there is no default member.
members	<<***in***>> *(p. 156)* Initial members of the union. This list may be empty. If the list is not empty, the elements must describe valid struct members. (For example, the names must be unique within the type.) This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a union.

References **ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY**.

8.387.2.8 create_enum_tc() [1/2]

```
TypeCode create_enum_tc (
            String name,
```

```
ExtensibilityKind extensibility_kind,
EnumMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the enum type. Cannot be null.
extensibility_kind	<<***in***>> *(*p. *156)* Type extensibility.
members	<<***in***>> *(*p. *156)* Initial members of the enumeration. All members must have non-null names, and both names and ordinal values must be unique within the type. Note that it is also possible to add members later with **com.rti.dds.typecode.TypeCode.add_member_to_enum** (p. 1908). This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing an enumeration.

References **Enum.ordinal()**, and **TCKind.TK_ENUM**.

8.387.2.9 create_enum_tc() [2/2]

```
TypeCode create_enum_tc (
            String name,
             EnumMember[] members ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_ENUM** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(*p. *156)* Name of the enum type. Cannot be null.
members	<<***in***>> *(*p. *156)* Initial members of the enumeration. All members must have non-null names, and both names and ordinal values must be unique within the type. Note that it is also possible to add members later with **com.rti.dds.typecode.TypeCode.add_member_to_enum** (p. 1908). This argument may be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing an enumeration.

References **ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY**, **Enum.ordinal()**, and **TCKind.TK_ENUM**.

8.387.2.10 create_alias_tc()

```
TypeCode create_alias_tc (
            String name,
             TypeCode original_type,
            boolean is_pointer ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_ALIAS** (p. 1791) (typedef) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

name	<<***in***>> *(p. 156)* Name of the alias. Cannot be null.
original_type	<<***in***>> *(p. 156)* Aliased type code. Cannot be null.
is_pointer	<<***in***>> *(p. 156)* Indicates if the alias is a pointer to the aliased type code.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing an alias.

8.387.2.11 create_string_tc()

```
TypeCode create_string_tc (
            int bound ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_STRING** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

bound	<<***in***>> *(p. 156)* Maximum length of the string. It cannot be negative.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a string.

References **TCKind.TK_STRING**.

8.387.2.12 create_wstring_tc()

```
TypeCode create_wstring_tc (
            int bound ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_WSTRING** (p. 1792) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

bound	*<<**in**>> (p. 156)* Maximum length of the wide string. It cannot be negative.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a wide string.

References **TCKind.TK_WSTRING**.

8.387.2.13 create_sequence_tc()

```
TypeCode create_sequence_tc (
            int bound,
             TypeCode element_type ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_SEQUENCE** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

bound	*<<**in**>> (p. 156)* The bound for the sequence (> 0).
element_type	*<<**in**>> (p. 156)* **com.rti.dds.typecode.TypeCode** (p. 1873) object describing the sequence elements.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a sequence.

8.387.2.14 create_array_tc() [1/2]

```
TypeCode create_array_tc (
            int[] dimensions,
             TypeCode element_type ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

dimensions	<<***in***>> *(*p. *156)* Dimensions of the array. Each dimension has to be greater than 0.
element_type	<<***in***>> *(*p. *156)* **com.rti.dds.typecode.TypeCode** (p. 1873) describing the array elements. Cannot be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a sequence.

Referenced by **TypeCodeFactory.create_array_tc()**.

8.387.2.15 create_array_tc() [2/2]

```
TypeCode create_array_tc (
            int length,
             TypeCode element_type ) throws  BAD_PARAM
```

Constructs a **com.rti.dds.typecode.TCKind.TK_ARRAY** (p. 1790) **com.rti.dds.typecode.TypeCode** (p. 1873) for a single-dimensional array.

Parameters

length	*<<**in**>> (p. 156)* Length of the single-dimensional array.
element_type	*<<**in**>> (p. 156)* **com.rti.dds.typecode.TypeCode** (p. 1873) describing the array elements. Cannot be null.

Exceptions

com.rti.dds.infrastructure.ExceptionCode_t.BAD_PARAM.	Illegal parameter value.

Returns

A newly-created **com.rti.dds.typecode.TypeCode** (p. 1873) object describing a sequence.

References **TypeCodeFactory.create_array_tc()**.

8.387.2.16 clone_tc()

```
TypeCode clone_tc (
            TypeCode tc )
```

Creates and returns a copy of the input **com.rti.dds.typecode.TypeCode** (p. 1873).

Parameters

tc	*<<**in**>> (p. 156)* Type code that will be copied. Cannot be null.

Returns

A clone of tc.

8.387.2.17 delete_tc()

```
void delete_tc (
            TypeCode tc )
```

Deletes the input **com.rti.dds.typecode.TypeCode** (p. 1873).

Calling this method is optional. If you do not call it, the garbage collector will perform the deletion when it is able.

Parameters

tc	<<***inout***>> *(p. 156)* Type code that will be deleted. Cannot be null.

8.387.2.18 get_primitive_tc()

```
TypeCode get_primitive_tc (
            TCKind kind ) throws  BAD_PARAM
```

Get the **com.rti.dds.typecode.TypeCode** (p. 1873) for a primitive type (integers, floating point values, etc.) identified by the given **com.rti.dds.typecode.TCKind** (p. 1786).

See also

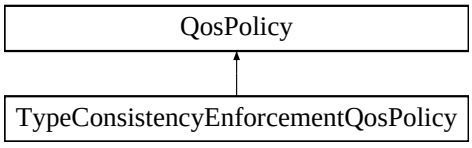
com.rti.dds.typecode.TypeCode.TypeCode.TC_LONG
com.rti.dds.typecode.TypeCode.TypeCode.TC_ULONG
com.rti.dds.typecode.TypeCode.TypeCode.TC_SHORT
com.rti.dds.typecode.TypeCode.TypeCode.TC_USHORT
com.rti.dds.typecode.TypeCode.TypeCode.TC_FLOAT
com.rti.dds.typecode.TypeCode.TypeCode.TC_DOUBLE
com.rti.dds.typecode.TypeCode.TypeCode.TC_LONGDOUBLE
com.rti.dds.typecode.TypeCode.TypeCode.TC_OCTET
com.rti.dds.typecode.TypeCode.TypeCode.TC_BOOLEAN
com.rti.dds.typecode.TypeCode.TypeCode.TC_CHAR
com.rti.dds.typecode.TypeCode.TypeCode.TC_WCHAR

References **TypeCode.TC_BOOLEAN**, **TypeCode.TC_CHAR**, **TypeCode.TC_DOUBLE**, **TypeCode.TC_FLOAT**, **TypeCode.TC_LONG**, **TypeCode.TC_LONGDOUBLE**, **TypeCode.TC_LONGLONG**, **TypeCode.TC_NULL**, **TypeCode.TC_OCTET**, **TypeCode.TC_SHORT**, **TypeCode.TC_ULONG**, **TypeCode.TC_ULONGLONG**, **TypeCode.TC_USHORT**, **TypeCode.TC_WCHAR**, **TCKind.TK_BOOLEAN**, **TCKind.TK_CHAR**, **TCKind.TK_DOUBLE**, **TCKind.TK_FLOAT**, **TCKind.TK_LONG**, **TCKind.TK_LONGDOUBLE**, **TCKind.TK_LONGLONG**, **TCKind.TK_NULL**, **TCKind.TK_OCTET**, **TCKind.TK_SHORT**, **TCKind.TK_ULONG**, **TCKind.TK_ULONGLONG**, **TCKind.TK_USHORT**, and **TCKind.TK_WCHAR**.

8.388 TypeConsistencyEnforcementQosPolicy Class Reference

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Inheritance diagram for TypeConsistencyEnforcementQosPolicy:



Public Attributes

- **TypeConsistencyKind kind**

  Type consistency kind.
- boolean **ignore_sequence_bounds**

  Controls whether sequence bounds are taken into consideration for type assignability.
- boolean **ignore_string_bounds**

  Controls whether string bounds are taken into consideration for type assignability.
- boolean **ignore_member_names**

  Controls whether member names are taken into consideration for type assignability.
- boolean **prevent_type_widening**

  Controls whether type widening is allowed.
- boolean **force_type_validation**

  *Controls whether type information must be available in order to complete matching between a **com.rti.dds.publication.DataWriter** (p. 553) and a **com.rti.dds.subscription.DataReader** (p. 450).*
- boolean **ignore_enum_literal_names**

  Controls whether enumeration constant names are taken into consideration for type assignability.

8.388.1 Detailed Description

Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

This policy defines a type consistency kind, which allows applications to select from among a set of predetermined behaviors. The following consistency kinds are specified: com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.DISALLOW_TYPE_COERCION, com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.ALLOW_TYPE_COERCION and com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.AUTO_TYPE_COERCION.

The type-consistency-enforcement rules consist of two steps:

Step 1. If both the DataWriter and DataReader specify a TypeObject, it is considered first. If the DataReader allows type coercion, then its type must be assignable from the DataWriter's type, taking into account the values of prevent_type_widening, ignore_sequence_bounds, ignore_string_bounds, ignore_member_names, and ignore_enum_literal_names. If the DataReader does not allow type coercion, then its type must be equivalent to the type of the DataWriter.

Step 2. If either the DataWriter or the DataReader does not provide a TypeObject definition, then the registered type names are examined. The DataReader's and DataWriter's registered type names must match exactly, as was true in RTI Connext releases prior to 5.0.0.

If either Step 1 or Step 2 fails, the Topics associated with the DataReader and DataWriter are considered to be inconsistent and the **com.rti.dds.topic.InconsistentTopicStatus** (p. 1149) is updated.

The default enforcement kind is com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.AUTO_TYPE_COERCION. This default kind translates to com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.ALLOW_TYPE_COERCION except in the following cases:

- When a **Zero Copy** (p. 56) DataReader is used, the kind is translated to com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.DISALLOW_TYPE_COERCION.

- When the middleware is introspecting the built-in topic data declaration of a remote DataReader in order to determine whether it can match with a local DataWriter, if it observes that no **TypeConsistencyEnforcement↩ QosPolicy** (p. 1935) value is provided (as would be the case when communicating with a Service implementation not in conformance with this specification), it assumes a kind of com.rti.dds.infrastructure.TypeConsistencyKind.↩ TypeConsistencyKind.DISALLOW_TYPE_COERCION.

For additional information on type consistency enforcement refer to the `Extensible Types Guide` and the `OMG Extensible and Dynamic Topic Types for DDS Specification`.

Entity:

com.rti.dds.subscription.DataReader (p. 450)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.388.2 Member Data Documentation

8.388.2.1 kind

```
TypeConsistencyKind kind
```

Type consistency kind.

[default] com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.AUTO_TYPE_COERCION

8.388.2.2 ignore_sequence_bounds

```
boolean ignore_sequence_bounds
```

Controls whether sequence bounds are taken into consideration for type assignability.

If the option is set to com.rti.dds.infrastructure.true, then sequence bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 sequence type with maximum length L2 would be assignable to a T1 sequence type with maximum length L1, even if L2 is greater than L1. If the option is set to com.rti.dds.infrastructure.↩ false, then sequence bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 >= L2.

[default] com.rti.dds.infrastructure.true

8.388.2.3 ignore_string_bounds

```
boolean ignore_string_bounds
```

Controls whether string bounds are taken into consideration for type assignability.

If the option is set to com.rti.dds.infrastructure.true, then string bounds (maximum lengths) are not considered as part of the type assignability. This means that a T2 string type with maximum length L2 would be assignable to a T1 string type with maximum length L1, even if L2 is greater than L1. If the option is set to com.rti.dds.infrastructure.false, then string bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that L1 >= L2.

[default] com.rti.dds.infrastructure.true

8.388.2.4 ignore_member_names

```
boolean ignore_member_names
```

Controls whether member names are taken into consideration for type assignability.

If the option is set to com.rti.dds.infrastructure.true, then member names are not considered as part of the type assignability. If the option is set to com.rti.dds.infrastructure.false, then member names are taken into consideration for type assignability, and in order for members with the same ID to be assignable, the members must also have the same name.

[default] com.rti.dds.infrastructure.false

8.388.2.5 prevent_type_widening

```
boolean prevent_type_widening
```

Controls whether type widening is allowed.

If the option is set to com.rti.dds.infrastructure.false, then type widening is permitted. If the option is set to com.rti.↩ dds.infrastructure.true, then a wider type may not be assignable from a narrower type.

[default] com.rti.dds.infrastructure.false

8.388.2.6 force_type_validation

```
boolean force_type_validation
```

Controls whether type information must be available in order to complete matching between a **com.rti.dds.↩ publication.DataWriter** (p. 553) and a **com.rti.dds.subscription.DataReader** (p. 450).

If the option is set to com.rti.dds.infrastructure.true, then type information must be available in order to complete matching between a **com.rti.dds.publication.DataWriter** (p. 553) and a **com.rti.dds.subscription.DataReader** (p. 450). If the option is set to com.rti.dds.infrastructure.false, then matching can occur without complete type information as long as the type names match exactly. Note that if the types have the same name but are not assignable, DataReaders may fail to deserialize incoming data samples.

[default] com.rti.dds.infrastructure.false

8.388.2.7 ignore_enum_literal_names

```
boolean ignore_enum_literal_names
```

Controls whether enumeration constant names are taken into consideration for type assignability.

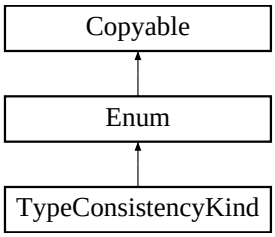
If the option is set to com.rti.dds.infrastructure.true, then enumeration constants may change their names, but not their values, and still maintain assignability. If the option is set to com.rti.dds.infrastructure.false, then in order for enumerations to be assignable, any constant that has the same value in both enumerations must also have the same name.

[default] com.rti.dds.infrastructure.false

8.389 TypeConsistencyKind Class Reference

Kinds of type consistency.

Inheritance diagram for TypeConsistencyKind:



Static Public Attributes

- static final **TypeConsistencyKind DISALLOW_TYPE_COERCION**

  The DataWriter and the DataReader must support the same data type in order for them to communicate.
- static final **TypeConsistencyKind ALLOW_TYPE_COERCION**

  The DataWriter and the DataReader need not support the same data type in order for them to communicate as long as the DataReader's type is assignable from the DataWriter's type. For example, the following two extensible types will be assignable to each other since MyDerivedType contains all the members of MyBaseType (member_1) plus some additional elements (member_2).
- static final **TypeConsistencyKind AUTO_TYPE_COERCION**

  This AUTO value will be applied as com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.DISALLOW_↩ TYPE_COERCION when the data type is annotated with @transfer_mode(SHMEM_REF) and the @language_binding is PLAIN (default) while using C, Traditional C++, or Modern C++ APIs. In all other cases, this AUTO value will be applied as com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.ALLOW_TYPE_COERCION.

Additional Inherited Members

8.389.1 Detailed Description

Kinds of type consistency.

QoS:

com.rti.dds.infrastructure.TypeConsistencyEnforcementQosPolicy (p. 1935)

8.389.2 Member Data Documentation

8.389.2.1 DISALLOW_TYPE_COERCION

```
final TypeConsistencyKind DISALLOW_TYPE_COERCION [static]
```

The DataWriter and the DataReader must support the same data type in order for them to communicate.

This is the degree of type consistency enforcement required by the OMG DDS Specification prior to the OMG Extensible and Dynamic Topic Types for DDS Specification.

8.389.2.2 ALLOW_TYPE_COERCION

```
final TypeConsistencyKind ALLOW_TYPE_COERCION [static]
```

Initial value:

```
=
        new TypeConsistencyKind("ALLOW_TYPE_COERCION", 1)
```

The DataWriter and the DataReader need not support the same data type in order for them to communicate as long as the DataReader's type is assignable from the DataWriter's type. For example, the following two extensible types will be assignable to each other since MyDerivedType contains all the members of MyBaseType (member_1) plus some additional elements (member_2).

```
struct MyBaseType {
 long member_1;
 };

 struct MyDerivedType: MyBaseType {
 long member_2;
 };
```

Even if MyDerivedType was not explicitly inheriting from MyBaseType the types would still be assignable. For example:

```
struct MyBaseType {
 long member_1;
 };

 struct MyDerivedType {
 long member_1;
 long member_2;
 };
```

For additional information on type assignability refer to the OMG Extensible and Dynamic Topic Types for DDS Specification.

8.389.2.3 AUTO_TYPE_COERCION

```
final TypeConsistencyKind AUTO_TYPE_COERCION [static]
```

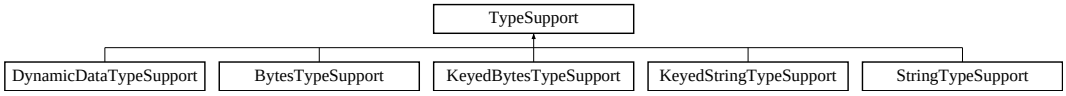
This AUTO value will be applied as com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.DISALLOW↩
_TYPE_COERCION when the data type is annotated with @transfer_mode(SHMEM_REF) and the @language_binding is PLAIN (default) while using C, Traditional C++, or Modern C++ APIs. In all other cases, this AUTO value will be applied as com.rti.dds.infrastructure.TypeConsistencyKind.TypeConsistencyKind.ALLOW_TYPE_COERCION.

[default]

8.390 TypeSupport Interface Reference

<<***interface***>> *(*p. *156)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application.

Inheritance diagram for TypeSupport:



8.390.1 Detailed Description

<<***interface***>> *(*p. *156)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application.

The implementation provides an automatic means to generate a type-specific class, **com.rti.ndds.example.FooType↩ Support** (p. 1118), from a description of the type in IDL.

A **com.rti.dds.topic.TypeSupport** (p. 1941) must be registered using the **com.rti.ndds.example.FooTypeSupport.↩ register_type** (p. 1119) operation on this type-specific class before it can be used to create **com.rti.dds.topic.Topic** (p. 1807) objects.

See also

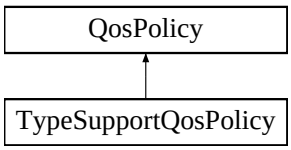
com.rti.ndds.example.FooTypeSupport (p. 1118)

the `Code Generator User's Manual`

8.391 TypeSupportQosPolicy Class Reference

Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.↩ subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

Inheritance diagram for TypeSupportQosPolicy:



Public Attributes

- transient Object **plugin_data**

  Value to pass into the type plugin's de-/serialization function.

8.391.1 Detailed Description

Allows you to attach application-specific values to a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.↩ subscription.DataReader** (p. 450), which are passed to the serialization or deserialization routine of the associated data type and choose whether to set padding bytes to zero during serialization.

The purpose of this QoS is to allow a user application to pass data to a type plugin's support functions and choose whether or not to set the padding bytes to zero when serializing a sample using CDR encapsulation.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.↩ dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO

Changeable (p. 256) = **UNTIL ENABLE** (p. 256)

8.391.2 Usage

The **com.rti.dds.infrastructure.TypeSupportQosPolicy.plugin_data** (p. 1943) allows you to associate a pointer to an object with a **com.rti.dds.publication.DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450). This object pointer is passed to the serialization routine of the data type associated with the **com.rti.dds.publication.Data↩ Writer** (p. 553) or the deserialization routine of the data type associated with the **com.rti.dds.subscription.DataReader** (p. 450).

You can modify the rtiddsgen-generated code so that the de/serialization routines act differently depending on the information passed in via the object pointer. (The generated serialization and deserialization code does not use the pointer.)

This functionality can be used to change how data sent by a **com.rti.dds.publication.DataWriter** (p. 553) or received by a **com.rti.dds.subscription.DataReader** (p. 450) is serialized or deserialized on a per DataWriter and DataReader basis.

It can also be used to dynamically change how serialization (or for a less common case, deserialization) occurs. For example, a data type could represent a table, including the names of the rows and columns. However, since the row/column names of an instance of the table (a Topic) don't change, they only need to be sent once. The information passed in through the TypeSupport QoS policy could be used to signal the serialization routine to send the row/column names the first time a **com.rti.dds.publication.DataWriter** (p. 553) calls **com.rti.ndds.example.FooDataWriter.write** (p. 1105), and then never again.

The com.rti.dds.infrastructure.TypeSupportQosPolicy.cdr_padding_kind allows you to choose whether or not the padding bytes are set to zero during CDR serialization.

8.391.3 Member Data Documentation

8.391.3.1 plugin_data

```
transient Object plugin_data
```

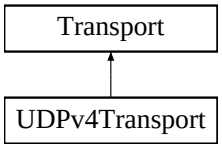
Value to pass into the type plugin's de-/serialization function.

[default] NULL

8.392 UDPv4Transport Interface Reference

Transport (p. 1841) plug-in using UDP/IPv4.

Inheritance diagram for UDPv4Transport:



Classes

- class **Property_t**

  Configurable IPv4/UDP Transport-Plugin properties.

Static Public Attributes

- static final int **CLASSID** = 1
- static final int **BLOCKING_NEVER**

  *Value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_blocking*** (p. *1414*) *to specify non-blocking sockets.*
- static final int **BLOCKING_ALWAYS**

  [default]** Value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_blocking (p. *1414*) *to specify blocking sockets.*
- static final int **UDPV4_PAYLOAD_SIZE_MAX**

  *Maximum value for **com.rti.ndds.transport.Transport.Property_t.message_size_max*** (p. *1404*) *for the UDPv4 transport.*
- static final int **MESSAGE_SIZE_MAX_DEFAULT**

  *Default value for **com.rti.ndds.transport.Transport.Property_t.message_size_max*** (p. *1404*).
- static final int **SOCKET_BUFFER_SIZE_OS_DEFAULT** = -1

  Used to specify that os default be used to specify socket buffer size.
- static final int **SEND_SOCKET_BUFFER_SIZE_DEFAULT** = 131072

  *Default value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_socket_buffer_size*** (p. *1410*) *and **com.↩rti.ndds.transport.UDPv4WanTransport.Property_t.send_socket_buffer_size*** (p. *1422*).
- static final int **RECV_SOCKET_BUFFER_SIZE_DEFAULT** = 131072

  *Default value for **com.rti.ndds.transport.UDPv4Transport.Property_t.recv_socket_buffer_size*** (p. *1410*) *and **com.↩rti.ndds.transport.UDPv4WanTransport.Property_t.recv_socket_buffer_size*** (p. *1422*).

8.392.1 Detailed Description

Transport (p. 1841) plug-in using UDP/IPv4.

This transport plugin uses UDPv4 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **com.rti.dds.infrastructure.TransportBuiltinKind.UDPv4_ALIAS** (p. 270).

You can configure an instance of this plugin to only use unicast or only use multicast, see **com.rti.ndds.transport.↩ UDPv4Transport.Property_t.unicast_enabled** (p. 1410) and **com.rti.ndds.transport.UDPv4Transport.Property_↩ t.multicast_enabled** (p. 1411).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **com.rti.ndds.transport.Transport.↩ Property_t.max_interface_count** (p. 1406) and the "white" and "black" lists in the base property's fields (**com.↩ rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404), **com.rti.ndds.transport.Transport.↩ Property_t.deny_interfaces_list** (p. 1405), **com.rti.ndds.transport.Transport.Property_t.allow_multicast_↩ interfaces_list** (p. 1405), **com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list** (p. 1406)).

RTI Connext can implicitly create this plugin and register with the **com.rti.dds.domain.DomainParticipant** (p. 670) if this transport is specified in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843).

To specify the properties of the builtin UDPv4 transport that is implicitly registered, you can either:

- call **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property** (p. 1863) or
- specify the predefined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipant** (p. 670). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1944)). Builtin transport plugin properties specified in **com.rti.↩ dds.infrastructure.PropertyQosPolicy** (p. 1438) always overwrite the ones specified through **com.rti.ndds.↩ transport.TransportSupport.set_builtin_transport_property()** (p. 1863). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

8.392.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **com.rti.dds.domain.Domain↩ ParticipantQos.property** (p. 801) to configure the builtin UDPv4 transport plugin.

Table 8.1194 ***Property Names for UDPv4 Transport (*** p. ***1841) Plugin***

Property Name	Description
dds.transport.UDPv4.builtin.parent.classid	See **com.rti.ndds.transport.Transport.Property_t.↩ classid** (p. 1403)
dds.transport.UDPv4.builtin.parent.address_bit_count	See **com.rti.ndds.transport.Transport.Property_t.↩ address_bit_count** (p. 1403)

Property Name	Description
dds.transport.UDPv4.builtin.parent.properties_bitmap	See **com.rti.ndds.transport.Transport.Property_t.properties_bitmap** (p. 1403)
dds.transport.UDPv4.builtin.parent.gather_send_buffer_count_max	See **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404)
dds.transport.UDPv4.builtin.parent.message_size_max	See **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404)
dds.transport.UDPv4.builtin.parent.allow_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404) and com.rti.ndds.transport.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
dds.transport.UDPv4.builtin.parent.deny_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405) and com.rti.ndds.transport.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
dds.transport.UDPv4.builtin.parent.allow_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list** (p. 1405) and com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list** (p. 1406) and com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.max_interface_count	See **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406)
dds.transport.UDPv4.builtin.parent.thread_name_prefix	See com.rti.ndds.transport.Transport.Property_t.thread_name_prefix
dds.transport.UDPv4.builtin.send_socket_buffer_size	See **com.rti.ndds.transport.UDPv4Transport.Property_t.send_socket_buffer_size** (p. 1410)
dds.transport.UDPv4.builtin.recv_socket_buffer_size	See **com.rti.ndds.transport.UDPv4Transport.Property_t.recv_socket_buffer_size** (p. 1410)
dds.transport.UDPv4.builtin.unicast_enabled	See **com.rti.ndds.transport.UDPv4Transport.Property_t.unicast_enabled** (p. 1410)
dds.transport.UDPv4.builtin.multicast_enabled	See **com.rti.ndds.transport.UDPv4Transport.Property_t.multicast_enabled** (p. 1411)
dds.transport.UDPv4.builtin.multicast_ttl	See **com.rti.ndds.transport.UDPv4Transport.Property_t.multicast_ttl** (p. 1411)
dds.transport.UDPv4.builtin.multicast_loopback_disabled	See **com.rti.ndds.transport.UDPv4Transport.Property_t.multicast_loopback_disabled** (p. 1411)

Property Name	Description
dds.transport.UDPv4.builtin.ignore_loopback_interface	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.ignore_loopback_interface** (p. 1412)
dds.transport.UDPv4.builtin.ignore_nonrunning_↩ interfaces	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.ignore_nonrunning_interfaces** (p. 1413)
dds.transport.UDPv4.builtin.ignore_nonup_interfaces	**[DEPRECATED]** See **com.rti.ndds.transport.↩ UDPv4Transport.Property_t.ignore_nonup_↩ interfaces** (p. 1412)
dds.transport.UDPv4.builtin.no_zero_copy	**[DEPRECATED]** See **com.rti.ndds.transport.↩ UDPv4Transport.Property_t.no_zero_copy** (p. 1413)
dds.transport.UDPv4.builtin.send_blocking	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.send_blocking** (p. 1414)
dds.transport.UDPv4.builtin.transport_priority_mask	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.transport_priority_mask** (p. 1415)
dds.transport.UDPv4.builtin.transport_priority_↩ mapping_low	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.transport_priority_mapping_low** (p. 1415)
dds.transport.UDPv4.builtin.transport_priority_↩ mapping_high	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.transport_priority_mapping_high** (p. 1415)
dds.transport.UDPv4.builtin.send_ping	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.send_ping** (p. 1416)
dds.transport.UDPv4.builtin.force_interface_poll_↩ detection	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.force_interface_poll_detection** (p. 1416)
dds.transport.UDPv4.builtin.interface_poll_period	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.interface_poll_period** (p. 1416)
dds.transport.UDPv4.builtin.reuse_multicast_receive_↩ resource	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.reuse_multicast_receive_resource** (p. 1417)
dds.transport.UDPv4.builtin.protocol_overhead_max	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.protocol_overhead_max** (p. 1417)
dds.transport.UDPv4.builtin.disable_interface_tracking	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.disable_interface_tracking** (p. 1417)
dds.transport.UDPv4.builtin.public_address	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.public_address** (p. 1418)
dds.transport.UDPv4.builtin.join_multicast_group_↩ timeout	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.join_multicast_group_timeout** (p. 1418)
dds.transport.UDPv4.builtin.port_offset	See **com.rti.ndds.transport.UDPv4Transport.↩ Property_t.port_offset** (p. 1419)

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.392.3 Member Data Documentation

8.392.3.1 CLASSID

```
final int CLASSID = 1  [static]
```

The UDPv4 **Transport** (p. 1841) class id

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.392.3.2 BLOCKING_NEVER

```
final int BLOCKING_NEVER  [static]
```

Value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_blocking** (p. 1414) to specify non-blocking sockets.

8.392.3.3 BLOCKING_ALWAYS

```
final int BLOCKING_ALWAYS  [static]
```

[default] Value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_blocking** (p. 1414) to specify blocking sockets.

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.392.3.4 UDPV4_PAYLOAD_SIZE_MAX

```
final int UDPV4_PAYLOAD_SIZE_MAX  [static]
```

Maximum value for **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404) for the UDPv4 transport.

8.392.3.5 MESSAGE_SIZE_MAX_DEFAULT

```
final int MESSAGE_SIZE_MAX_DEFAULT  [static]
```

Default value for **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404).

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.392.3.6 SOCKET_BUFFER_SIZE_OS_DEFAULT

```
final int SOCKET_BUFFER_SIZE_OS_DEFAULT = -1  [static]
```

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

8.392.3.7 SEND_SOCKET_BUFFER_SIZE_DEFAULT

```
final int SEND_SOCKET_BUFFER_SIZE_DEFAULT = 131072  [static]
```

Default value for **com.rti.ndds.transport.UDPv4Transport.Property_t.send_socket_buffer_size** (p. 1410) and **com.rti.ndds.transport.UDPv4WanTransport.Property_t.send_socket_buffer_size** (p. 1422).

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.392.3.8 RECV_SOCKET_BUFFER_SIZE_DEFAULT

```
final int RECV_SOCKET_BUFFER_SIZE_DEFAULT = 131072  [static]
```

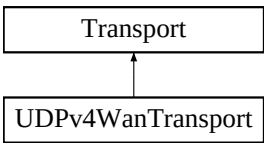
Default value for **com.rti.ndds.transport.UDPv4Transport.Property_t.recv_socket_buffer_size** (p. 1410) and **com.rti.ndds.transport.UDPv4WanTransport.Property_t.recv_socket_buffer_size** (p. 1422).

Referenced by **UDPv4Transport.Property_t.Property_t()**.

8.393 UDPv4WanTransport Interface Reference

Transport (p. 1841) plug-in using UDP/IPv4 for WAN communications..

Inheritance diagram for UDPv4WanTransport:



Classes

- class **Property_t**

  Configurable IPv4/UDP WAN Transport-Plugin properties.

8.393.1 Detailed Description

Transport (p. 1841) plug-in using UDP/IPv4 for WAN communications..

RTI Real-Time WAN **Transport** (p. 1841) (RWT) is a transport that enables secure, scalable, and high-performance communication over wide area networks (WANs), including public networks.

It extends RTI Connext capabilities to WAN environments. Real-Time WAN **Transport** (p. 1841) uses UDPv4 as the underlying IP transport-layer protocol to better anticipate and adapt to the challenges of diverse network conditions, device mobility, and the dynamic nature of WAN system architectures.

Real-Time WAN **Transport** (p. 1841), in combination with RTI Cloud Discovery Service (CDS), provides a complete, seamless solution out of the box for WAN connectivity.

This transport is not installed as part of an RTI Connext package; it must be downloaded and installed separately.

Real-Time WAN **Transport** (p. 1841) replaces the transport capabilities of the Secure WAN **Transport** (p. 1841) optionally available with previous RTI Connext releases (prior to 7.0.0), and provides the following capabilities:

- NAT (Network Address Translator) traversal: Ability to communicate between DomainParticipants running in a Local Area Network (LAN) that is behind a NAT-enabled router, and DomainParticipants on the outside of the NAT across a WAN. This functionality is provided in combination with Cloud Discovery Service.
- IP mobility: Support for network transitions and changes in IP addresses in any of the DomainParticipants participating in the communication.
- Security: Secure communications between DomainParticipants using Security Plugins.

Real-Time WAN **Transport** (p. 1841) does not require third-party components, such as STUN servers, or protocols like SIP to handle session establishment. Using a single API and security model, you can leverage the extensive capabilities of the RTI Connext framework and ecosystem, including tools and infrastructure services, even for real-time connectivity from edge to cloud and back in highly distributed systems that communicate across wide area networks.

This transport plugin uses UDPv4 sockets to send and receive messages. It supports unicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **com.rti.↩ dds.infrastructure.TransportBuiltinKind.UDPv4_WAN_ALIAS** (p. 271).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node by specifying the **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406) and the "white" and "black" lists in the base property's fields (**com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404), **com.rti.↩ ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405)).

RTI Connext can implicitly create this plugin and register with the **com.rti.dds.domain.DomainParticipant** (p. 670) if this transport is specified in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843).

To specify the properties of the Real-Time WAN **Transport** (p. 1841) that is implicitly registered, you can either:

- call **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property** (p. 1863) or

- specify the predefined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipant** (p. 670). (see **Real-Time WAN Transport Property** (p. 1950)). Builtin transport plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) always overwrite the ones specified through **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property()** (p. 1863). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

For additional details on how to configure and use the Real-Time WAN Transport (p. 1841), see the `Core Libraries User's Manual`.

8.393.2 Real-Time WAN Transport Property

Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in the **com.rti.dds.domain.DomainParticipantQos.property** (p. 801) to configure the Real-Time WAN **Transport** (p. 1841) plugin.

Table 8.1195 ***Property Names for Real-Time WAN Transport (p. 1841) Plugin***

Property Name	Description
dds.transport.UDPv4_WAN.builtin.parent.classid	See **com.rti.ndds.transport.Transport.Property_t.classid** (p. 1403)
dds.transport.UDPv4_WAN.builtin.parent.address_bit_count	See **com.rti.ndds.transport.Transport.Property_t.address_bit_count** (p. 1403)
dds.transport.UDPv4_WAN.builtin.parent.properties_bitmap	See **com.rti.ndds.transport.Transport.Property_t.properties_bitmap** (p. 1403)
dds.transport.UDPv4_WAN.builtin.parent.gather_send_buffer_count_max	See **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404)
dds.transport.UDPv4_WAN.builtin.parent.message_size_max	See **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404)
dds.transport.UDPv4_WAN.builtin.parent.allow_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404) and com.rti.ndds.transport.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
dds.transport.UDPv4_WAN.builtin.parent.deny_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405) and com.rti.ndds.transport.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
dds.transport.UDPv4_WAN.builtin.parent.max_interface_count	See **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406)

Property Name	Description
dds.transport.UDPv4_WAN.builtin.parent.thread_↩ name_prefix	See com.rti.ndds.transport.Transport.Property_t.↩ thread_name_prefix
dds.transport.UDPv4_WAN.builtin.send_socket_↩ buffer_size	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.send_socket_buffer_size** (p. 1422)
dds.transport.UDPv4_WAN.builtin.recv_socket_buffer↩ _size	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.recv_socket_buffer_size** (p. 1422)
dds.transport.UDPv4_WAN.builtin.ignore_loopback_↩ interface	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.ignore_loopback_interface** (p. 1422)
dds.transport.UDPv4_WAN.builtin.ignore_nonrunning↩ _interfaces	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.ignore_nonrunning_interfaces** (p. 1423)
dds.transport.UDPv4_WAN.builtin.ignore_nonup_↩ interfaces	**[DEPRECATED]** See **com.rti.ndds.transport.↩ UDPv4WanTransport.Property_t.ignore_nonup_↩ interfaces** (p. 1422)
dds.transport.UDPv4_WAN.builtin.no_zero_copy	**[DEPRECATED]** See **com.rti.ndds.transport.↩ UDPv4WanTransport.Property_t.no_zero_copy** (p. 1423)
dds.transport.UDPv4_WAN.builtin.send_blocking	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.send_blocking** (p. 1424)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mask	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.transport_priority_mask** (p. 1425)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_low	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.transport_priority_mapping_low** (p. 1425)
dds.transport.UDPv4_WAN.builtin.transport_priority_↩ mapping_high	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.transport_priority_mapping_high** (p. 1425)
dds.transport.UDPv4_WAN.builtin.send_ping	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.send_ping** (p. 1426)
dds.transport.UDPv4_WAN.builtin.force_interface_↩ poll_detection	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.force_interface_poll_detection** (p. 1426)
dds.transport.UDPv4_WAN.builtin.interface_poll_period	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.interface_poll_period** (p. 1426)
dds.transport.UDPv4_WAN.builtin.protocol_overhead↩ _max	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.protocol_overhead_max** (p. 1427)
dds.transport.UDPv4_WAN.builtin.disable_interface_↩ tracking	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.disable_interface_tracking** (p. 1427)
dds.transport.UDPv4_WAN.builtin.public_address	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.public_address** (p. 1427)
dds.transport.UDPv4_WAN.builtin.comm_ports	See com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.comm_ports_list
dds.transport.UDPv4_WAN.builtin.port_offset	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.port_offset** (p. 1429)
dds.transport.UDPv4_WAN.builtin.binding_ping_period	See **com.rti.ndds.transport.UDPv4WanTransport.↩ Property_t.binding_ping_period** (p. 1429)

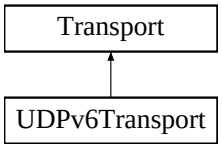
See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.394 UDPv6Transport Interface Reference

Transport (p. 1841) plug-in using UDP/IPv6.

Inheritance diagram for UDPv6Transport:



Classes

- class **Property_t**

  Configurable IPv6/UDP Transport-Plugin properties.

Static Public Attributes

- static final int **CLASSID** = 2
- static final int **BLOCKING_NEVER**

  *Value for **com.rti.ndds.transport.UDPv6Transport.Property_t.send_blocking*** (p. *1435) to specify non-blocking sockets.*
- static final int **BLOCKING_ALWAYS**

  [default]** Value for **com.rti.ndds.transport.UDPv6Transport.Property_t.send_blocking (p. *1435) to specify blocking sockets.*

8.394.1 Detailed Description

Transport (p. 1841) plug-in using UDP/IPv6.

This transport plugin uses UDPv6 sockets to send and receive messages. It supports both unicast and multicast communications in a single instance of the plugin. By default, this plugin will use all interfaces that it finds enabled and "UP" at instantiation time to send and receive messages. This transport is installed as a built-in transport plugin with the alias **com.rti.dds.infrastructure.TransportBuiltinKind.UDPv6_ALIAS** (p. 271).

You can configure an instance of this plugin to only use unicast or only use multicast, see **com.rti.ndds.transport.↩ UDPv6Transport.Property_t.unicast_enabled** (p. 1433) and **com.rti.ndds.transport.UDPv6Transport.Property_↩ t.multicast_enabled** (p. 1433).

In addition, you can configure an instance of this plugin to selectively use the network interfaces of a node (and restrict a plugin from sending multicast messages on specific interfaces) by specifying the **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406) and the "white" and "black" lists in the base property's fields (**com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404), **com.rti.ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405), **com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list** (p. 1405), **com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list** (p. 1406)).

RTI Connext can implicitly create this plugin and register it with the **com.rti.dds.domain.DomainParticipant** (p. 670) if this transport is specified in the **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1843).

To specify the properties of the builtin UDPv6 transport that is implicitly registered, you can either:

- call **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property** (p. 1863) or
- specify the predefined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) associated with the **com.rti.dds.domain.DomainParticipant** (p. 670). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1953)). Builtin transport plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) always overwrite the ones specified through **com.rti.ndds.transport.TransportSupport.set_builtin_transport_property()** (p. 1863). The default value is assumed on any unspecified property.

Note that all properties should be set before the transport is implicitly created and registered by RTI Connext. Any properties that are set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 102) for details on when a builtin transport is registered.

8.394.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1438) of a **com.rti.dds.domain.DomainParticipant** (p. 670) to configure the builtin UDPv6 transport plugin.

Table 8.1196 ***Property Names for UDPv6 Transport (*** p. ***1841) Plugin***

Property Name	Description
dds.transport.UDPv6.builtin.parent.address_bit_count	See **com.rti.ndds.transport.Transport.Property_t.address_bit_count** (p. 1403)
dds.transport.UDPv6.builtin.parent.properties_bitmap	See **com.rti.ndds.transport.Transport.Property_t.properties_bitmap** (p. 1403)
dds.transport.UDPv6.builtin.parent.gather_send_buffer_count_max	See **com.rti.ndds.transport.Transport.Property_t.gather_send_buffer_count_max** (p. 1404)
dds.transport.UDPv6.builtin.parent.message_size_max	See **com.rti.ndds.transport.Transport.Property_t.message_size_max** (p. 1404)
dds.transport.UDPv6.builtin.parent.allow_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_interfaces_list** (p. 1404) and com.rti.ndds.transport.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

Property Name	Description
dds.transport.UDPv6.builtin.parent.deny_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_interfaces_list** (p. 1405) and com.rti.ndds.transport.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
dds.transport.UDPv6.builtin.parent.allow_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list** (p. 1405) and com.rti.ndds.transport.Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_multicast_interfaces	See **com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list** (p. 1406) and com.rti.ndds.transport.Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.max_interface_count	See **com.rti.ndds.transport.Transport.Property_t.max_interface_count** (p. 1406)
dds.transport.UDPv6.builtin.parent.thread_name_prefix	See com.rti.ndds.transport.Transport.Property_t.thread_name_prefix
dds.transport.UDPv6.builtin.send_socket_buffer_size	See **com.rti.ndds.transport.UDPv6Transport.Property_t.send_socket_buffer_size** (p. 1432)
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See **com.rti.ndds.transport.UDPv6Transport.Property_t.recv_socket_buffer_size** (p. 1432)
dds.transport.UDPv6.builtin.unicast_enabled	See **com.rti.ndds.transport.UDPv6Transport.Property_t.unicast_enabled** (p. 1433)
dds.transport.UDPv6.builtin.multicast_enabled	See **com.rti.ndds.transport.UDPv6Transport.Property_t.multicast_enabled** (p. 1433)
dds.transport.UDPv6.builtin.multicast_ttl	See **com.rti.ndds.transport.UDPv6Transport.Property_t.multicast_ttl** (p. 1433)
dds.transport.UDPv6.builtin.multicast_loopback_disabled	See **com.rti.ndds.transport.UDPv6Transport.Property_t.multicast_loopback_disabled** (p. 1434)
dds.transport.UDPv6.builtin.ignore_loopback_interface	See **com.rti.ndds.transport.UDPv6Transport.Property_t.ignore_loopback_interface** (p. 1434)
dds.transport.UDPv6.builtin.ignore_nonrunning_interfaces	See **com.rti.ndds.transport.UDPv6Transport.Property_t.ignore_nonrunning_interfaces** (p. 1434)
dds.transport.UDPv6.builtin.no_zero_copy	**[DEPRECATED]** See **com.rti.ndds.transport.UDPv6Transport.Property_t.no_zero_copy** (p. 1435)
dds.transport.UDPv6.builtin.send_blocking	See **com.rti.ndds.transport.UDPv6Transport.Property_t.send_blocking** (p. 1435)
dds.transport.UDPv6.builtin.enable_v4mapped	See **com.rti.ndds.transport.UDPv6Transport.Property_t.enable_v4mapped** (p. 1436)
dds.transport.UDPv6.builtin.transport_priority_mask	See **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mask** (p. 1436)
dds.transport.UDPv6.builtin.transport_priority_mapping_low	See **com.rti.ndds.transport.UDPv6Transport.Property_t.transport_priority_mapping_low** (p. 1436)

Property Name	Description
dds.transport.UDPv6.builtin.transport_priority_↩ mapping_high	See **com.rti.ndds.transport.UDPv6Transport.↩ Property_t.transport_priority_mapping_high** (p. 1437)
dds.transport.UDPv6.builtin.send_ping	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.send_ping interface_poll_detection_kind
dds.transport.UDPv6.builtin.force_interface_poll_↩ detection	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.force_interface_poll_detection
dds.transport.UDPv6.builtin.interface_poll_period	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.interface_poll_period
dds.transport.UDPv6.builtin.reuse_multicast_receive_↩ resource	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.reuse_multicast_receive_resource
dds.transport.UDPv6.builtin.protocol_overhead_max	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.protocol_overhead_max
dds.transport.UDPv6.builtin.disable_interface_tracking	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.disable_interface_tracking
dds.transport.UDPv6.builtin.public_address	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.public_address
dds.transport.UDPv6.builtin.join_multicast_group_↩ timeout	See com.rti.ndds.transport.UDPv6Transport.Property↩ _t.join_multicast_group_timeout
dds.transport.UDPv6.builtin.port_offset	See **com.rti.ndds.transport.UDPv6Transport.↩ Property_t.port_offset** (p. 1437)

See also

com.rti.ndds.transport.TransportSupport.set_builtin_transport_property() (p. 1863)

8.394.3 Member Data Documentation

8.394.3.1 CLASSID

```
final int CLASSID = 2  [static]
```

The UDPv6 **Transport** (p. 1841) class id

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.394.3.2 BLOCKING_NEVER

```
final int BLOCKING_NEVER  [static]
```

Value for **com.rti.ndds.transport.UDPv6Transport.Property_t.send_blocking** (p. 1435) to specify non-blocking sockets.

8.394.3.3 BLOCKING_ALWAYS

```
final int BLOCKING_ALWAYS  [static]
```

[default] Value for **com.rti.ndds.transport.UDPv6Transport.Property_t.send_blocking** (p. 1435) to specify blocking sockets.

Referenced by **UDPv6Transport.Property_t.Property_t()**.

8.395 Union Class Reference

Inherited by TypeId.

8.395.1 Detailed Description

Base class for all generated unions. The purpose of this class is to recognize a nddsgen generated union using reflection

Author

jaime_c

8.396 UnionMember Class Reference

A description of a member of a union.

Inherits Serializable.

Public Member Functions

- **UnionMember** (String **name**, boolean **is_pointer**, int[ ] **labels**, **TypeCode type**)

Public Attributes

- String **name**

  The name of the union member.
- boolean **is_pointer**

  Indicates whether the union member is a pointer or not.
- int[ ] **labels**

  The labels of the union member.
- **TypeCode type**

  The type of the union member.
- int **id**

  The member ID.

8.396.1 Detailed Description

A description of a member of a union.

See also

com.rti.dds.typecode.TypeCodeFactory.create_union_tc (p. 1927)

8.396.2 Constructor & Destructor Documentation

8.396.2.1 UnionMember()

```
UnionMember (
            String name,
            boolean is_pointer,
            int[] labels,
             TypeCode type )
```

Constructs a **UnionMember** (p. 1956) object initialized with the given values.

References **UnionMember.is_pointer**, **UnionMember.labels**, **TypeCode.MEMBER_ID_INVALID**, **Union↩ Member.name**, and **UnionMember.type**.

8.396.3 Member Data Documentation

8.396.3.1 name

```
String name
```

The name of the union member.

Cannot be null.

Referenced by **TypeCode.member_name()**, and **UnionMember.UnionMember()**.

8.396.3.2 is_pointer

```
boolean is_pointer
```

Indicates whether the union member is a pointer or not.

Referenced by **TypeCode.is_member_pointer()**, and **UnionMember.UnionMember()**.

8.396.3.3 labels

```
int [] labels
```

The labels of the union member.

Each union member should contain at least one label. If the union discriminator type is not com.rti.dds.infrastructure.int the label value should be evaluated to an integer value. For instance, 'a' would be evaluated to 97.

Referenced by **TypeCode.member_label()**, **TypeCode.member_label_count()**, and **UnionMember.Union↩ Member()**.

8.396.3.4 type

```
TypeCode type
```

The type of the union member.

Cannot be null.

Referenced by **TypeCode.member_type()**, and **UnionMember.UnionMember()**.

8.396.3.5 id

```
int id
```

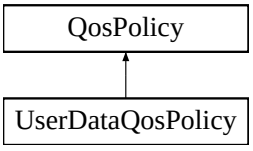
The member ID.

Referenced by **TypeCode.member_id()**.

8.397 UserDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Inheritance diagram for UserDataQosPolicy:



Public Attributes

- final **ByteSeq value**
  a sequence of octets

8.397.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 51) during discovery.

Entity:

com.rti.dds.domain.DomainParticipant (p. 670), **com.rti.dds.subscription.DataReader** (p. 450), **com.rti.dds.publication.DataWriter** (p. 553)

Properties:

RxO (p. 256) = NO;
Changeable (p. 256) = **YES** (p. 256)

See also

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 720)

8.397.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **com.rti.dds.infrastructure.Entity** (p. 1029) objects, so that when a remote application discovers their existence, it can access that information and use it for its own purposes. This information is not used by RTI Connext.

One possible use of this QoS is to attach security credentials or some other information that can be used by the remote application to authenticate the source.

In combination with operations such as **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 723), **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 725), **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 726), and **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 724), this QoS policy can assist an application to define and enforce its own security policies.

The use of this QoS is not limited to security; it offers a simple, yet flexible extensibility mechanism.

Important: RTI Connext stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connext with the maximum size of the data that will be stored in policies of this type. This size is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.participant_user_data_max_length** (p. 815), **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.writer_user_data_max_length** (p. 816), and **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.reader_user_data_max_length** (p. 816).

8.397.3 Member Data Documentation

8.397.3.1 value

```
final ByteSeq value
```

a sequence of octets

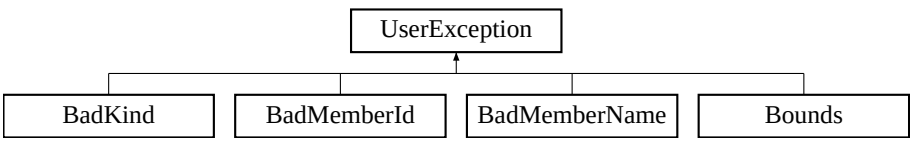
[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

8.398 UserException Class Reference

User exception.

Inheritance diagram for UserException:



8.398.1 Detailed Description

User exception.

This class is based on a similar class in CORBA.

See also

`http://java.sun.com/javase/6/docs/api/org/omg/CORBA/UserException.html`

8.399 Utility Class Reference

Other Utilities APIs.

Static Public Member Functions

- static void **spin** (long spinCount)
  Performs the spin operation as many times as indicated.
- static native long **get_spin_per_microsecond** ()
  Returns the number of spin operations to perform to wait 1 microsecond.
- static native boolean **enable_heap_monitoring** ()
  [DEPRECATED] *See:* ***com.rti.ndds.utility.HeapMonitoring.enable*** *(*p. *1135).*
- static native void **disable_heap_monitoring** ()
  [DEPRECATED] *See:* ***com.rti.ndds.utility.HeapMonitoring.disable*** *(*p. *1136).*
- static native boolean **pause_heap_monitoring** ()
  [DEPRECATED] *See:* ***com.rti.ndds.utility.HeapMonitoring.pause*** *(*p. *1137).*
- static native boolean **resume_heap_monitoring** ()
  [DEPRECATED] *See:* ***com.rti.ndds.utility.HeapMonitoring.resume*** *(*p. *1137)*
- static native boolean **take_heap_snapshot** (String filename, boolean print_details)
  [DEPRECATED] *See:* ***com.rti.ndds.utility.HeapMonitoring.take_heap_snapshot*** *(*p. *1137).*

8.399.1 Detailed Description

Other Utilities APIs.

8.399.2 Member Function Documentation

8.399.2.1 spin()

```
static void spin (
            long spinCount )  [static]
```

Performs the spin operation as many times as indicated.

Spinning is the action of performing useless operations in a for loop in order to actively wait some time without yielding the CPU. Given that the resolution of sleep is in the order of ms, you can use this utility to wait times in the order of microseconds. To properly use this functionality, it is useful to measure previously the number of spin operations needed to wait the equivalent to microsecond (using the utility get_spin_per_microsecond) and then compute the corresponding spin count desired.

Parameters

spinCount	<<***in***>> *(p. 156)* Number of spin operations to perform.

8.399.2.2 get_spin_per_microsecond()

```
static native long get_spin_per_microsecond ( )  [static]
```

Returns the number of spin operations to perform to wait 1 microsecond.

This utility can be used to measure how many spin operations must be performed to wait 1 microsecond. Since the time that it takes the CPU to perform 1 spin operation depends on the CPU frequency, it is recommended to use this utility before using **spin()** (p. 1961).

Returns

Number of spin operations to wait 1 microsecond.

See also

com.rti.ndds.utility.Utility.spin

8.399.2.3 enable_heap_monitoring()

```
static native boolean enable_heap_monitoring ( )  [static]
```

[DEPRECATED] See: **com.rti.ndds.utility.HeapMonitoring.enable** (p. 1135).

8.399.2.4 disable_heap_monitoring()

```
static native void disable_heap_monitoring ( )  [static]
```

[DEPRECATED] See: **com.rti.ndds.utility.HeapMonitoring.disable** (p. 1136).

8.399.2.5 pause_heap_monitoring()

```
static native boolean pause_heap_monitoring ( )  [static]
```

[DEPRECATED] See: **com.rti.ndds.utility.HeapMonitoring.pause** (p. 1137).

8.399.2.6 resume_heap_monitoring()

```
static native boolean resume_heap_monitoring ( )  [static]
```

[DEPRECATED] See: **com.rti.ndds.utility.HeapMonitoring.resume** (p. 1137)

8.399.2.7 take_heap_snapshot()

```
static native boolean take_heap_snapshot (
            String filename,
            boolean print_details )  [static]
```

[DEPRECATED] See: **com.rti.ndds.utility.HeapMonitoring.take_heap_snapshot** (p. 1137).

8.400 ValueMember Class Reference

A description of a member of a value type.

Inherits Serializable.

Public Member Functions

- **ValueMember** (String **name**, boolean **is_pointer**, short **bits**, boolean **is_key**, short **access**, **TypeCode type**)

Public Attributes

- String **name**

  The name of the value member.
- **TypeCode type**

  The type of the value member.
- boolean **is_pointer**

  Indicates whether the value member is a pointer or not.
- short **bits**

  Number of bits of a bitfield member.
- boolean **is_key**

  Indicates if the value member is a key member or not.
- short **access**

  The type of access (public, private) for the value member.
- int **id**

  The member ID.
- boolean **is_optional**

  Indicates if the value member is optional or required.

8.400.1 Detailed Description

A description of a member of a value type.

See also

com.rti.dds.typecode.TypeCodeFactory.create_value_tc (p. 1925)

8.400.2 Constructor & Destructor Documentation

8.400.2.1 ValueMember()

```
ValueMember (
            String name,
            boolean is_pointer,
            short bits,
            boolean is_key,
            short access,
             TypeCode type )
```

Constructs a **ValueMember** (p. 1963) object initialized with the given values.

References **ValueMember.access**, **ValueMember.bits**, **ValueMember.is_key**, **ValueMember.is_pointer**, **TypeCode.MEMBER_ID_INVALID**, **ValueMember.name**, and **ValueMember.type**.

8.400.3 Member Data Documentation

8.400.3.1 name

```
String name
```

The name of the value member.

Cannot be null.

Referenced by **TypeCode.member_name()**, and **ValueMember.ValueMember()**.

8.400.3.2 type

```
TypeCode type
```

The type of the value member.

Cannot be null.

Referenced by **TypeCode.member_type()**, and **ValueMember.ValueMember()**.

8.400.3.3 is_pointer

```
boolean is_pointer
```

Indicates whether the value member is a pointer or not.

Referenced by **TypeCode.is_member_pointer()**, and **ValueMember.ValueMember()**.

8.400.3.4 bits

```
short bits
```

Number of bits of a bitfield member.

If the struct member is a bitfield, this field contains the number of bits of the bitfield. Otherwise, bits should contain **com.rti.dds.typecode.TypeCode.NOT_BITFIELD** (p. 1920).

Referenced by **TypeCode.member_bitfield_bits()**, and **ValueMember.ValueMember()**.

8.400.3.5 is_key

```
boolean is_key
```

Indicates if the value member is a key member or not.

Referenced by **TypeCode.is_member_key()**, and **ValueMember.ValueMember()**.

8.400.3.6 access

```
short access
```

The type of access (public, private) for the value member.

It can take the values: com.rti.dds.typecode.Visibility.PRIVATE_MEMBER or com.rti.dds.typecode.Visibility.PUBLIC_↩ MEMBER.

Referenced by **TypeCode.member_visibility()**, and **ValueMember.ValueMember()**.

8.400.3.7 id

```
int id
```

The member ID.

Use **com.rti.dds.typecode.TypeCode.MEMBER_ID_INVALID** (p. 1918) to have the member ID automatically assigned.

Referenced by **TypeCode.member_id()**.

8.400.3.8 is_optional

```
boolean is_optional
```

Indicates if the value member is optional or required.

Referenced by **TypeCode.is_member_required()**.

8.401 VendorId_t Class Reference

<<***extension***>> *(*p. *155)* Type used to represent the vendor of the service implementing the RTPS protocol.

Inherits Struct.

Public Member Functions

- **VendorId_t** ()
  Constructor.

Public Attributes

- final byte[ ] **vendorId** = new byte[**LENGTH_MAX**]
  The vendor Id.

Static Public Attributes

- static final **VendorId_t UNKNOWN**
  The ID used when the vendor of the service implementing the RTPS protocol is not known.
- static final int **LENGTH_MAX** = 2
  Length of vendor id.

8.401.1 Detailed Description

<<***extension***>> *(*p. *155)* Type used to represent the vendor of the service implementing the RTPS protocol.

8.401.2 Constructor & Destructor Documentation

8.401.2.1 VendorId_t()

```
VendorId_t ( )
```

Constructor.

8.401.3 Member Data Documentation

8.401.3.1 UNKNOWN

```
final  VendorId_t UNKNOWN  [static]
```

The ID used when the vendor of the service implementing the RTPS protocol is not known.

8.401.3.2 LENGTH_MAX

```
final int LENGTH_MAX = 2  [static]
```

Length of vendor id.

8.401.3.3 vendorId

```
final byte [] vendorId = new byte[ LENGTH_MAX]
```

The vendor Id.

8.402 Version Class Reference

<<***interface***>> *(*p. *156)* The version of an RTI Connext distribution.

Public Member Functions

- **ProductVersion_t get_product_version** ()
  Get the RTI Connext product version.
- **LibraryVersion_t get_java_api_version** ()
  Get the version of the Java API library.
- **LibraryVersion_t get_c_api_version** ()
  Get the version of the C API library.
- **LibraryVersion_t get_core_version** ()
  Get the version of the core library.
- String **toString** ()
  Get this version in string form.

Static Public Member Functions

- static **Version get_instance** ()
  Get the singleton instance of this type.

8.402.1 Detailed Description

<<***interface***>> *(p. 156)* The version of an RTI Connext distribution.

The complete version is made up of the versions of the individual libraries that make up the product distribution.

8.402.2 Member Function Documentation

8.402.2.1 get_instance()

```
static  Version get_instance ( )  [static]
```

Get the singleton instance of this type.

8.402.2.2 get_product_version()

```
ProductVersion_t get_product_version ( )
```

Get the RTI Connext product version.

8.402.2.3 get_java_api_version()

```
LibraryVersion_t get_java_api_version ( )
```

Get the version of the Java API library.

8.402.2.4 get_c_api_version()

```
LibraryVersion_t get_c_api_version ( )
```

Get the version of the C API library.

8.402.2.5 get_core_version()

```
LibraryVersion_t get_core_version ( )
```

Get the version of the core library.

8.402.2.6 toString()

```
String toString ( )
```

Get this version in string form.

Combine all of the constituent library versions into a single string.

8.403 ViewStateKind Class Reference

Indicates whether or not an instance is new.

Static Public Attributes

- static final int **NEW_VIEW_STATE** = 0x0001 << 0

  New instance. This latest generation of the instance has not previously been accessed.
- static final int **NOT_NEW_VIEW_STATE** = 0x0001 << 1

  Not a new instance. This latest generation of the instance has previously been accessed.
- static final int **ANY_VIEW_STATE** = 0xffff

  Any view state com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE | com.rti.dds.subscription.↩ ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE.

8.403.1 Detailed Description

Indicates whether or not an instance is new.

For each instance (identified by the key), the middleware internally maintains a view state relative to each **com.rti.↩ dds.subscription.DataReader** (p. 450). The view state can be either:

- com.rti.dds.subscription.ViewStateKind.ViewStateKind.NEW_VIEW_STATE indicates that either this is the first time that the **com.rti.dds.subscription.DataReader** (p. 450) has ever accessed samples of that instance, or else that the **com.rti.dds.subscription.DataReader** (p. 450) has accessed previous samples of the instance, but the instance has since been reborn (i.e. become not-alive and then alive again). These two cases are distinguished by examining the **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1641) and the **com.↩ rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1641).

- com.rti.dds.subscription.ViewStateKind.ViewStateKind.NOT_NEW_VIEW_STATE indicates that the **com.rti.↩ dds.subscription.DataReader** (p. 450) has already accessed samples of the same instance and that the instance has not been reborn since.

The view_state available in the **com.rti.dds.subscription.SampleInfo** (p. 1634) is a snapshot of the view state of the instance relative to the **com.rti.dds.subscription.DataReader** (p. 450) used to access the samples at the time the collection was obtained (i.e. at the time read or take was called). The view_state is therefore the same for all samples in the returned collection that refer to the same instance.

Once an instance has been detected as not having any "live" writers and all the samples associated with the instance are "taken" from the **com.rti.dds.subscription.DataReader** (p. 450), the middleware can reclaim all local resources regarding the instance. Future samples will be treated as "never seen."

8.403.2 Member Data Documentation

8.403.2.1 NEW_VIEW_STATE

```
final int NEW_VIEW_STATE = 0x0001 << 0  [static]
```

New instance. This latest generation of the instance has not previously been accessed.

8.403.2.2 NOT_NEW_VIEW_STATE

```
final int NOT_NEW_VIEW_STATE = 0x0001 << 1  [static]
```

Not a new instance. This latest generation of the instance has previously been accessed.

8.404 VM_ABSTRACT Class Reference

Constant used to indicate that a value type has the `abstract` modifier.

Static Public Attributes

- static final short **VALUE**

8.404.1 Detailed Description

Constant used to indicate that a value type has the `abstract` modifier.

An abstract value type may not be instantiated.

8.404.2 Member Data Documentation

8.404.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

8.405 VM_CUSTOM Class Reference

Constant used to indicate that a value type has the `custom` modifier.

Static Public Attributes

- static final short **VALUE**

8.405.1 Detailed Description

Constant used to indicate that a value type has the `custom` modifier.

This modifier is used to specify whether the value type uses custom marshaling.

8.405.2 Member Data Documentation

8.405.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

8.406 VM_NONE Class Reference

Constant used to indicate that a value type has no modifiers.

Static Public Attributes

- static final short **VALUE**

8.406.1 Detailed Description

Constant used to indicate that a value type has no modifiers.

8.406.2 Member Data Documentation

8.406.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

8.407 VM_TRUNCATABLE Class Reference

Constant used to indicate that a value type has the `truncatable` modifier.

Static Public Attributes

- static final short **VALUE**

8.407.1 Detailed Description

Constant used to indicate that a value type has the `truncatable` modifier.

A value with a state that derives from another value with a state can be declared as truncatable. A truncatable type means the object can be truncated to the base type.

8.407.2 Member Data Documentation

8.407.2.1 VALUE

```
final short VALUE  [static]
```

Constant value.

8.408 WaitSet Class Reference

<<***interface***>> *(*p. *156)* Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.↩ Condition** (p. 429) objects has a `trigger_value` of com.rti.dds.infrastructure.true or else until the timeout expires.

Inherits AbstractNativeObject, and AutoCloseable.

Public Member Functions

- **WaitSet** ()

  Default no-argument constructor.
- **WaitSet** (**WaitSetProperty_t** prop)

  *<<**extension**>> (p. 155) Constructor for a **com.rti.dds.infrastructure.WaitSet** (p. 1973) that may delay for more while specifying that will be woken up after the given number of events or delay period, whichever happens first*
- void **wait** (**ConditionSeq** active_conditions, **Duration_t** timeout)

  Allows an application thread to wait for the occurrence of certain conditions.
- void **attach_condition** (**Condition** cond)

  *Attaches a **com.rti.dds.infrastructure.Condition** (p. 429) to the **com.rti.dds.infrastructure.WaitSet** (p. 1973).*
- void **detach_condition** (**Condition** cond)

  *Detaches a **com.rti.dds.infrastructure.Condition** (p. 429) from the **com.rti.dds.infrastructure.WaitSet** (p. 1973).*
- void **get_conditions** (**ConditionSeq** attached_conditions)

  *Retrieves the list of attached **com.rti.dds.infrastructure.Condition** (p. 429) (s).*
- void **set_property** (**WaitSetProperty_t** prop)

  *<<**extension**>> (p. 155) Sets the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1982), to configure the associated **com.rti.dds.infrastructure.WaitSet** (p. 1973) to return after one or more trigger events have occurred.*
- void **get_property** (**WaitSetProperty_t** prop)

  *<<**extension**>> (p. 155) Retrieves the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1982) configuration of the associated **com.rti.dds.infrastructure.WaitSet** (p. 1973).*
- void **delete** ()

  Destructor.
- void **close** ()

  *See **delete()** (p. 1982).*

8.408.1 Detailed Description

*<<**interface**>> (p. 156)* Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.↩ Condition** (p. 429) objects has a `trigger_value` of com.rti.dds.infrastructure.true or else until the timeout expires.

8.408.2 Usage

com.rti.dds.infrastructure.Condition (p. 429) (s) (in conjunction with wait-sets) provide an alternative mechanism to allow the middleware to communicate communication status changes (including arrival of data) to the application.

"com.rti.dds.infrastructure.WaitSet and com.rti.dds.infrastructure.Condition (s)"

This mechanism is wait-based. Its general use pattern is as follows:

- The application indicates which relevant information it wants to get by creating **com.rti.dds.infrastructure.↩ Condition** (p. 429) objects (**com.rti.dds.infrastructure.StatusCondition** (p. 1699), **com.rti.dds.subscription.↩ ReadCondition** (p. 1514) or **com.rti.dds.subscription.QueryCondition** (p. 1510)) and attaching them to a **com.rti.dds.infrastructure.WaitSet** (p. 1973).

- It then waits on that **com.rti.dds.infrastructure.WaitSet** (p. 1973) until the `trigger_value` of one or several **com.rti.dds.infrastructure.Condition** (p. 429) objects become com.rti.dds.infrastructure.true.

- It then uses the result of the wait (i.e., active_conditions, the list of **com.rti.dds.infrastructure.Condition** (p. 429) objects with `trigger_value` == com.rti.dds.infrastructure.true) to actually get the information:
  - by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034) and then `get_<communication_status>`() on the relevant **com.rti.dds.infrastructure.Entity** (p. 1029), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) and the status changes, refer to plain communication status;
  - by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034) and then **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1741) on the relevant **com.rti.dds.subscription.Subscriber** (p. 1730) (and then **com.rti.ndds.example.FooDataReader.read()** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) on the returned **com.rti.dds.subscription.DataReader** (p. 450) objects), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) and the status changes refers to com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_ON_READERS_STATUS;
  - by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 1034) and then **com.rti.ndds.example.FooDataReader.read()** (p. 1069) or **com.rti.ndds.example.FooDataReader.take** (p. 1071) on the relevant **com.rti.dds.subscription.DataReader** (p. 450), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) and the status changes refers to com.rti.dds.infrastructure.StatusKind.StatusKind.DATA_AVAILABLE_STATUS;
  - by calling directly **com.rti.ndds.example.FooDataReader.read_w_condition** (p. 1076) or **com.rti.ndds.example.FooDataReader.take_w_condition** (p. 1077) on a **com.rti.dds.subscription.DataReader** (p. 450) with the **com.rti.dds.infrastructure.Condition** (p. 429) as a parameter if it is a **com.rti.dds.subscription.ReadCondition** (p. 1514) or a **com.rti.dds.subscription.QueryCondition** (p. 1510).

Usually the first step is done in an initialization phase, while the others are put in the application main loop.

As there is no extra information passed from the middleware to the application when a wait returns (only the list of triggered **com.rti.dds.infrastructure.Condition** (p. 429) objects), **com.rti.dds.infrastructure.Condition** (p. 429) objects are meant to embed all that is needed to react properly when enabled. In particular, **com.rti.dds.infrastructure.Entity** (p. 1029)-related conditions are related to exactly one **com.rti.dds.infrastructure.Entity** (p. 1029) and cannot be shared.

The blocking behavior of the **com.rti.dds.infrastructure.WaitSet** (p. 1973) is illustrated below.

blocking behavior"

The result of a com.rti.dds.infrastructure.WaitSet.WaitSet.wait operation depends on the state of the **com.rti.dds.infrastructure.WaitSet** (p. 1973), which in turn depends on whether at least one attached **com.rti.dds.infrastructure.Condition** (p. 429) has a `trigger_value` of com.rti.dds.infrastructure.true. If the wait operation is called on **com.rti.dds.infrastructure.WaitSet** (p. 1973) with state BLOCKED, it will block the calling thread. If wait is called on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) with state UNBLOCKED, it will return immediately. In addition, when the **com.rti.dds.infrastructure.WaitSet** (p. 1973) transitions from BLOCKED to UNBLOCKED it wakes up any threads that had called wait on it.

A key aspect of the **Condition** (p. 429) and **WaitSet** (p. 1973) mechanism is the setting of the `trigger_value` of each **com.rti.dds.infrastructure.Condition** (p. 429).

The **com.rti.dds.infrastructure.WaitSet** (p. 1973) cannot be used after calling **com.rti.dds.domain.DomainParticipantFactory.finalize_instance** (p. 764).

8.408.3 Trigger State of a com.rti.dds.infrastructure.StatusCondition

The `trigger_value` of a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) is the boolean OR of the ChangedStatusFlag of all the communication statuses (see **Status Kinds** (p. 262)) to which it is sensitive. That is, `trigger_value` == com.rti.dds.infrastructure.false only if all the values of the ChangedStatusFlags are com.↩ rti.dds.infrastructure.false.

The sensitivity of the **com.rti.dds.infrastructure.StatusCondition** (p. 1699) to a particular communication status is controlled by the list of `enabled_statuses` set on the condition by means of the **com.rti.dds.infrastructure.↩ StatusCondition.set_enabled_statuses** (p. 1700) operation.

Once the `trigger_value` of a **StatusCondition** (p. 1699) becomes true, it remains true until the status that changed is reset. To reset a status, call the related get_*_status() operation. Or, in the case of the data available status, call read(), take(), or one of their variants. Therefore, if you are using a **com.rti.dds.infrastructure.StatusCondition** (p. 1699) on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) to be notified of events, your thread will wake up when one of the statuses associated with the **StatusCondition** (p. 1699) becomes true. If you do not reset the status, the **Status↩ Condition** (p. 1699) `trigger_value` remains true and your **WaitSet** (p. 1973) will not block again; it will immediately wake up when you call com.rti.dds.infrastructure.WaitSet.WaitSet.wait.

8.408.4 Trigger State of a com.rti.dds.subscription.ReadCondition

Similar to the **com.rti.dds.infrastructure.StatusCondition** (p. 1699), a **com.rti.dds.subscription.ReadCondition** (p. 1514) also has a `trigger_value` that determines whether the attached **com.rti.dds.infrastructure.WaitSet** (p. 1973) is BLOCKED or UNBLOCKED. However, unlike the **com.rti.dds.infrastructure.StatusCondition** (p. 1699), the `trigger_value` of the **com.rti.dds.subscription.ReadCondition** (p. 1514) is tied to the presence of *at least a sample* managed by RTI Connext with **com.rti.dds.subscription.SampleStateKind** (p. 1663) and **com.rti.dds.↩ subscription.ViewStateKind** (p. 1970) matching those of the **com.rti.dds.subscription.ReadCondition** (p. 1514). Furthermore, for the **com.rti.dds.subscription.QueryCondition** (p. 1510) to have a `trigger_value` == com.↩ rti.dds.infrastructure.true, the data associated with the sample must be such that the `query_expression` evaluates to com.rti.dds.infrastructure.true.

The fact that the trigger_value of a **com.rti.dds.subscription.ReadCondition** (p. 1514) depends on the presence of samples on the associated **com.rti.dds.subscription.DataReader** (p. 450) implies that a single take operation can potentially change the `trigger_value` of several **com.rti.dds.subscription.ReadCondition** (p. 1514) or **com.↩ rti.dds.subscription.QueryCondition** (p. 1510) conditions. For example, if all samples are taken, any **com.rti.dds.↩ subscription.ReadCondition** (p. 1514) and **com.rti.dds.subscription.QueryCondition** (p. 1510) conditions associated with the **com.rti.dds.subscription.DataReader** (p. 450) that had their `trigger_value==TRUE` before will see the `trigger_value` change to FALSE. Note that this does not guarantee that **com.rti.dds.infrastructure.WaitSet** (p. 1973) objects that were separately attached to those conditions will not be woken up. Once we have `trigger_↩ value==TRUE` on a condition, it may wake up the attached **com.rti.dds.infrastructure.WaitSet** (p. 1973), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwakeup' the **WaitSet** (p. 1973) as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) may return from the wait with a list of conditions, some of which are not no longer 'active'. This is unavoidable if multiple threads are concurrently waiting on separate **com.rti.dds.infrastructure.WaitSet** (p. 1973) objects and taking data associated with the same **com.rti.dds.subscription.DataReader** (p. 450) entity.

To elaborate further, consider the following example: A **com.rti.dds.subscription.ReadCondition** (p. 1514) that has a `sample_state_mask` = {com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_READ_SAMPLE_↩ STATE} will have `trigger_value` of com.rti.dds.infrastructure.true whenever a new sample arrives and will transition to com.rti.dds.infrastructure.false as soon as all the newly-arrived samples are either read (so their sample state

changes to READ) or taken (so they are no longer managed by RTI Connext). However if the same **com.rti.dds.↩ subscription.ReadCondition** (p. 1514) had a `sample_state_mask` = { com.rti.dds.subscription.SampleState↩ Kind.SampleStateKind.READ_SAMPLE_STATE, com.rti.dds.subscription.SampleStateKind.SampleStateKind.NOT_↩ READ_SAMPLE_STATE }, then the `trigger_value` would only become com.rti.dds.infrastructure.false once all the newly-arrived samples are taken (it is not sufficient to read them as that would only change the sample state to READ), which overlaps the mask on the **com.rti.dds.subscription.ReadCondition** (p. 1514).

8.408.5 Trigger State of a com.rti.dds.infrastructure.GuardCondition

The `trigger_value` of a **com.rti.dds.infrastructure.GuardCondition** (p. 1129) is completely controlled by the application via the operation **com.rti.dds.infrastructure.GuardCondition.set_trigger_value** (p. 1130).

Important: The **com.rti.dds.infrastructure.WaitSet** (p. 1973) allocates native resources. When **com.rti.dds.↩ infrastructure.WaitSet** (p. 1973) is no longer being used, user should call com.rti.dds.infrastructure.WaitSet.Wait↩ Set.delete explicitly to properly cleanup all native resources.

See also

Status Kinds (p. 262)

com.rti.dds.infrastructure.StatusCondition (p. 1699), **com.rti.dds.infrastructure.GuardCondition** (p. 1129)

com.rti.dds.infrastructure.Listener (p. 1236)

8.408.6 Constructor & Destructor Documentation

8.408.6.1 WaitSet() [1/2]

```
WaitSet ( )
```

Default no-argument constructor.

Construct a new **com.rti.dds.infrastructure.WaitSet** (p. 1973).

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.↩ rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipant↩ Factory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.↩ rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.↩ WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.dds.↩ infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.↩ ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Important: **WaitSet** (p. 1973) allocates native resources. When a **WaitSet** (p. 1973) is no longer being used, **close()** (p. 1982) must be called.

Exceptions

com.rti.dds.infrastructure.RETCODE_OUT_OF_↩ RESOURCES *(p. 1598)*	if a new **com.rti.dds.infrastructure.WaitSet** (p. 1973) could not be allocated.

8.408.6.2 WaitSet() [2/2]

```
WaitSet (
            WaitSetProperty_t prop )
```

<<***extension***>> *(p. 155)* Constructor for a **com.rti.dds.infrastructure.WaitSet** (p. 1973) that may delay for more while specifying that will be woken up after the given number of events or delay period, whichever happens first

Constructs a new **com.rti.dds.infrastructure.WaitSet** (p. 1973).

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this method while another thread may be simultaneously calling **com.↩ rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.dds.domain.DomainParticipant↩ Factory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.get_instance** (p. 1923), com.↩ rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.dds.infrastructure.WaitSet.↩ WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty_t), **com.rti.dds.↩ infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.↩ ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Important: **WaitSet** (p. 1973) allocates native resources. When a **WaitSet** (p. 1973) is no longer being used, **close()** (p. 1982) must be called.

Parameters

prop	<<***in***>> *(p. 156)* Property of wait set controlling when the wait set should be woken up

Exceptions

com.rti.dds.infrastructure.RETCODE_OUT_OF_↩ RESOURCES *(p. 1598)*	if a new **com.rti.dds.infrastructure.WaitSet** (p. 1973) could not be allocated.

8.408.7 Member Function Documentation

8.408.7.1 wait()

```
void wait (
              ConditionSeq active_conditions,
              Duration_t timeout )
```

Allows an application thread to wait for the occurrence of certain conditions.

If none of the conditions attached to the **com.rti.dds.infrastructure.WaitSet** (p. 1973) have a `trigger_value` of com.rti.dds.infrastructure.true, the wait operation will block, suspending the calling thread.

The result of the wait operation is the list of all the attached conditions that have a `trigger_value` of com.rti.dds.↩ infrastructure.true (i.e., the conditions that unblocked the wait).

The wait operation takes a `timeout` argument that specifies the maximum duration for the wait. If this duration is exceeded and none of the attached **com.rti.dds.infrastructure.Condition** (p. 429) objects are com.rti.dds.↩ infrastructure.true, wait fails with **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599). In this case, the resulting list of conditions will be empty. If a negative duration is passed, wait fails with **com.rti.dds.infrastructure.RETCODE**↩ **_BAD_PARAMETER** (p. 1594).

Note: The resolution of the `timeout` period is constrained by the resolution of the system clock.

When the **com.rti.dds.infrastructure.WaitSet** (p. 1973) is configured to wait for more than one trigger event and the timeout is exceeded before that number is reached, this function returns normally as long as at least one trigger event has occurred.

It is not allowable for more than one application thread to be waiting on the same **com.rti.dds.infrastructure.Wait**↩ **Set** (p. 1973). If the wait operation is invoked on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) that already has a thread blocking on it, the operation will return immediately with the value **com.rti.dds.infrastructure.RETCODE_**↩ **PRECONDITION_NOT_MET** (p. 1598).

Parameters

active_conditions	<<***inout***>> *(p. 156)* a valid non-null **com.rti.dds.infrastructure.ConditionSeq** (p. 430) object. Note that RTI Connext will not allocate a new object if `active_conditions` is null; the method will return **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).
timeout	<<***in***>> *(p. 156)* a wait timeout

Exceptions

One	of the **Standard Return Codes** (p. 261) or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598) or **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599).

References **Duration_t.nanosec**, and **Duration_t.sec**.

8.408.7.2 attach_condition()

```
void attach_condition (
              Condition cond )
```

Attaches a **com.rti.dds.infrastructure.Condition** (p. 429) to the **com.rti.dds.infrastructure.WaitSet** (p. 1973).

It is possible to attach a **com.rti.dds.infrastructure.Condition** (p. 429) on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) that is currently being waited upon (via the wait operation). In this case, if the **com.rti.dds.infrastructure.↩ Condition** (p. 429) has a `trigger_value` of com.rti.dds.infrastructure.true, then attaching the condition will unblock the **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Parameters

cond	<<***in***>> *(p. 156)* **Condition** (p. 429) to be attached.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_OUT_OF_RESOURCES** (p. 1598).

8.408.7.3 detach_condition()

```
void detach_condition (
              Condition cond )
```

Detaches a **com.rti.dds.infrastructure.Condition** (p. 429) from the **com.rti.dds.infrastructure.WaitSet** (p. 1973).

It is possible to detach a **com.rti.dds.infrastructure.Condition** (p. 429) on a **com.rti.dds.infrastructure.WaitSet** (p. 1973) that is currently being waited upon (via the wait operation). If the **com.rti.dds.infrastructure.Condition** (p. 429) was not attached to the **com.rti.dds.infrastructure.WaitSet** (p. 1973), the operation will return **com.rti.dds.↩ infrastructure.RETCODE_BAD_PARAMETER** (p. 1594).

Parameters

cond	<<***in***>> *(p. 156)* **Condition** (p. 429) to be detached.

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.408.7.4 get_conditions()

```
void get_conditions (
            ConditionSeq attached_conditions )
```

Retrieves the list of attached **com.rti.dds.infrastructure.Condition** (p. 429) (s).

Parameters

attached_conditions	*<<**inout**>> (p. 156)* a **com.rti.dds.infrastructure.ConditionSeq** (p. 430) object where the list of attached conditions will be returned

Exceptions

One	of the **Standard Return Codes** (p. 261), or **com.rti.dds.infrastructure.RETCODE_PRECONDITION_NOT_MET** (p. 1598).

8.408.7.5 set_property()

```
void set_property (
            WaitSetProperty_t prop )
```

*<<**extension**>> (p. 155)* Sets the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1982), to configure the associated **com.rti.dds.infrastructure.WaitSet** (p. 1973) to return after one or more trigger events have occurred.

Parameters

prop	*<<**in**>> (p. 156)*

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.408.7.6 get_property()

```
void get_property (
            WaitSetProperty_t prop )
```

*<<**extension**>> (p. 155)* Retrieves the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1982) configuration of the associated **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Parameters

prop	<<***out***>> *(*p. *156)*

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.408.7.7 delete()

```
void delete ( )
```

Destructor.

Releases the resources asociated with this **com.rti.dds.infrastructure.WaitSet** (p. 1973).

Calling this method multiple times on the same **com.rti.dds.infrastructure.WaitSet** (p. 1973) is safe; subsequent deletions will have no effect. **com.rti.dds.domain.DomainParticipantFactory.get_instance** (p. 764), **com.rti.↩ dds.domain.DomainParticipantFactory.finalize_instance** (p. 764), **com.rti.dds.typecode.TypeCodeFactory.↩ get_instance** (p. 1923), com.rti.dds.infrastructure.GuardCondition.GuardCondition.GuardCondition(), com.rti.↩ dds.infrastructure.WaitSet.WaitSet.WaitSet(), com.rti.dds.infrastructure.WaitSet.WaitSet.WaitSet(WaitSetProperty↩ _t), **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1131), com.rti.dds.infrastructure.WaitSet.WaitSet.delete, **com.rti.ndds.utility.NetworkCapture.enable** (p. 1324), or **com.rti.ndds.utility.NetworkCapture.disable** (p. 1324).

Referenced by **WaitSet.close()**.

8.408.7.8 close()

```
void close ( )
```

See **delete()** (p. 1982).

References **WaitSet.delete()**.

8.409 WaitSetProperty_t Class Reference

<<***extension***>> *(*p. *155)* Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1973) behavior for multiple trigger events.

Inherits Struct.

Public Attributes

- int **max_event_count**

  *Maximum number of trigger events to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1973) to awaken.*
- final **Duration_t max_event_delay**

  *Maximum delay from occurrence of first trigger event to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1973) to awaken.*

8.409.1 Detailed Description

<<***extension***>> *(p. 155)* Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1973) behavior for multiple trigger events.

In simple use, a **com.rti.dds.infrastructure.WaitSet** (p. 1973) returns when a single trigger event occurs on one of its attached **com.rti.dds.infrastructure.Condition** (p. 429) (s), or when the `timeout` maximum wait duration specified in the com.rti.dds.infrastructure.WaitSet.WaitSet.wait call expires.

The **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1982) allows configuration of the waiting behavior of a **com.rti.dds.infrastructure.WaitSet** (p. 1973). If no conditions are true at the time of the call to wait, then the `max_event_count` parameter may be used to configure the **WaitSet** (p. 1973) to wait for `max_event_count` trigger events to occur before returning, or to wait for up to `max_event_delay` time from the occurrence of the first trigger event before returning.

The `timeout` maximum wait duration specified in the com.rti.dds.infrastructure.WaitSet.WaitSet.wait call continues to apply.

Entity:

com.rti.dds.infrastructure.WaitSet (p. 1973)

Properties:

RxO (p. 256) = N/A

Changeable (p. 256) = **YES** (p. 256)

8.409.2 Member Data Documentation

8.409.2.1 max_event_count

```
int max_event_count
```

Maximum number of trigger events to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1973) to awaken.

The **com.rti.dds.infrastructure.WaitSet** (p. 1973) will wait until up to `max_event_count` trigger events have occurred before returning. The **com.rti.dds.infrastructure.WaitSet** (p. 1973) may return earlier if either the `timeout` duration has expired, or `max_event_delay` has elapsed since the occurrence of the first trigger event. `max_event_count` may be used to "collect" multiple trigger events for processing at the same time.

[default] 1

[range] >= 1

8.409.2.2 max_event_delay

```
final Duration_t max_event_delay
```

Maximum delay from occurrence of first trigger event to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1973) to awaken.

The **com.rti.dds.infrastructure.WaitSet** (p. 1973) will return no later than `max_event_delay` after the first trigger event. `max_event_delay` may be used to establish a maximum latency for events reported by the **com.rti.dds.infrastructure.WaitSet** (p. 1973).

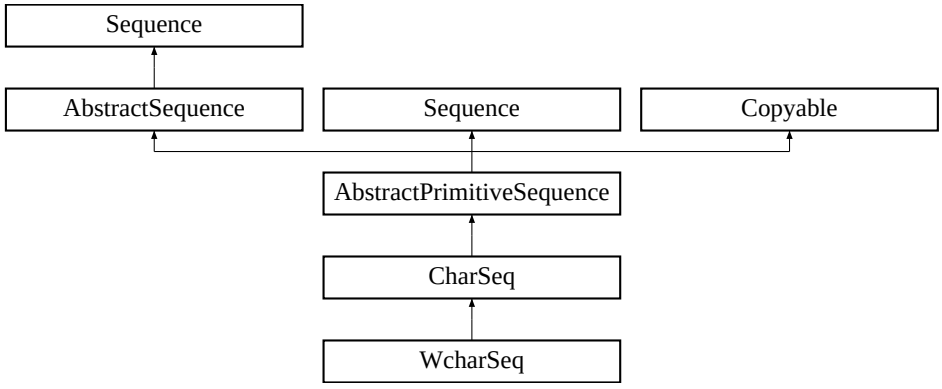
Note that **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) is *not* returned if `max_event_delay` is exceeded. **com.rti.dds.infrastructure.RETCODE_TIMEOUT** (p. 1599) is returned only if the `timeout` duration expires before any trigger events occur.

[default] com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE (p. 846)

8.410 WcharSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.char >

Inheritance diagram for WcharSeq:



Public Member Functions

- **WcharSeq** ()

  Constructs an empty sequence of wide characters with an initial maximum of zero.
- **WcharSeq** (int initialMaximum)

  Constructs an empty sequence of wide characters with the given initial maximum.
- **WcharSeq** (char[ ] chars)

  Constructs a new sequence containing the given wide characters.

8.410.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ char >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.char

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.410.2 Constructor & Destructor Documentation

8.410.2.1 WcharSeq() [1/3]

```
WcharSeq ( )
```

Constructs an empty sequence of wide characters with an initial maximum of zero.

8.410.2.2 WcharSeq() [2/3]

```
WcharSeq (
            int initialMaximum )
```

Constructs an empty sequence of wide characters with the given initial maximum.

8.410.2.3 WcharSeq() [3/3]

```
WcharSeq (
            char[] chars )
```

Constructs a new sequence containing the given wide characters.

Parameters

chars	the initial contents of this sequence

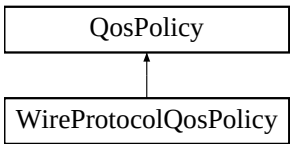
Exceptions

NullPointerException	if the input array is null

8.411 WireProtocolQosPolicy Class Reference

Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 670).

Inheritance diagram for WireProtocolQosPolicy:



Public Attributes

- int **participant_id**

  A value used to distinguish among different participants belonging to the same domain on the same host.
- **RtpsWellKnownPorts_t rtps_well_known_ports**

  Configures the RTPS well-known port mappings.
- int **rtps_host_id**

  The RTPS Host ID of the domain participant.
- int **rtps_app_id**

  The RTPS App ID of the domain participant.
- int **rtps_instance_id**

  *The RTPS Instance ID of the **com.rti.dds.domain.DomainParticipant** (p. 670).*
- int **rtps_reserved_port_mask**

  Specifies which well-known ports to reserve when enabling the participant.
- **WireProtocolQosPolicyAutoKind rtps_auto_id_kind** = **WireProtocolQosPolicyAutoKind.RTPS_AUTO_↩ ID_FROM_UUID**

  Kind of auto mechanism used to calculate the GUID prefix.
- boolean **compute_crc**

  Adds RTPS CRC submessage to every message when this field is set to com.rti.dds.infrastructure.true.
- boolean **check_crc**

  Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to com.rti.dds.infrastructure.true.

Static Public Attributes

- static final int **RTPS_AUTO_ID** = 0

  Indicates that RTI Connext should choose an appropriate host, app, instance or object ID automatically.

8.411.1 Detailed Description

Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 670).

Entity:

com.rti.dds.domain.DomainParticipant (p. 670)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **NO** (p. 256)

8.411.2 Usage

This QoS policy configures some participant-wide properties of the DDS Real-Time Publish Subscribe (RTPS) on-the-wire protocol. (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 596) and **com.rti.dds.infrastructure.↩ DataReaderProtocolQosPolicy** (p. 501) configure RTPS and reliability properties on a per **com.rti.dds.publication.↩ DataWriter** (p. 553) or **com.rti.dds.subscription.DataReader** (p. 450) basis.)

NOTE: The default QoS policies returned by RTI Connext contain the correctly initialized wire protocol attributes. The defaults are not normally expected to be modified, but are available to the advanced user customizing the implementation behavior.

The default values should not be modified without an understanding of the underlying Real-Time Publish Subscribe (RTPS) wire protocol.

In order for the discovery process to work correctly, each **com.rti.dds.domain.DomainParticipant** (p. 670) must have a unique identifier. This QoS policy specifies how that identifier should be generated.

RTPS defines a 96-bit prefix to this identifier; each **com.rti.dds.domain.DomainParticipant** (p. 670) must have a unique value for this prefix relative to all other participants in its domain.

If an application dies unexpectedly and is restarted, the IDs used by the new instance of DomainParticipants should be different than the ones used by the previous instances. A change in these values allows other DomainParticipants to know that they are communicating with a new instance of an application, and not the previous instance.

For legacy reasons, RTI Connext divides the 96-bit prefix into three integers:

- The first integer is called host ID. The original purpose of this integer was to contain the identity of the machine on which the DomainParticipant is executing.

- The second integer is called an application ID. The original purpose of this integer was to contain a value that identifies the process or task in which the DomainParticipant is contained.
- The third integer is called instance ID. The original purpose was to contain a value that uniquely identifies a DomainParticipant within a task or process.

The **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) field can be used to configure the algorithm that RTI Connext uses to populate the 96-bit prefix. Then you can optionally overwrite specific parts of the 96-bit prefix by explicitly configuring the **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_host_id** (p. 1991) (first integer), **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1991) (second integer), and **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1992) (third integer).

The **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) field supports three different prefix generation algorithms:

1. In the default and most common scenario, **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_↩ id_kind** (p. 1993) is set to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_↩ FROM_UUID** (p. 284). As the name suggests, this mechanism uses a unique, randomly generated UUID to fill the **rtps_host_id**, **rtps_app_id**, or **rtps_instance_id** fields. The first two bytes of the **rtps_host_id** are replaced with the RTI vendor ID (0x0101).

2. (Legacy) When rtps_auto_id_kind is set to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.↩ RTPS_AUTO_ID_FROM_IP** (p. 284), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The 32-bit value of the IPv4 of the first up and running interface of the host machine is assigned. If the host does not have an IPv4 address, the host-id will be automatically set to 0x7F000001.
   - **rtps_app_id:** The process (or task) ID is assigned.
   - **rtps_instance_id:** A counter is assigned that is incremented per new participant within a process.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

3. (Legacy) When rtps_auto_id_kind is set to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.↩ RTPS_AUTO_ID_FROM_MAC** (p. 284), the 96-bit prefix is generated as follows:

   - **rtps_host_id:** The first 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_app_id:** The last 32 bits of the MAC address of the first up and running interface of the host machine are assigned.
   - **rtps_instance_id:** This field is split into two different parts. The process (or task) ID is assigned to the first 24 bits. A counter is assigned to the last 8 bits. This counter is incremented per new participant.

   DDS_RTPS_AUTO_ID_FROM_IP is not a good algorithm to guarantee prefix uniqueness, because the process ID can be recycled by the OSs.

Some examples are provided to clarify the behavior of this QoSPolicy in case you want to change the default behavior with **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 284).

First, get the participant QoS from the DomainParticipantFactory:

```
DomainParticipantFactory.TheParticipantFactory.

                get_default_participant_qos(participant_qos);
```

<P>

Second, change the **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1986) using one of the options shown below.

Third, create the **com.rti.dds.domain.DomainParticipant** (p. 670) as usual, using the modified QoS structure instead of the default one.

Option 1: Use **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 284) to explicitly set just the application/task identifier portion of the **rtps_instance_id** field.

```
participant_qos.wire_protocol.rtps_auto_id_kind =

                WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      =

                WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       =

                WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = (/* App ID */ (12 << 8) |

                                                   /* Instance ID*/ (WireProtocolQosPolicy.RTPS_AUTO_ID));
```

Option 2: Handle only the per participant counter and let RTI Connext handle the application/task identifier:

```
participant_qos.wire_protocol.rtps_auto_id_kind =

                        WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  =  (/* App ID */ (WireProtocolQosPolicy.RTPS_AUTO_ID) |

                                                   /* Instance ID*/ (12));
```

Option 3: Handle the entire **rtps_instance_id** field yourself:

```
participant_qos.wire_protocol.rtps_auto_id_kind = WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = ( /* App ID */ (12 << 8)) |

                                                    /* Instance ID */ (9) )
```

NOTE: If you are using **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 284) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to avoid errors.

Option 4: Let RTI Connext handle the entire **rtps_instance_id** field:

```
participant_qos.wire_protocol.rtps_auto_id_kind =

                WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;

participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;

participant_qos.wire_protocol.rtps_instance_id  = WireProtocolQosPolicy.RTPS_AUTO_ID;
```

NOTE: If you are using **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 284) as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both parts are non-zero (otherwise RTI Connext will take responsibility for them). RTI recommends that you always specify the two parts separately in order to clearly show the difference.

8.411.3 Member Data Documentation

8.411.3.1 RTPS_AUTO_ID

```
final int RTPS_AUTO_ID = 0  [static]
```

Indicates that RTI Connext should choose an appropriate host, app, instance or object ID automatically.

If this special value is assigned to **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_host_id** (p. 1991), **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1991), **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1992), **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_object_id** (p. 598) or **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_object_id** (p. 502) RTI Connext will assign the ID automatically.

The actual ID value is chosen when the QoS is set: the QoS returned from **com.rti.dds.domain.DomainParticipant.get_qos** (p. 715), **com.rti.dds.publication.DataWriter.get_qos** (p. 558) or **com.rti.dds.subscription.DataReader.get_qos** (p. 459) will never have this value.

QoS:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_host_id (p. 1991) **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1991) **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1992)

8.411.3.2 participant_id

```
int participant_id
```

A value used to distinguish among different participants belonging to the same domain on the same host.

Determines the unicast port on which meta-traffic is received. Also defines the *default* unicast port for receiving user-traffic for DataReaders and DataWriters (can be overridden by the **com.rti.dds.subscription.DataReaderQos.unicast** (p. 524) or **com.rti.dds.publication.DataWriterQos.unicast** (p. 620)).

For more information on port mapping, please refer to **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622).

Each **com.rti.dds.domain.DomainParticipant** (p. 670) in the same domain and running on the same host, must have a unique `participant_id`. The participants may be in the same address space or in distinct address spaces.

A negative number (-1) means that RTI Connext will *automatically* resolve the participant ID as follows.

- RTI Connext will pick the *smallest* participant ID based on the unicast ports available on the transports enabled for discovery.
- RTI Connext will attempt to resolve an automatic port index either when a DomainPartcipant is enabled, or when a DataReader or DataWriter is created. Therefore, all the transports enabled for discovery must have been registered by this time. Otherwise, the discovery transports registered after resolving the automatic port index may produce port conflicts when the DomainPartcipant is enabled.

[default] -1 [automatic], i.e. RTI Connext will automatically pick the `participant_id`, as described above.

[range] [>= 0], or -1, and does not violate guidelines stated in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622).

See also

com.rti.dds.infrastructure.Entity.enable() (p. 1032)

8.411.3.3 rtps_well_known_ports

```
RtpsWellKnownPorts_t rtps_well_known_ports
```

Configures the RTPS well-known port mappings.

Determines the well-known multicast and unicast port mappings for discovery (meta) traffic and user traffic.

[default] **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 283)

8.411.3.4 rtps_host_id

```
int rtps_host_id
```

The RTPS Host ID of the domain participant.

A specific host ID that is unique in the domain.

[default] **com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID** (p. 1990). The default value is interpreted as follows:
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_IP** (p. 284), the value will be interpreted as the IPv4 address of the *first* up and running interface of the host machine.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 284), the value will be interpreted as the first 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_UUID** (p. 284), the value will be the first 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

8.411.3.5 rtps_app_id

```
int rtps_app_id
```

The RTPS App ID of the domain participant.

A participant specific ID that, together with the rtps_instance_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an app ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID (p. 1990). The default value is interpreted as follows:
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_IP*** *(p. 284)* the value will be the process (or task) ID.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC*** *(p. 284)* the value will be the last 32 bits of the MAC address assigned to the *first* up and running interface of the host machine.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_UUID*** *(p. 284)*, the value will be the middle 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

8.411.3.6 rtps_instance_id

```
int rtps_instance_id
```

The RTPS Instance ID of the **com.rti.dds.domain.DomainParticipant** (p. 670).

This is an instance-specific ID of a participant that, together with the rtps_app_id, is unique within the scope of the rtps_host_id.

If a participant dies and is restarted, it is recommended that it be given an instance ID that is distinct from the previous one so that other participants in the domain can distinguish between them.

[default] com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID (p. 1990). The default value is interpreted as follows:
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_IP*** *(p. 284)*, a counter is assigned that is incremented per new participant. For VxWorks-653, the first 8 bits are assigned to the partition id for the application.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC*** *(p. 284)*, the first 24 bits are assigned to the application/task identifier and the last 8 bits are assigned to a counter that is incremented per new paricipant.
If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1993) is equal to ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_UUID*** *(p. 284)*, the value will be the last 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff] **NOTE:** If you use ***com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC*** *(p. 284)* as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you must ensure that both the two parts are non-zero, otherwise the middleware will take responsibility for them. We recommend that you always specify the two parts separately in order to avoid errors. (`examples`)

8.411.3.7 rtps_reserved_port_mask

```
int rtps_reserved_port_mask
```

Specifies which well-known ports to reserve when enabling the participant.

Specifies which of the well-known multicast and unicast ports will be reserved when the DomainParticipant is enabled. Failure to allocate a port that is computed based on the **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1622) will be detected at this time, and the enable operation will fail.

[default] com.rti.dds.infrastructure.RtpsReservedPortKind.MASK_DEFAULT (p. 281)

8.411.3.8 rtps_auto_id_kind

```
WireProtocolQosPolicyAutoKind rtps_auto_id_kind = WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_↩
FROM_UUID
```

Kind of auto mechanism used to calculate the GUID prefix.

[default] com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_UUID (p. 284)

8.411.3.9 compute_crc

```
boolean compute_crc
```

Adds RTPS CRC submessage to every message when this field is set to com.rti.dds.infrastructure.true.

The computed CRC covers the entire RTPS message excluding the RTPS header.

[default] com.rti.dds.infrastructure.false

8.411.3.10 check_crc

```
boolean check_crc
```

Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to com.rti.dds.infrastructure.true.

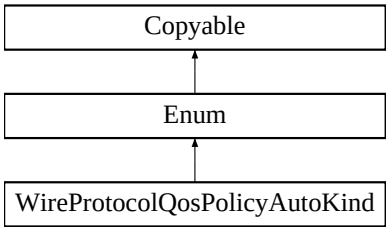
com.rti.dds.infrastructure.WireProtocolQosPolicy.compute_crc (p. 1993) must be enabled at the publishing application for validating the message at the subscribing application.

[default] com.rti.dds.infrastructure.false

8.412 WireProtocolQosPolicyAutoKind Class Reference

Mechanism to automatically calculate the GUID prefix.

Inheritance diagram for WireProtocolQosPolicyAutoKind:



Static Public Attributes

- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_IP** = new **WireProtocolQosPolicy↩ AutoKind**("RTPS_AUTO_ID_FROM_IP", 0)

  Mechanism to automatically calculate the GUID prefix.
- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_MAC** = new **WireProtocolQos↩ PolicyAutoKind**("RTPS_AUTO_ID_FROM_MAC", 1)

  Mechanism to automatically calculate the GUID prefix.
- static final **WireProtocolQosPolicyAutoKind** **RTPS_AUTO_ID_FROM_UUID** = new **WireProtocolQos↩ PolicyAutoKind**("RTPS_AUTO_ID_FROM_UUID", 2)

  Mechanism to automatically calculate the GUID prefix.

Additional Inherited Members

8.412.1 Detailed Description

Mechanism to automatically calculate the GUID prefix.

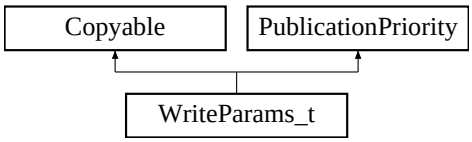
See also

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind (p. 1993)

8.413 WriteParams_t Class Reference

<<***extension***>> *(p. 155)* Input parameters for writing with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), **com.rti.ndds.example.FooDataWriter.dispose_w_params** (p. 1114), **com.rti.ndds.example.FooData↩ Writer.register_instance_w_params** (p. 1100), **com.rti.ndds.example.FooDataWriter.unregister_instance_w_↩ params** (p. 1104)

Inheritance diagram for WriteParams_t:



Public Member Functions

- **WriteParams_t** ()

  *Construct a new **WriteParams_t** (p. 1994).*
- **WriteParams_t** (boolean replace_auto, **SampleIdentity_t identity**, **SampleIdentity_t related_sample_identity**, **Time_t source_timestamp**, **Cookie_t** cookie, **InstanceHandle_t handle**, int **priority**, int **flag**, **GUID_t source_guid**, **GUID_t related_source_guid**, **GUID_t related_reader_guid**)

  *Construct a new **WriteParams_t** (p. 1994) with the given members.*
- Object **copy_from** (Object src)

  Copy value of a data type from source.

Public Attributes

- **SampleIdentity_t identity**

  Identity of the sample.
- final **SampleIdentity_t related_sample_identity**

  The identity of another sample related to this one.
- **Time_t source_timestamp** = new **Time_t**(**Time_t.TIME_INVALID**)

  Source timestamp upon write.
- **InstanceHandle_t handle** = new **InstanceHandle_t**()

  Instance handle.
- int **priority** = **UNDEFINED**

  Publication priority.
- int **flag** = 0

  Flags associated with the sample.
- **GUID_t source_guid** = new **GUID_t**(**GUID_t.GUID_AUTO**)

  Identifies the application logical data source associated with the sample being written.
- **GUID_t related_source_guid** = new **GUID_t**()

  Identifies the application logical data source that is related to the sample being written.
- **GUID_t related_reader_guid** = new **GUID_t**()

  Identifies a DataReader that is logically related to the sample that is being written.

Additional Inherited Members

8.413.1 Detailed Description

<<***extension***>> *(p. 155)* Input parameters for writing with **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110), **com.rti.ndds.example.FooDataWriter.dispose_w_params** (p. 1114), **com.rti.ndds.example.FooDataWriter.register_instance_w_params** (p. 1100), **com.rti.ndds.example.FooDataWriter.unregister_instance_w_params** (p. 1104)

8.413.2 Constructor & Destructor Documentation

8.413.2.1 WriteParams_t() [1/2]

```
WriteParams_t ( )
```

Construct a new **WriteParams_t** (p. 1994).

References **InstanceHandle_t.equals()**, **WriteParams_t.handle**, and **InstanceHandle_t.HANDLE_NIL**.

Referenced by **WriteParams_t.copy_from()**.

8.413.2.2 WriteParams_t() [2/2]

```
WriteParams_t (
            boolean replace_auto,
             SampleIdentity_t identity,
             SampleIdentity_t related_sample_identity,
             Time_t source_timestamp,
             Cookie_t cookie,
             InstanceHandle_t handle,
            int priority,
            int flag,
             GUID_t source_guid,
             GUID_t related_source_guid,
             GUID_t related_reader_guid )
```

Construct a new **WriteParams_t** (p. 1994) with the given members.

References **InstanceHandle_t.copy_from()**, **AbstractPrimitiveSequence.copy_from()**, **WriteParams_t.flag**, **WriteParams_t.handle**, **WriteParams_t.identity**, **Time_t.nanosec**, **WriteParams_t.priority**, **WriteParams_t.↩ related_reader_guid**, **WriteParams_t.related_sample_identity**, **WriteParams_t.related_source_guid**, **Time_t.↩ sec**, **WriteParams_t.source_guid**, **WriteParams_t.source_timestamp**, and **Cookie_t.value**.

8.413.3 Member Function Documentation

8.413.3.1 copy_from()

```
Object copy_from (
            Object src )
```

Copy value of a data type from source.

Copy data into *this* object from another. This copy is intended to be a deep copy, so that all data members (recursively) are copied (not just resetting Object references).

This operation returns the object that is copied if copy is successful.

Parameters

src	<<***in***>> *(*p. *156)* The Object which contains the data to be copied.

Returns

Generally, return *this* but special cases (such as Enum) exist.

Exceptions

NullPointerException	If *src* is null.
ClassCastException	If *src* is not the same type as *this*.

Implements **Copyable** (p. 445).

References **InstanceHandle_t.copy_from()**, **AbstractPrimitiveSequence.copy_from()**, **WriteParams_t.flag**, **WriteParams_t.handle**, **WriteParams_t.identity**, **Time_t.nanosec**, **WriteParams_t.priority**, **WriteParams_t.↩ related_reader_guid**, **WriteParams_t.related_sample_identity**, **WriteParams_t.related_source_guid**, **Time_t.↩ sec**, **WriteParams_t.source_guid**, **WriteParams_t.source_timestamp**, **Cookie_t.value**, and **WriteParams_t.↩ WriteParams_t()**.

8.413.4 Member Data Documentation

8.413.4.1 identity

```
SampleIdentity_t identity
```

Initial value:

```
=
        new SampleIdentity_t(SampleIdentity_t.AUTO_SAMPLE_IDENTITY)
```

Identity of the sample.

Identifies the sample being written. The identity consists of a pair (Virtual Writer GUID, Virtual Sequence Number).

Use the default value to let RTI Connext determine the sample identity as follows:

- The Virtual Writer GUID is the virtual GUID associated with the writer writing the sample. This virtual GUID is configured using **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid** (p. 598).
- The sequence number is increased by one with respect to the previous value.

The virtual sequence numbers for a virtual writer must be strictly monotonically increasing. If the user tries to write a sample with a sequence number smaller or equal to the last sequence number, the write operation will fail.

A DataReader can access the identity of a received sample by using the fields **com.rti.dds.subscription.Sample↩ Info.original_publication_virtual_guid** (p. 1644) and **com.rti.dds.subscription.SampleInfo.original_publication↩ _virtual_sequence_number** (p. 1644) in the **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] com.rti.dds.infrastructure.SampleIdentity_t.SampleIdentity_t.AUTO_SAMPLE_IDENTITY.

Referenced by **WriteParams_t.copy_from()**, **Requester**< **TReq, TRep** >**.sendRequest()**, and **WriteParams_t.↩ WriteParams_t()**.

8.413.4.2 related_sample_identity

```
final  SampleIdentity_t related_sample_identity
```

Initial value:

```
=
       new SampleIdentity_t(SampleIdentity_t.UNKNOWN_SAMPLE_IDENTITY)
```

The identity of another sample related to this one.

Identifies another sample that is logically related to the one that is written.

When this field is set, the related sample identity is propagated and subscribing applications can retrieve it from the **com.rti.dds.subscription.SampleInfo** (p. 1634) (see com.rti.dds.subscription.SampleInfo.get_related_sample_↩ identity).

The default value is com.rti.dds.infrastructure.SampleIdentity_t.SampleIdentity_t.UNKNOWN_SAMPLE_IDENTITY, and is not propagated.

A DataReader can access the related identity of a received sample by using the fields **com.rti.dds.subscription.↩ SampleInfo.related_original_publication_virtual_guid** (p. 1645) and **com.rti.dds.subscription.SampleInfo.↩ related_original_publication_virtual_sequence_number** (p. 1645) in the **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] com.rti.dds.infrastructure.SampleIdentity_t.SampleIdentity_t.UNKNOWN_SAMPLE_IDENTITY

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.3 source_timestamp

```
Time_t source_timestamp = new  Time_t( Time_t.TIME_INVALID)
```

Source timestamp upon write.

Specifies the source timestamp that will be available to the **com.rti.dds.subscription.DataReader** (p. 450) objects by means of the `source_timestamp` attribute within the **com.rti.dds.subscription.SampleInfo** (p. 1634).

[default] com.rti.dds.infrastructure.Time_t.INVALID.

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.4 handle

```
InstanceHandle_t handle = new  InstanceHandle_t()
```

Instance handle.

Either the handle returned by a previous call to **com.rti.ndds.example.FooDataWriter.register_instance** (p. 1098), or else the special value **com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL** (p. 1156).

[default] com.rti.dds.infrastructure.InstanceHandle_t.HANDLE_NIL (p. 1156)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.5 priority

```
int priority =  UNDEFINED
```

Publication priority.

A positive integer value designating the relative priority of the **com.rti.dds.publication.DataWriter** (p. 553), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (com.rti.dds.infrastructure.PublishModeQosPolicy↩ Kind.PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS) with **com.rti.dds.publication.Flow↩ ControllerProperty_t.scheduling_policy** (p. 1059) set to com.rti.dds.publication.FlowControllerSchedulingPolicy.↩ FlowControllerSchedulingPolicy.HPF_FLOW_CONTROLLER_SCHED_POLICY.

Larger numbers have higher priority.

For multi-channel DataWriters, if the publication priority of any channel is set to any value other than com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the channel's priority will take precedence over that of the DataWriter.

For multi-channel DataWriters, if the publication priority of any channel is com.rti.dds.infrastructure.PUBLICATION_↩ PRIORITY_UNDEFINED, then the channel will inherit the publication priority of the DataWriter.

If the publicaton priority of the DataWriter, and of any channel of a multi-channel DataWriter, are com.rti.dds.↩ infrastructure.PUBLICATION_PRIORITY_UNDEFINED, then the priority of the DataWriter or DataWriter channel will be assigned the lowest priority value.

If the publication priority of the DataWriter is com.rti.dds.infrastructure.PUBLICATION_PRIORITY_AUTOMATIC, then the DataWriter will be assigned the priority of the largest publication priority of all samples in the DataWriter.

The publication priority of each sample can be set in the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of the **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110) function.

For dispose and unregister samples, use the **com.rti.dds.infrastructure.WriteParams_t** (p. 1994) of **com.rti.ndds.↩ example.FooDataWriter.dispose_w_params** (p. 1114) and **com.rti.ndds.example.FooDataWriter.unregister_↩ instance_w_params** (p. 1104).

[default] com.rti.dds.infrastructure.PUBLICATION_PRIORITY_UNDEFINED

[range] [-1, MAX_INT]

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.6 flag

```
int flag = 0
```

Flags associated with the sample.

The flags are represented as a 32-bit integer, of which only the 16 least-significant bits are used.

RTI reserves least-significant bits [0-7] for middleware-specific usage.

The application can use least-significant bits [8-15].

The first bit (**com.rti.dds.infrastructure.SampleFlagBits.REDELIVERED_SAMPLE** (p. 1631)) is reserved for marking samples as redelivered when using RTI Queuing Service.

The second bit (**com.rti.dds.infrastructure.SampleFlagBits.INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1631)) is used to indicate that a response sample is not the last response sample for a given request. This bit is usually set by a Replier sending multiple responses for a request.

An application can inspect the flags associated with a received sample by checking the field **com.rti.dds.↩ subscription.SampleInfo.flag** (p. 1645).

[default] 0 (no flags are set)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.7 source_guid

```
GUID_t source_guid = new GUID_t( GUID_t.GUID_AUTO)
```

Identifies the application logical data source associated with the sample being written.

When this field is set, the source_guid is propagated and subscribing applications can retrieve it from the **com.rti.↩ dds.subscription.SampleInfo** (p. 1634) (see **com.rti.dds.subscription.SampleInfo.source_guid** (p. 1646)).

The default value is **com.rti.dds.infrastructure.GUID_t.GUID_AUTO** (p. 1134), and is not propagated.

The main use case for source_guid and related_source_guid is a request/reply scenario in which a reply has to be sent only to the Requester that issue the related request.

In this case, the Requester's DataWriter will send a request setting the source_guid to a unique value. This value must be the same value even after Requester restart.

The Replier's DataReader will get the request's source_guid from the SampleInfo and it will send it as the related_↩ source_guid of the reply using the Replier's DataWriter.

The Requester's DataReader will install a CFT on the related_source_guid using a filter expression. For example:
`@related_source_guid.value = &hex(00000000000000000000000000000001)`

This way the reply will be send only to the right Requester.

The source_guid and related_source_guid fields are used by RTI Queuing Service in a request/reply scenario.

[default] com.rti.dds.infrastructure.GUID_t.GUID_AUTO (p. 1134) (the source_guid is automatically set to the **com.rti.dds.publication.DataWriter** (p. 553) virtual GUID).

See also

com.rti.dds.infrastructure.WriteParams_t.related_source_guid (p. 2000)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.8 related_source_guid

```
GUID_t related_source_guid = new  GUID_t()
```

Identifies the application logical data source that is related to the sample being written.

When this field is set, the related_source_guid is propagated and subscribing applications can retrieve it from the **com.rti.dds.subscription.SampleInfo** (p. 1634) (see **com.rti.dds.subscription.SampleInfo.related_source_guid** (p. 1646)).

The default value is **com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134), and is not propagated.

[default] com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN (p. 1134)

See also

com.rti.dds.infrastructure.WriteParams_t.source_guid (p. 2000)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.413.4.9 related_reader_guid

```
GUID_t related_reader_guid = new  GUID_t()
```

Identifies a DataReader that is logically related to the sample that is being written.

When this field is set, the related_reader_guid is propagated and subscribing applications can retrieve it from the **com.rti.dds.subscription.SampleInfo** (p. 1634) (see **com.rti.dds.subscription.SampleInfo.related_subscription_guid** (p. 1646)).

The default value is **com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN** (p. 1134), and is not propagated.

The main use case for this field is point-to-point sample distribution using CFT. DataReaders install a CFT on the related_reader_guid using a unique GUID. For example, the filter for DataReader 'n' can be:

```
@related_reader_guid.value = &hex(00000000000000000000000000000001)
```

Then, a DataWriter that wants to send the sample to DataReader 'n' will use the **com.rti.ndds.example.FooDataWriter.write_w_params** (p. 1110) method and set related_reader_guid to the value used by DataReader 'n' in its filter expression.

This field is currently used by RTI Queuing Service to distribute a sample to only one of the Consumer's DataReaders attached to a SharedReaderQueue.

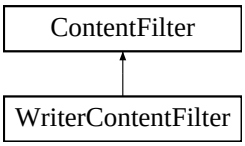
[default] com.rti.dds.infrastructure.GUID_t.GUID_UNKNOWN (p. 1134)

Referenced by **WriteParams_t.copy_from()**, and **WriteParams_t.WriteParams_t()**.

8.414 WriterContentFilter Interface Reference

<<***interface***>> *(*p. *156)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

Inheritance diagram for WriterContentFilter:



Public Member Functions

- void **writer_attach** (**ObjectHolder** writer_filter_data)

  A writer-side filtering API to create some state that can facilitate filtering on the writer side.
- void **writer_detach** (Object writer_filter_data)

  A writer-side filtering API to clean up a previously created state using ***com.rti.dds.topic.WriterContentFilter.writer_attach*** *(*p. *2003).*
- void **writer_compile** (Object writer_filter_data, **ExpressionProperty** prop, String expression, **StringSeq** parameters, **TypeCode** type_code, String type_class_name, **Cookie_t** cookie)

  A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching ***com.rti.dds.subscription.DataReader*** *(*p. *450).*
- **CookieSeq writer_evaluate** (Object writer_filter_data, Object sample, **FilterSampleInfo** meta_data)

  A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.
- void **writer_finalize** (Object writer_filter_data, **Cookie_t** cookie)

  A writer-side filtering API to clean up a previously compiled instance of the content filter.

8.414.1 Detailed Description

<<***interface***>> *(*p. *156)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 436).

Entity:

com.rti.dds.topic.ContentFilteredTopic (p. 436)

This interface can be implemented by an application-provided class and then registered with the **com.rti.dds.domain.DomainParticipant** (p. 670) such that samples can be filtered for a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) with the given filter name.

Note: The API for using a custom content filter is subject to change in a future release.

See also

com.rti.dds.topic.ContentFilteredTopic (p. 436)

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 740)

8.414.2 Member Function Documentation

8.414.2.1 writer_attach()

```
void writer_attach (
             ObjectHolder writer_filter_data )
```

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

This method is called to create some state required to perform filtering on the writer side using writer-side filtering APIs. This method will be called for every **com.rti.dds.publication.DataWriter** (p. 553); it will be called only the first time the **com.rti.dds.publication.DataWriter** (p. 553) matches a **com.rti.dds.subscription.DataReader** (p. 450) using the specified filter. This function will not be called for any subsequent DataReaders that match the DataWriter and are using the same filter.

Parameters

writer_filter_data	*<<**out**>> (p. 156)* A user-specified opaque pointer to some state created on the **com.rti.dds.publication.DataWriter** (p. 553) that will help perform writer-side filtering efficiently.

8.414.2.2 writer_detach()

```
void writer_detach (
             Object writer_filter_data )
```

A writer-side filtering API to clean up a previously created state using **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003).

This method is called to delete any state created using the **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003) function. This method will be called when the **com.rti.dds.publication.DataWriter** (p. 553) is deleted.

Parameters

writer_filter_data	*<<**in**>> (p. 156)* A pointer to the state created using **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003).

8.414.2.3 writer_compile()

```
void writer_compile (
             Object writer_filter_data,
```

```
 ExpressionProperty prop,
String expression,
 StringSeq parameters,
 TypeCode type_code,
String type_class_name,
 Cookie_t cookie )
```

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **com.rti.dds.subscription.DataReader** (p. 450).

This method is called when the **com.rti.dds.publication.DataWriter** (p. 553) discovers a **com.rti.dds.subscription.↩ DataReader** (p. 450) with a **com.rti.dds.topic.ContentFilteredTopic** (p. 436) or when a **com.rti.dds.publication.↩ DataWriter** (p. 553) is notified of a change in a DataReader's filter parameter for the locally registered content filter instance.

It is possible for multiple threads to be calling into this function at the same time.

Parameters

writer_filter_data	*<<**in**>> (p. 156)* A pointer to the state created using **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003) .
prop	*<<**out**>> (p. 156)* A pointer to **com.rti.dds.topic.ExpressionProperty** (p. 1046) that allows you to indicate to RTI Connext if a filter expression can be optimized.
expression	*<<**in**>> (p. 156)* An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connext and **must** not be freed. If you want to manipulate this string, you **must** first make a copy of it.
parameters	*<<**in**>> (p. 156)* A string sequence with the expression parameters with which the **com.rti.dds.topic.ContentFilteredTopic** (p. 436) was created. The string sequence is **equal** (but **not** identical) to the string sequence passed to **com.rti.dds.domain.DomainParticipant.create_contentfilteredtopic** (p. 710). Note that the sequence passed to the compile function is **owned** by RTI Connext and **must not** be referenced outside the compile function.
type_code	*<<**in**>> (p. 156)* A pointer to the type code for the related **com.rti.dds.topic.Topic** (p. 1807) of the **com.rti.dds.topic.ContentFilteredTopic** (p. 436). A type_code is a description of a type in terms of which types it contains (such as long, string, etc.) and the corresponding member field names in the data type structure. The type code can be used to write custom content filters that can be used with any type. This parameter is always NULL in Java.
type_class_name	*<<**in**>> (p. 156)* Fully qualified class name of the related **com.rti.dds.topic.Topic** (p. 1807).
cookie	*<<**in**>> (p. 156)* **com.rti.dds.infrastructure.Cookie_t** (p. 442) to uniquely identify **com.rti.dds.subscription.DataReader** (p. 450) for which **com.rti.dds.topic.WriterContentFilter.writer_compile** (p. 2003) was called.

Exceptions

One	of the **Standard Return Codes** (p. 261)

8.414.2.4 writer_evaluate()

```
CookieSeq writer_evaluate (
            Object writer_filter_data,
            Object sample,
             FilterSampleInfo meta_data )
```

A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.

This method is called every time a **com.rti.dds.publication.DataWriter** (p. 553) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **com.rti.dds.publication.DataWriter** (p. 553) is performing writer-side filtering and return the list of **com.rti.dds.infrastructure.Cookie_t** (p. 442) structures associated with the DataReaders whose filters pass the sample.

It is possible for multiple threads to be calling into this function at the same time

Parameters

writer_filter_data	*<<**in**>> (p. 156)* A pointer to the state created using **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003) .
sample	*<<**in**>> (p. 156)* Pointer to a deserialized sample to be filtered.
meta_data	*<<**in**>> (p. 156)* Pointer to meta data associated with the sample.

Returns

The function returns **com.rti.dds.infrastructure.CookieSeq** (p. 443) which identifies the set of DataReaders whose filters pass the sample.

8.414.2.5 writer_finalize()

```
void writer_finalize (
            Object writer_filter_data,
             Cookie_t cookie )
```

A writer-side filtering API to clean up a previously compiled instance of the content filter.

This method is called to notify the filter implementation that the **com.rti.dds.publication.DataWriter** (p. 553) is no longer matching with a **com.rti.dds.subscription.DataReader** (p. 450) for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the **com.rti.dds.subscription.DataReader** (p. 450).

It is possible for multiple threads to be calling into this function at the same time.

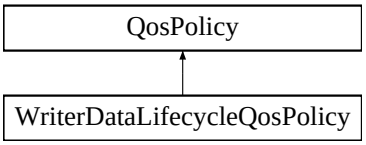
Parameters

writer_filter_data	*<<**in**>> (p. 156)* A pointer to the state created using **com.rti.dds.topic.WriterContentFilter.writer_attach** (p. 2003).
cookie	**com.rti.dds.infrastructure.Cookie_t** (p. 442) to uniquely identify
Generated by Doxygen	**com.rti.dds.subscription.DataReader** (p. 450) for which **com.rti.dds.topic.WriterContentFilter.writer_finalize** (p. 2005) was called.

8.415 WriterDataLifecycleQosPolicy Class Reference

Controls how a **com.rti.dds.publication.DataWriter** (p. 553) handles the lifecycle of the instances (keys) that it is registered to manage.

Inheritance diagram for WriterDataLifecycleQosPolicy:



Public Attributes

- boolean **autodispose_unregistered_instances**

  *Boolean flag that controls the behavior when the **com.rti.dds.publication.DataWriter** (p. 553) unregisters an instance by means of the unregister operations.*
- final **Duration_t autopurge_unregistered_instances_delay**

  *<<**extension**>> (p. 155) Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance once it has unregistered the instance.*
- final **Duration_t autopurge_disposed_instances_delay**

  *<<**extension**>> (p. 155) Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance once it has disposed the instance.*

8.415.1 Detailed Description

Controls how a **com.rti.dds.publication.DataWriter** (p. 553) handles the lifecycle of the instances (keys) that it is registered to manage.

Entity:

com.rti.dds.publication.DataWriter (p. 553)

Properties:

RxO (p. 256) = N/A
Changeable (p. 256) = **YES** (p. 256)

8.415.2 Usage

This policy determines how the **com.rti.dds.publication.DataWriter** (p. 553) acts with regards to the lifecycle of the data instances it manages (data instances that have been either explicitly registered with the **com.rti.dds.publication.Data↩ Writer** (p. 553) or implicitly registered by directly writing the data).

You may use **com.rti.ndds.example.FooDataWriter.unregister_instance** (p. 1100) to indicate that the **com.rti.dds.↩ publication.DataWriter** (p. 553) no longer wants to send data for a **com.rti.dds.topic.Topic** (p. 1807).

The behavior controlled by this QoS policy applies on a per instance (key) basis for keyed Topics, so that when a **com.rti.dds.publication.DataWriter** (p. 553) unregisters an instance, RTI Connext can automatically also dispose that instance. This is the default behavior.

In many cases where the ownership of a Topic is shared (see **com.rti.dds.infrastructure.OwnershipQosPolicy** (p. 1342)), DataWriters may want to relinquish their ownership of a particular instance of the Topic to allow other Data↩ Writers to send updates for the value of that instance regardless of Ownership Strength. In that case, you may only want a DataWriter to unregister an instance without disposing the instance. *Disposing* an instance is a statement that an instance no longer exists. User applications may be coded to trigger on the disposal of instances, thus the ability to unregister without disposing may be useful to properly maintain the semantic of disposal.

8.415.3 Member Data Documentation

8.415.3.1 autodispose_unregistered_instances

```
boolean autodispose_unregistered_instances
```

Boolean flag that controls the behavior when the **com.rti.dds.publication.DataWriter** (p. 553) unregisters an instance by means of the unregister operations.

- com.rti.dds.infrastructure.true
  The **com.rti.dds.publication.DataWriter** (p. 553) will dispose of the instance each time it is unregistered. The behavior is identical to explicitly calling one of the `dispose` operations on the instance prior to calling the `unregister` operation.
- com.rti.dds.infrastructure.false (default)
  The **com.rti.dds.publication.DataWriter** (p. 553) will not dispose of the instance. The application can still call one of the dispose operations prior to unregistering the instance and dispose of the instance that way.

[default] com.rti.dds.infrastructure.false

8.415.3.2 autopurge_unregistered_instances_delay

```
final Duration_t autopurge_unregistered_instances_delay
```

<<***extension***>> *(p. 155)* Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance once it has unregistered the instance.

Determines how long the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance that has been unregistered. By default, the **com.rti.dds.publication.DataWriter** (p. 553) resources associated with an instance (e.g., the space needed to remember the Instance Key or KeyHash) are released lazily. This means the resources are only reclaimed when the space is needed for another instance because **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) is exceeded. This behavior can be changed by setting autopurge_unregistered_instances_delay to a value other than **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846).

After this time elapses, the **com.rti.dds.publication.DataWriter** (p. 553) will purge all internal information regarding the instance, including historical samples, even if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1592) has not been reached.

The purging of unregistered instances can be done based on the source timestamp of the unregister sample or the time where the unregister sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

For durable writer history, autopurge_unregistered_instances_delay supports only the **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) value.

[default] **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) (disabled) for all **com.rti.dds.publication.DataWriter** (p. 553) except for the built-in discovery DataWriters
com.rti.dds.infrastructure.Duration_t.ZERO for built-in discovery DataWriters (see **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer_data_lifecycle** (p. 653), **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer_data_lifecycle** (p. 654) and **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_configuration_writer_data_lifecycle** (p. 664)).

[range] [0, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.415.3.3 autopurge_disposed_instances_delay

```
final Duration_t autopurge_disposed_instances_delay
```

<<***extension***>> *(p. 155)* Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance once it has disposed the instance.

Determines how long the **com.rti.dds.publication.DataWriter** (p. 553) will maintain information regarding an instance that has been disposed of. By default, disposing of an instance does not make it eligible to be purged. By setting autopurge_disposed_instances_delay to a value other than **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846), the DataWriter will delete the resources associated with an instance (including historical samples) once the time has elapsed and all matching DataReaders have acknowledged all the samples for this instance including the dispose sample.

The purging of disposed instances can be done based on the source timestamp of the dispose sample or the time when the dispose sample was added to the DataWriter queue by setting the following property to 1 or 0 respectively (default: 0): **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**.

This QoS value is supported with durable DataWriter queues only for com.rti.dds.infrastructure.Duration_t.ZERO and **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) values (finite values are not supported).

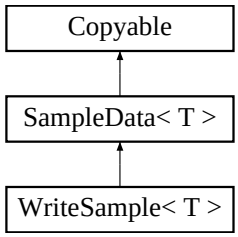
[default] **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846) (disabled)

[range] [0, 1 year] or **com.rti.dds.infrastructure.Duration_t.DURATION_INFINITE** (p. 846)

8.416 WriteSample< T > Interface Template Reference

A sample for writing data.

Inheritance diagram for WriteSample< T >:



Public Member Functions

- **WriteParams_t getInfo** ()
  Gets the write parameters.
- void **setInfo** (**WriteParams_t** info)
  Replaces the parameters with a new value.
- void **setData** (T data)
  Sets the data to be written.

8.416.1 Detailed Description

A sample for writing data.

WriteSamples are value types containing data that can be sent and optional parameters that configure how the data is written.

Template Parameters

T	The data type that this sample contains

8.416.2 Member Function Documentation

8.416.2.1 getInfo()

```
WriteParams_t getInfo ( )
```

Gets the write parameters.

The parameters are input and output to write operations. They can be set to configure the write operation (e.g. setting a specific publication timestamp) or they can be looked up afterward (e.g. what was the actual timestamp the sample was written with).

See also

com.rti.connext.requestreply.Requester<TReq,TRep>.sendRequest(WriteSample<TReq>) (p. 1568)

com.rti.connext.requestreply.Replier<TReq,TRep>.sendReply(WriteSample<TRep>, SampleIdentity_t) (p. 1547)

Referenced by **Replier< TReq, TRep >.sendReply()**, and **Requester< TReq, TRep >.sendRequest()**.

8.416.2.2 setInfo()

```
void setInfo (
              WriteParams_t info )
```

Replaces the parameters with a new value.

8.416.2.3 setData()

```
void setData (
             T data )
```

Sets the data to be written.

Replaces the current data with a new element.

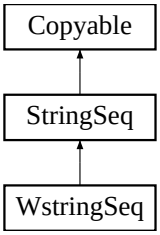
Parameters

data	The new data

8.417 WstringSeq Class Reference

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ char* >

Inheritance diagram for WstringSeq:



Public Member Functions

- **WstringSeq** ()

  Constructs an empty sequence of wide strings with an initial maximum of zero.
- **WstringSeq** (int initialMaximum)

  Constructs an empty sequence of wide strings with the given initial maximum.
- **WstringSeq** (Collection strings)

  Constructs a new sequence containing the given wide strings.

Static Public Member Functions

- static void **readWstringArray** (String[ ] value, CdrObjectInput in, int length) throws IOException
- static void **writeWstringArray** (String[ ] value, CdrObjectOutput out, int length, int maxStringLength) throws IOException

8.417.1 Detailed Description

Instantiates `com.rti.dds.infrastructure.com.rti.dds.util.Sequence` < com.rti.dds.infrastructure.↩ char∗ >

Instantiates:

<<***generic***>> *(*p. *156)* com.rti.dds.infrastructure.com.rti.dds.util.Sequence

See also

com.rti.dds.infrastructure.char

com.rti.dds.infrastructure.StringSeq (p. 1718)

com.rti.dds.infrastructure.com.rti.dds.util.Sequence

8.417.2 Constructor & Destructor Documentation

8.417.2.1 WstringSeq() [1/3]

```
WstringSeq ( )
```

Constructs an empty sequence of wide strings with an initial maximum of zero.

8.417.2.2 WstringSeq() [2/3]

```
WstringSeq (
            int initialMaximum )
```

Constructs an empty sequence of wide strings with the given initial maximum.

8.417.2.3 WstringSeq() [3/3]

```
WstringSeq (
            Collection strings )
```

Constructs a new sequence containing the given wide strings.

Parameters

strings	the initial contents of this sequence

8.417.3 Member Function Documentation

8.417.3.1 readWstringArray()

```
static void readWstringArray (
            String[] value,
            CdrObjectInput in,
            int length ) throws IOException  [static]
```

Read array of strings. The length specified **must** match the expected length of array. Otherwise, the stream will be positioned incorrectly, leading to corrupt reads. The length of array must be at least the value of length parameter (otherwise, ArrayOutOfBoundException will be thrown).

Parameters

value	array to read into
in	Interface for reading object in CDR encoding.
length	the length of array (<= value.length)

8.417.3.2 writeWstringArray()

```
static void writeWstringArray (
             String[] value,
             CdrObjectOutput out,
             int length,
             int maxStringLength ) throws IOException  [static]
```

Write array of wstring up to the specified length.

Chapter 9

Example Documentation

9.1 HelloWorld.idl

9.1.1 IDL Type Description

The data type to be disseminated by RTI Connext is described in language-independent IDL. The IDL file is input to rtiddsgen (see the `Code Generator User's Manual` for more information), which produces the following files.

The programming language specific type representation of the type Foo = HelloWorld, for use in the application code.

- HelloWorld.java
- HelloWorldSeq.java

User Data Type Support (p. 56) types as required by the DDS specification for use in the application code.

- HelloWorldTypeSupport.java
- HelloWorldDataWriter.java
- HelloWorldDataReader.java

9.1.1.1 HelloWorld.idl

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

9.2 HelloWorldSeq.java

9.2.1 Programming Language Type Description

The following programming language specific type representation is generated by rtiddsgen (see the `Code Generator User's Manual` for more information) for use in application code, where:

- Foo = HelloWorld
- FooSeq = HelloWorldSeq

9.2.1.1 HelloWorld.java

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

This file was generated from .idl
using RTI Code Generator (rtiddsgen) version 4.3.0.
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.
*/
import com.rti.dds.infrastructure.*;
import com.rti.dds.infrastructure.Copyable;
import java.io.Serializable;
import com.rti.dds.cdr.CdrHelper;
// Depending on the type represented in the IDL, we may perform some redundant
// casts, we are suppressing that warning
@SuppressWarnings("cast")
public class HelloWorld   implements Copyable, Serializable{
    private static final long serialVersionUID = -742582614L;
    public String msg= (String)""; /* maximum length = (128) */
    public HelloWorld() {
    }
    public HelloWorld (HelloWorld other) {
        this();
        copy_from(other);
    }
    public static java.lang.Object create() {
        HelloWorld self;
        self = new  HelloWorld();
        self.clear();
        return self;
    }
    public void clear() {
        msg = (String)"";
    }
    @Override
    public boolean equals(java.lang.Object o) {
        if (o == null) {
            return false;
        }
        if(getClass() != o.getClass()) {
            return false;
        }
        HelloWorld otherObj = (HelloWorld)o;
        if(!this.msg.equals(otherObj.msg)) {
            return false;
        }
        return true;
    }
    @Override
    public int hashCode() {
        final int __prime = 31;
        int __result = 1;
        __result = __prime * __result + msg.hashCode();
        return __result;
    }
    public java.lang.Object copy_from(java.lang.Object src) {
        HelloWorld typedSrc = (HelloWorld) src;
        HelloWorld typedDst = this;
        typedDst.msg = typedSrc.msg;
        return this;
    }
    @Override
    public java.lang.String toString(){
        return toString("", 0);
    }
    public java.lang.String toString(java.lang.String desc, int indent) {
        java.lang.StringBuffer strBuffer = new java.lang.StringBuffer();
        if (desc != null) {
            CdrHelper.printIndent(strBuffer, indent);
            strBuffer.append(desc).append(":\n");
        }
        CdrHelper.printIndent(strBuffer, indent+1);
        strBuffer.append("msg: ").append(this.msg).append("\n");
        return strBuffer.toString();
    }
}
```

9.2.1.2 HelloWorldSeq.java

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

This file was generated from .idl
using RTI Code Generator (rtiddsgen) version 4.3.0.
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.
*/
import java.util.Collection;
import com.rti.dds.infrastructure.Copyable;
import com.rti.dds.util.Enum;
import com.rti.dds.util.Sequence;
import com.rti.dds.util.LoanableSequence;
public final class HelloWorldSeq extends LoanableSequence implements Copyable {
    private static final long serialVersionUID = 2003772976L;
    // --------------------------------------------------------------------
    // Package Fields
    // --------------------------------------------------------------------
    /*package*/ transient Sequence _loanedInfoSequence = null;
    // --------------------------------------------------------------------
    // Public Fields
    // --------------------------------------------------------------------
    // --- Constructors: ------------------------------------------------
    public HelloWorldSeq() {
        super(HelloWorld.class);
    }
    public HelloWorldSeq (int initialMaximum) {
        super(HelloWorld.class, initialMaximum);
    }
    public HelloWorldSeq (Collection<?> elements) {
        super(HelloWorld.class, elements);
    }
    public  HelloWorld get(int index) {
        return (HelloWorld) super.get(index);
    }
    // --- From Copyable: -----------------------------------------------
    @Override
    public java.lang.Object copy_from(java.lang.Object src) {
        Sequence typedSrc = (Sequence) src;
        final int srcSize = typedSrc.size();
        final int origSize = size();
        // if this object's size is less than the source, ensure we have
        // enough room to store all of the objects
        if (getMaximum() < srcSize) {
            setMaximum(srcSize);
        }
        // trying to avoid clear() method here since it allocates memory
        // (an Iterator)
        // if the source object has fewer items than the current object,
        // remove from the end until the sizes are equal
        if (srcSize < origSize){
            removeRange(srcSize, origSize);
        }
        // copy the data from source into this (into positions that already
        // existed)
        for(int i = 0; (i < origSize) && (i < srcSize); i++){
            if (typedSrc.get(i) == null){
                set(i, null);
            } else {
                // check to see if our entry is null, if it is, a new instance has to be allocated
                if (get(i) == null){
                    set(i, HelloWorld.create());
                }
                set(i, ((Copyable) get(i)).copy_from(typedSrc.get(i)));
            }
        }
        // copy 'new' HelloWorld objects (beyond the original size of this object)
        for(int i = origSize; i < srcSize; i++){
            if (typedSrc.get(i) == null) {
                add(null);
            } else {
                // NOTE: we need to create a new object here to hold the copy
                add(HelloWorld.create());
                // we need to do a set here since enums aren't truely Copyable
                set(i, ((Copyable) get(i)).copy_from(typedSrc.get(i)));
            }
        }
        return this;
```

```
    }
}
```

9.3 HelloWorldDataReader.java

9.3.1 User Data Type Support

Files generated by rtiddsgen (see the `Code Generator User's Manual` for more information) that implement the type specific APIs required by the DDS specification, as described in the **User Data Type Support** (p. 56), where:

- FooTypeSupport = HelloWorldTypeSupport
- FooDataWriter = HelloWorldDataWriter
- FooDataReader = HelloWorldDataReader

9.3.1.1 HelloWorldTypeSupport.java

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

This file was generated from .idl
using RTI Code Generator (rtiddsgen) version 4.3.0.
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.
*/
import com.rti.dds.cdr.CdrEncapsulation;
import com.rti.dds.cdr.CdrInputStream;
import com.rti.dds.cdr.CdrOutputStream;
import com.rti.dds.cdr.CdrPrimitiveType;
import com.rti.dds.cdr.CdrBuffer;
import com.rti.dds.cdr.CdrHeader;
import com.rti.dds.dynamicdata.DynamicData;
import com.rti.dds.cdr.IllegalCdrStateException;
import com.rti.dds.publication.DataWriter;
import com.rti.dds.publication.DataWriterListener;
import com.rti.dds.subscription.DataReader;
import com.rti.dds.subscription.DataReaderListener;
import com.rti.dds.topic.DefaultEndpointData;
import com.rti.dds.topic.TypeSupportImpl;
import com.rti.dds.topic.TypeSupportType;
import com.rti.dds.infrastructure.*;
import com.rti.dds.infrastructure.RETCODE_ERROR;
import com.rti.dds.topic.PrintFormatProperty;
import com.rti.dds.topic.PrintFormatKind;
import com.rti.dds.typecode.TypeCode;
import com.rti.dds.typecode.ExtensibilityKind;
import com.rti.dds.topic.TypeSupportParticipantInfo;
import com.rti.dds.topic.TypeSupportEndpointInfo;
import com.rti.dds.domain.DomainParticipant;
public class HelloWorldTypeSupport extends TypeSupportImpl {
    // ----------------------------------------------------------------------
    // Private Fields
    // ----------------------------------------------------------------------
    private static final java.lang.String TYPE_NAME = "HelloWorld";
    private static final char[] PLUGIN_VERSION = {2, 0, 0, 0};
    private static final HelloWorldTypeSupport _singleton
    = new HelloWorldTypeSupport();
    // ----------------------------------------------------------------------
    // Public Methods
    // ----------------------------------------------------------------------
    // --- External methods: ----------------------------------------------
    /* The methods in this section are for use by users of RTI Connext
    */
    public static java.lang.String get_type_name() {
        return _singleton.get_type_nameI();
```

```
}
public static void register_type(DomainParticipant participant,
java.lang.String type_name) {
    _singleton.register_typeI(participant, type_name);
}
public static void unregister_type(DomainParticipant participant,
java.lang.String type_name) {
    _singleton.unregister_typeI(participant, type_name);
}
/* The methods in this section are for use by RTI Connext
* itself and by the code generated by rtiddsgen for other types.
* They should be used directly or modified only by advanced users and are
* subject to change in future versions of RTI Connext.
*/
public static HelloWorldTypeSupport get_instance() {
    return _singleton;
}
public static HelloWorldTypeSupport getInstance() {
    return get_instance();
}
public static TypeCode getTypeCode(){
    return HelloWorldTypeCode.VALUE;
}
@Override
public java.lang.Object create_data() {
    return HelloWorld.create();
}
@Override
public void destroy_data(java.lang.Object data) {
    return;
}
@Override
public java.lang.Object create_key() {
    return new HelloWorld();
}
@Override
public void destroy_key(java.lang.Object key) {
    return;
}
@Override
public java.lang.Class<?> get_type() {
    return HelloWorld.class;
}
@Override
public java.lang.Object copy_data(java.lang.Object destination, java.lang.Object source) {
    HelloWorld typedDst = (HelloWorld) destination;
    HelloWorld typedSrc = (HelloWorld) source;
    return typedDst.copy_from(typedSrc);
}
@Override
public long get_serialized_sample_max_size(java.lang.Object endpoint_data,boolean
   include_encapsulation,short final_encapsulation_id,long currentAlignment) {
    CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
    short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
        final_encapsulation_id,
        ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
    boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
    DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
    if (epd == null) {
        /* Coverity reports that epd store a object that might not
        be used later. It is true that it might not be used,
        depending on the type complexity. This is intentional
        because it doesn't affect generated code and fixing it
        would make generated code more difficult to maintain */
        /* coverity[defect] */
        epd = new DefaultEndpointData();
    }
    long origAlignment = currentAlignment;
    long encapsulation_size = currentAlignment;
    if(include_encapsulation) {
        if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
            throw new RETCODE_ERROR("Unsupported encapsulation");
        }
        encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
        encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
        encapsulation_size -= currentAlignment;
        currentAlignment = 0;
        origAlignment = 0;
        if(xcdr1){
            epd.setBaseAlignment(currentAlignment);
        }
```

```
        }
        if (!xcdr1) {
            currentAlignment += _cdrPrimitiveType.getIntMaxSizeSerialized(currentAlignment);
        }
        currentAlignment +=_cdrPrimitiveType.getStringMaxSizeSerialized(epd.getAlignment(currentAlignment),
       (128)+1);
        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }
        return currentAlignment - origAlignment;
    }
    @Override
    public long get_serialized_sample_min_size(java.lang.Object endpoint_data,boolean
       include_encapsulation,short final_encapsulation_id,long currentAlignment) {
        CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
        short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
            final_encapsulation_id,
            ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        long origAlignment = currentAlignment;
        long encapsulation_size = currentAlignment;
        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;
            if(xcdr1){
                epd.setBaseAlignment(currentAlignment);
            }
        }
        if (!xcdr1) {
            currentAlignment += _cdrPrimitiveType.getIntMaxSizeSerialized(currentAlignment);
        }
        currentAlignment += _cdrPrimitiveType.getStringMaxSizeSerialized(epd.getAlignment(currentAlignment), 1);
        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }
        return currentAlignment - origAlignment;
    }
    @Override
    public long get_serialized_sample_size(
        java.lang.Object endpoint_data, boolean include_encapsulation,
        short final_encapsulation_id, long currentAlignment,
        java.lang.Object sample)
    {
        CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
        short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
            final_encapsulation_id,
            ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
        HelloWorld typedSrc = (HelloWorld) sample;
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        long origAlignment = currentAlignment;
        long encapsulation_size = currentAlignment;
        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }
```

```
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;
            if(xcdr1){
                epd.setBaseAlignment(currentAlignment);
            }
        }
        if (!xcdr1) {
            currentAlignment += _cdrPrimitiveType.getIntMaxSizeSerialized(currentAlignment);
        }
        currentAlignment += _cdrPrimitiveType.getStringSerializedSize(epd.getAlignment(currentAlignment),
       typedSrc.msg );
        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }
        return currentAlignment - origAlignment;
    }
    @Override
    public long get_serialized_key_max_size(
        java.lang.Object endpoint_data,
        boolean include_encapsulation,
        short final_encapsulation_id,
        long currentAlignment)
    {
        CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
        short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
            final_encapsulation_id,
            ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        long origAlignment = currentAlignment;
        long encapsulation_size = currentAlignment;
        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;
            if(xcdr1){
                epd.setBaseAlignment(currentAlignment);
            }
        }
        currentAlignment += get_serialized_sample_max_size(
            epd,false,final_encapsulation_id,currentAlignment);
        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }
        return currentAlignment - origAlignment;
    }
    @Override
    public long get_serialized_key_for_keyhash_max_size(
        java.lang.Object endpoint_data,
        boolean include_encapsulation,
        short final_encapsulation_id,
        long currentAlignment)
    {
        CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
        short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
            final_encapsulation_id,
            ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
```

```
        would make generated code more difficult to maintain */
        /* coverity[defect] */
        epd = new DefaultEndpointData();
    }
    if (xcdr1){
        return get_serialized_key_max_size(epd, include_encapsulation, final_encapsulation_id,
   currentAlignment) ;
    }
    long origAlignment = currentAlignment;
    long encapsulation_size = currentAlignment;
    if(include_encapsulation) {
        if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
            throw new RETCODE_ERROR("Unsupported encapsulation");
        }
        encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
        encapsulation_size += _cdrPrimitiveType.getShortMaxSizeSerialized(encapsulation_size);
        encapsulation_size -= currentAlignment;
        currentAlignment = 0;
        origAlignment = 0;
    }
    currentAlignment +=_cdrPrimitiveType.getStringMaxSizeSerialized(epd.getAlignment(currentAlignment),
   (128)+1);
    if (include_encapsulation) {
        currentAlignment += encapsulation_size;
    }
    return currentAlignment - origAlignment;
}
@Override
public void serialize(java.lang.Object endpoint_data,java.lang.Object src,
CdrOutputStream dst, boolean serialize_encapsulation, short final_encapsulation_id,
boolean serialize_sample, java.lang.Object endpoint_plugin_qos) {
    int position = 0;
    int dheaderArrPosition = -1;
    int dheaderSeqPosition = -1;
    DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
    if (epd == null) {
        /* Coverity reports that epd store a object that might not
        be used later. It is true that it might not be used,
        depending on the type complexity. This is intentional
        because it doesn't affect generated code and fixing it
        would make generated code more difficult to maintain */
        /* coverity[defect] */
        epd = new DefaultEndpointData();
    }
    int dheaderPosition = -1;
    boolean inBaseClass_tmp = false;
    inBaseClass_tmp =  dst.inBaseClass;
    dst.inBaseClass = false;
    if(serialize_encapsulation) {
        dst.serializeAndSetCdrEncapsulation(final_encapsulation_id,
   ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        position = dst.resetAlignment();
    }
    if(serialize_sample) {
        boolean xcdr1 = (final_encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true:
   false;
        if (!inBaseClass_tmp && !xcdr1) {
            dheaderPosition=dst.writeDHeader();
        }
        HelloWorld typedSrc = (HelloWorld) src;
        dst.writeString(typedSrc.msg,128);
        if (!xcdr1 && dheaderPosition != -1) {
            dst.setDHeader(dheaderPosition);
            dheaderPosition = -1;
        }
    }
    if (serialize_encapsulation) {
        dst.restoreAlignment(position);
    }
}
public long serialize_to_cdr_buffer(
    byte[] buffer,
    long length,
    HelloWorld src)
{
    return super.serialize_to_cdr_buffer(buffer,length,src);
}
public long serialize_to_cdr_buffer(
    byte[] buffer,
    long length,
    HelloWorld src,
```

```
        short representation)
    {
        return super.serialize_to_cdr_buffer(
            buffer,
            length,
            src,
            representation);
    }
    @Override
    public void serialize_key(
        java.lang.Object endpoint_data,
        java.lang.Object src,
        CdrOutputStream dst,
        boolean serialize_encapsulation,
        short final_encapsulation_id,
        boolean serialize_key,
        java.lang.Object endpoint_plugin_qos)
    {
        int dheaderArrPosition = -1;
        int dheaderSeqPosition = -1;
        int position = 0;
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        boolean inBaseClass_tmp = false;
        inBaseClass_tmp =  dst.inBaseClass;
        dst.inBaseClass = false;
        if (serialize_encapsulation) {
            dst.serializeAndSetCdrEncapsulation(final_encapsulation_id,
       ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
            position = dst.resetAlignment();
        } else {
            dst.setEncapsulationKind(final_encapsulation_id);
        }
        if (serialize_key) {
            boolean xcdr1 = (final_encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true:
       false;
            HelloWorld typedSrc = (HelloWorld) src;
            dst.inBaseClass = false;
            serialize(epd, src, dst, false, final_encapsulation_id, true, endpoint_plugin_qos);
        }
        if (serialize_encapsulation) {
            dst.restoreAlignment(position);
        }
    }
    @Override
    public void serialize_key_for_keyhash(
        java.lang.Object endpoint_data,
        java.lang.Object src,
        CdrOutputStream dst,
        boolean serialize_encapsulation,
        short final_encapsulation_id,
        boolean serialize_key,
        java.lang.Object endpoint_plugin_qos)
    {
        int position = 0;
        int dheaderArrPosition = -1;
        int dheaderSeqPosition = -1;
        CdrPrimitiveType _cdrPrimitiveType = CdrPrimitiveType.getInstance(final_encapsulation_id);
        short encapsulation_id = CdrEncapsulation.getEncapsulationFromFinal(
            final_encapsulation_id,
            ExtensibilityKind.EXTENSIBLE_EXTENSIBILITY);
        boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE);
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        if (xcdr1){
```

```
            serialize_key (
                epd,
                src,
                dst,
                serialize_encapsulation,
                final_encapsulation_id,
                serialize_key,
                endpoint_plugin_qos);
        } else {
            if (serialize_encapsulation) {
                dst.serializeAndSetCdrEncapsulation(encapsulation_id);
                position = dst.resetAlignment();
            } else {
                /* We do this to prepare the stream to serialize using xcdr2 if needed
                * as in md5Stream we pass serialize_encapsulation ton false.
                */
                dst.setEncapsulationKind(final_encapsulation_id);
            }
            if (serialize_key) {
                HelloWorld typedSrc = (HelloWorld) src;
                dst.inBaseClass = false;
                dst.writeString(typedSrc.msg,128);
            }
            if (serialize_encapsulation) {
                dst.restoreAlignment(position);
            }
        }
    }
    // Depending on the type represented in the IDL, we may perform some redundant
    // casts, we are suppressing that warning
    @SuppressWarnings("cast")
    @Override
    public java.lang.Object deserialize_sample(
        java.lang.Object endpoint_data,
        java.lang.Object dst,
        CdrInputStream src, boolean deserialize_encapsulation,
        boolean deserialize_sample,
        java.lang.Object endpoint_plugin_qos)
    {
        int position = 0;
        int tmpPosition = 0, tmpSize = 0;
        long tmpLength = 0;
        CdrBuffer buffer = null;
        boolean inBaseClass_tmp = false;
        inBaseClass_tmp =  src.inBaseClass;
        src.inBaseClass = false;
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();
            position = src.resetAlignment();
        }
        if(deserialize_sample) {
            short encapsulation_id = src.getEncapsulationKind();
            boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true: false;
            if(!xcdr1){
                buffer = src.getBuffer();
            }
            HelloWorld typedDst = (HelloWorld) dst;
            typedDst.clear();
            int DHtmpPosition = 0;
            int DHtmpSize = 0;
            long DHtmpLength = 0;
            if (!xcdr1 && !inBaseClass_tmp) {
                DHtmpLength = src.readInt();
                DHtmpPosition = buffer.currentPosition();
                DHtmpSize = buffer.getDesBufferSize();
                buffer.setDesBufferSize((int)(DHtmpPosition + DHtmpLength));
            }
            try{
                typedDst.msg = src.readString(128);
            } catch (IllegalCdrStateException stateEx) {
                if (src.available() >= CdrEncapsulation.CDR_ENCAPSULATION_PARAMETER_ID_ALIGNMENT) {
```

```
                throw new RETCODE_ERROR("Error deserializing sample! Remainder: " + src.available() + "\n" +
                "Exception caused by: " + stateEx.getMessage());
            }
        } catch (java.lang.Exception ex) {
            throw new RETCODE_ERROR(ex.getMessage());
        }
        if (!xcdr1 && !inBaseClass_tmp ) {
            buffer.restore(DHtmpSize, (int) (DHtmpPosition + DHtmpLength));
        }
    }
    if (deserialize_encapsulation) {
        src.restoreAlignment(position);
    }
    return dst;
}
public void deserialize_from_cdr_buffer(
    HelloWorld dst,
    byte[] buffer,
    long length)
{
    super.deserialize_from_cdr_buffer(dst,buffer,length);
}
public java.lang.String data_to_string(
    HelloWorld sample,
    PrintFormatProperty property)
{
    return super.data_to_string(sample, property);
}
public java.lang.String data_to_string(
    HelloWorld sample)
{
    return super.data_to_string(sample);
}
@java.lang.Override
public HelloWorld data_from_string(
    java.lang.String string,
    PrintFormatKind kind)
{
    return (HelloWorld) super.data_from_string(string, kind);
}
@java.lang.Override
public HelloWorld data_from_string(
    java.lang.String string)
{
    return (HelloWorld) super.data_from_string(string);
}
@java.lang.Override
public HelloWorld data_from_dynamicdata(
    DynamicData dynamicData)
{
    return (HelloWorld) super.data_from_dynamicdata(dynamicData);
}
@Override
public java.lang.Object deserialize_key_sample(
    java.lang.Object endpoint_data,
    java.lang.Object dst,
    CdrInputStream src,
    boolean deserialize_encapsulation,
    boolean deserialize_key,
    java.lang.Object endpoint_plugin_qos)
{
    int position = 0;
    int tmpPosition = 0, tmpSize = 0;
    long tmpLength = 0;
    CdrBuffer buffer = null;
    boolean inBaseClass_tmp = false;
    inBaseClass_tmp =  src.inBaseClass;
    src.inBaseClass = false;
    DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
    if (epd == null) {
        /* Coverity reports that epd store a object that might not
        be used later. It is true that it might not be used,
        depending on the type complexity. This is intentional
        because it doesn't affect generated code and fixing it
        would make generated code more difficult to maintain */
        /* coverity[defect] */
        epd = new DefaultEndpointData();
    }
    if(deserialize_encapsulation) {
        src.deserializeAndSetCdrEncapsulation();
        position = src.resetAlignment();
```

```
        }
        if(deserialize_key) {
            short encapsulation_id = src.getEncapsulationKind();
            boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true: false;
            if(!xcdr1){
                buffer = src.getBuffer();
            }
            HelloWorld typedDst = (HelloWorld) dst;
            deserialize_sample(epd, dst, src, false, true, endpoint_plugin_qos);
        }
        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }
        return dst;
    }
    @Override
    public void skip(java.lang.Object endpoint_data,
    CdrInputStream src,
    boolean skip_encapsulation,
    boolean skip_sample,
    java.lang.Object endpoint_plugin_qos)
    {
        int position = 0;
        int tmpPosition = 0, tmpSize = 0;
        long tmpLength = 0;
        CdrBuffer buffer = null;
        boolean inBaseClass_tmp = false;
        inBaseClass_tmp =  src.inBaseClass;
        src.inBaseClass = false;
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        if (skip_encapsulation) {
            src.skipEncapsulation();
            position = src.resetAlignment();
        }
        if (skip_sample) {
            short encapsulation_id = src.getEncapsulationKind();
            boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true: false;
            if(!xcdr1){
                buffer = src.getBuffer();
            }
            int DHtmpPosition = 0;
            long DHtmpLength = 0;
            if (!xcdr1 && !inBaseClass_tmp) {
                DHtmpLength = src.readInt();
                DHtmpPosition = buffer.currentPosition();
                buffer.setCurrentPosition((int) (DHtmpPosition + DHtmpLength));
                if (skip_encapsulation) {
                    src.restoreAlignment(position);
                }
                return;
            }
            try {
                src.skipString();
            } catch (IllegalCdrStateException stateEx) {
                if (src.available() >=
                CdrEncapsulation.CDR_ENCAPSULATION_PARAMETER_ID_ALIGNMENT) {
                    throw new IllegalCdrStateException(
                        "Error skipping sample! Remainder:" + src.available()
                        + "\nException caused by: " + stateEx.getMessage());
                }
            }
        }
        if (skip_encapsulation) {
            src.restoreAlignment(position);
        }
    }
    @Override
    public java.lang.Object serialized_sample_to_key(
        java.lang.Object endpoint_data,
        java.lang.Object sample,
        CdrInputStream src,
        boolean deserialize_encapsulation,
```

```
        boolean deserialize_key,
        java.lang.Object endpoint_plugin_qos)
    {
        int position = 0;
        int tmpPosition = 0, tmpSize = 0;
        long tmpLength = 0;
        CdrBuffer buffer = null;
        boolean inBaseClass_tmp = false;
        inBaseClass_tmp =  src.inBaseClass;
        src.inBaseClass = false;
        DefaultEndpointData epd = (DefaultEndpointData) endpoint_data;
        if (epd == null) {
            /* Coverity reports that epd store a object that might not
            be used later. It is true that it might not be used,
            depending on the type complexity. This is intentional
            because it doesn't affect generated code and fixing it
            would make generated code more difficult to maintain */
            /* coverity[defect] */
            epd = new DefaultEndpointData();
        }
        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();
            position = src.resetAlignment();
        }
        if (deserialize_key) {
            short encapsulation_id = src.getEncapsulationKind();
            boolean xcdr1 = (encapsulation_id <= CdrEncapsulation.CDR_ENCAPSULATION_ID_PL_CDR_LE)? true: false;
            if(!xcdr1){
                buffer = src.getBuffer();
            }
            HelloWorld typedDst = (HelloWorld) sample;
            deserialize_sample(
                epd, sample, src, false,
                true, endpoint_plugin_qos);
        }
        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }
        return sample;
    }
    // -----------------------------------------------------------------------
    // Callbacks
    // -----------------------------------------------------------------------
    @Override
    public Object on_participant_attached(java.lang.Object registration_data,
    TypeSupportParticipantInfo participant_info,
    boolean top_level_registration,
    java.lang.Object container_plugin_context,
    TypeCode type_code) {
        return super.on_participant_attached(
            registration_data, participant_info, top_level_registration,
            container_plugin_context, type_code);
    }
    @Override
    public void on_participant_detached(java.lang.Object participant_data) {
        super.on_participant_detached(participant_data);
    }
    @Override
    public java.lang.Object on_endpoint_attached(java.lang.Object participantData,
    TypeSupportEndpointInfo endpoint_info,
    boolean top_level_registration,
    java.lang.Object container_plugin_context) {
        return super.on_endpoint_attached(
            participantData,  endpoint_info,
            top_level_registration, container_plugin_context);
    }
    @Override
    public void on_endpoint_detached(java.lang.Object endpoint_data) {
        super.on_endpoint_detached(endpoint_data);
    }
    // -----------------------------------------------------------------------
    // Protected Methods
    // -----------------------------------------------------------------------
    @Override
    protected DataWriter create_datawriter(long native_writer,
    DataWriterListener listener,
    int mask) {
        return new HelloWorldDataWriter (native_writer, listener, mask, this);
    }
    @Override
    protected DataReader create_datareader(long native_reader,
```

```
    DataReaderListener listener,
    int mask) {
        return new HelloWorldDataReader(native_reader, listener, mask, this);
    }
    // --------------------------------------------------------------------
    // Constructor
    // --------------------------------------------------------------------
    protected HelloWorldTypeSupport() {
        /* If the user data type supports keys, then the second argument
        to the constructor below should be true.  Otherwise it should
        be false. */
        super(TYPE_NAME, false, HelloWorldTypeCode.VALUE, HelloWorld.class,TypeSupportType.TST_STRUCT,
       PLUGIN_VERSION);
    }
    protected HelloWorldTypeSupport (boolean enableKeySupport) {
        super(TYPE_NAME, enableKeySupport, HelloWorldTypeCode.VALUE,
       HelloWorld.class,TypeSupportType.TST_STRUCT, PLUGIN_VERSION);
    }
}
```

9.3.1.2 HelloWorldDataWriter.java

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

This file was generated from .idl
using RTI Code Generator (rtiddsgen) version 4.3.0.
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.
*/
import com.rti.dds.infrastructure.Time_t;
import com.rti.dds.infrastructure.WriteParams_t;
import com.rti.dds.infrastructure.InstanceHandle_t;
import com.rti.dds.publication.DataWriterImpl;
import com.rti.dds.publication.DataWriterListener;
import com.rti.dds.topic.TypeSupportImpl;
// ===========================================================================
public class HelloWorldDataWriter extends DataWriterImpl {
    // --------------------------------------------------------------------
    // Public Methods
    // --------------------------------------------------------------------
    public InstanceHandle_t register_instance(HelloWorld instance_data) {
        return register_instance_untyped(instance_data);
    }
    public InstanceHandle_t register_instance_w_timestamp(HelloWorld instance_data,
    Time_t source_timestamp) {
        return register_instance_w_timestamp_untyped(
            instance_data, source_timestamp);
    }
    public InstanceHandle_t register_instance_w_params(HelloWorld instance_data,
    WriteParams_t params) {
        return register_instance_w_params_untyped(
            instance_data, params);
    }
    public void unregister_instance(HelloWorld instance_data,
    InstanceHandle_t handle) {
        unregister_instance_untyped(instance_data, handle);
    }
    public void unregister_instance_w_timestamp(HelloWorld instance_data,
    InstanceHandle_t handle, Time_t source_timestamp) {
        unregister_instance_w_timestamp_untyped(
            instance_data, handle, source_timestamp);
    }
    public void unregister_instance_w_params(HelloWorld instance_data,
    WriteParams_t params) {
        unregister_instance_w_params_untyped(
            instance_data, params);
    }
    public void write(HelloWorld instance_data, InstanceHandle_t handle) {
        write_untyped(instance_data, handle);
    }
    public void write_w_timestamp(HelloWorld instance_data,
    InstanceHandle_t handle, Time_t source_timestamp) {
        write_w_timestamp_untyped(instance_data, handle, source_timestamp);
    }
    public void write_w_params(HelloWorld instance_data,
    WriteParams_t params) {
```

```
        write_w_params_untyped(instance_data, params);
    }
    public void dispose(HelloWorld instance_data, InstanceHandle_t instance_handle){
        dispose_untyped(instance_data, instance_handle);
    }
    public void dispose_w_timestamp(HelloWorld instance_data,
    InstanceHandle_t instance_handle, Time_t source_timestamp) {
        dispose_w_timestamp_untyped(
            instance_data, instance_handle, source_timestamp);
    }
    public void dispose_w_params(HelloWorld instance_data,
    WriteParams_t params) {
        dispose_w_params_untyped(instance_data, params);
    }
    public void get_key_value(HelloWorld key_holder, InstanceHandle_t handle) {
        get_key_value_untyped(key_holder, handle);
    }
    public InstanceHandle_t lookup_instance(HelloWorld key_holder) {
        return lookup_instance_untyped(key_holder);
    }
    // --------------------------------------------------------------------
    // Package Methods
    // --------------------------------------------------------------------
    // --- Constructors: -------------------------------------------------
    /*package*/ HelloWorldDataWriter(long native_writer, DataWriterListener listener,
    int mask, TypeSupportImpl type) {
        super(native_writer, listener, mask, type);
    }
}
```

9.3.1.3 HelloWorldDataReader.java

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

This file was generated from .idl
using RTI Code Generator (rtiddsgen) version 4.3.0.
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.
*/
import com.rti.dds.infrastructure.InstanceHandle_t;
import com.rti.dds.subscription.DataReaderImpl;
import com.rti.dds.subscription.DataReaderListener;
import com.rti.dds.subscription.ReadCondition;
import com.rti.dds.subscription.SampleInfo;
import com.rti.dds.subscription.SampleInfoSeq;
import com.rti.dds.topic.TypeSupportImpl;
// ===========================================================================
public class HelloWorldDataReader extends DataReaderImpl {
    // --------------------------------------------------------------------
    // Public Methods
    // --------------------------------------------------------------------
    public void read(HelloWorldSeq received_data, SampleInfoSeq info_seq,
    int max_samples,
    int sample_states, int view_states, int instance_states) {
        read_untyped(received_data, info_seq, max_samples, sample_states,
        view_states, instance_states);
    }
    public void take(HelloWorldSeq received_data, SampleInfoSeq info_seq,
    int max_samples,
    int sample_states, int view_states, int instance_states) {
        take_untyped(received_data, info_seq, max_samples, sample_states,
        view_states, instance_states);
    }
    public void read_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq,
    int max_samples,
    ReadCondition condition) {
        read_w_condition_untyped(received_data, info_seq, max_samples,
        condition);
    }
    public void take_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq,
    int max_samples,
    ReadCondition condition) {
        take_w_condition_untyped(received_data, info_seq, max_samples,
        condition);
```

```
    }
    public void read_next_sample(HelloWorld received_data, SampleInfo sample_info) {
        read_next_sample_untyped(received_data, sample_info);
    }
    public void take_next_sample(HelloWorld received_data, SampleInfo sample_info) {
        take_next_sample_untyped(received_data, sample_info);
    }
    public void read_instance(HelloWorldSeq received_data, SampleInfoSeq info_seq,
    int max_samples, InstanceHandle_t a_handle, int sample_states,
    int view_states, int instance_states) {
        read_instance_untyped(received_data, info_seq, max_samples, a_handle,
        sample_states, view_states, instance_states);
    }
    public void take_instance(HelloWorldSeq received_data, SampleInfoSeq info_seq,
    int max_samples, InstanceHandle_t a_handle, int sample_states,
    int view_states, int instance_states) {
        take_instance_untyped(received_data, info_seq, max_samples, a_handle,
        sample_states, view_states, instance_states);
    }
    public void read_instance_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, ReadCondition condition) {
        read_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
    }
    public void take_instance_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, ReadCondition condition) {
        take_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
    }
    public void read_next_instance(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, int sample_states, int view_states,
    int instance_states) {
        read_next_instance_untyped(received_data, info_seq, max_samples,
        a_handle, sample_states, view_states, instance_states);
    }
    public void take_next_instance(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, int sample_states, int view_states,
    int instance_states) {
        take_next_instance_untyped(received_data, info_seq, max_samples,
        a_handle, sample_states, view_states, instance_states);
    }
    public void read_next_instance_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, ReadCondition condition) {
        read_next_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
    }
    public void take_next_instance_w_condition(HelloWorldSeq received_data,
    SampleInfoSeq info_seq, int max_samples,
    InstanceHandle_t a_handle, ReadCondition condition) {
        take_next_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
    }
    public void return_loan(HelloWorldSeq received_data, SampleInfoSeq info_seq) {
        return_loan_untyped(received_data, info_seq);
    }
    public void get_key_value(HelloWorld key_holder, InstanceHandle_t handle){
        get_key_value_untyped(key_holder, handle);
    }
    public InstanceHandle_t lookup_instance(HelloWorld key_holder) {
        return lookup_instance_untyped(key_holder);
    }
    // -----------------------------------------------------------------------
    // Package Methods
    // -----------------------------------------------------------------------
    // --- Constructors: -----------------------------------------------------
    /*package*/  HelloWorldDataReader (long native_reader, DataReaderListener listener,
    int mask, TypeSupportImpl data_type) {
        super(native_reader, listener, mask, data_type);
    }
}
```

9.4 HelloWorldPublisher.java

9.4.1 RTI Connext Publication Example

The publication example generated by rtiddsgen (see the `Code Generator User's Manual` for more information). The example has been modified slightly to update the sample value.

9.4.1.1 HelloWorldPublisher.java

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
import java.util.Objects;
import java.lang.reflect.*;
import com.rti.dds.domain.DomainParticipant;
import com.rti.dds.domain.DomainParticipantFactory;
import com.rti.dds.infrastructure.InstanceHandle_t;
import com.rti.dds.infrastructure.StatusKind;
import com.rti.dds.publication.Publisher;
import com.rti.dds.topic.Topic;
public class HelloWorldPublisher extends Application implements AutoCloseable {
    // Usually one per application
    private DomainParticipant participant = null;
    private static final String TAG = "RTIConnextLog";
    private void runApplication() {
        // Start communicating in a domain
        participant = Objects.requireNonNull(
            DomainParticipantFactory.get_instance().create_participant(
                getDomainId(),
                DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // A Publisher allows an application to create one or more DataWriters
        Publisher publisher = Objects.requireNonNull(
            participant.create_publisher(
                DomainParticipant.PUBLISHER_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // Register the datatype to use when creating the Topic
        String typeName = HelloWorldTypeSupport.get_type_name();
        HelloWorldTypeSupport.register_type(participant, typeName);
        // Create a Topic with a name and a datatype
        Topic topic = Objects.requireNonNull(
            participant.create_topic(
                "Example HelloWorld",
                typeName,
                DomainParticipant.TOPIC_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // This DataWriter writes data on "Example HelloWorld" Topic
        HelloWorldDataWriter writer = (HelloWorldDataWriter) Objects.requireNonNull(
            publisher.create_datawriter(
                topic,
                Publisher.DATAWRITER_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // Create data sample for writing
        HelloWorld data = new HelloWorld();
        for (int samplesWritten = 0; !isShutdownRequested()
        && samplesWritten < getMaxSampleCount(); samplesWritten++) {
            // Modify the data to be written here
            System.out.println("Writing HelloWorld, count " + samplesWritten);
            writer.write(data, InstanceHandle_t.HANDLE_NIL);
            try {
                final long sendPeriodMillis = 1000; // 1 second
```

```
                Thread.sleep(sendPeriodMillis);
            } catch (InterruptedException ix) {
                System.err.println("INTERRUPTED");
                break;
            }
        }
    }
    @Override
    public void close() {
        // Delete all entities (DataWriter, Topic, Publisher, DomainParticipant)
        if (participant != null) {
            participant.delete_contained_entities();
            DomainParticipantFactory.get_instance()
            .delete_participant(participant);
        }
    }
    public static void main(String[] args) {
        // Create example and run: Uses try-with-resources,
        // publisherApplication.close() automatically called
        try (HelloWorldPublisher publisherApplication = new HelloWorldPublisher()) {
            publisherApplication.parseArguments(args);
            publisherApplication.addShutdownHook();
            publisherApplication.runApplication();
        }
        // Releases the memory used by the participant factory. Optional at application exit.
        DomainParticipantFactory.finalize_instance();
    }
}
```

9.5 HelloWorldSubscriber.java

9.5.1 RTI Connext Subscription Example

The unmodified subscription example generated by rtiddsgen (see the `Code Generator User's Manual` for more information).

9.5.1.1 HelloWorldSubscriber.java

```
/*
* (c) Copyright, Real-Time Innovations, 2020.  All rights reserved.
* RTI grants Licensee a license to use, modify, compile, and create derivative
* works of the software solely for use with RTI Connext DDS. Licensee may
* redistribute copies of the software provided that all such copies are subject
* to this license. The software is provided "as is", with no warranty of any
* type, including any warranty for fitness for any purpose. RTI is under no
* obligation to maintain or support the software. RTI shall not be liable for
* any incidental or consequential damages arising out of the use or inability
* to use the software.
*/
import java.util.Objects;
import java.lang.reflect.*;
import com.rti.dds.domain.DomainParticipant;
import com.rti.dds.domain.DomainParticipantFactory;
import com.rti.dds.infrastructure.ConditionSeq;
import com.rti.dds.infrastructure.Duration_t;
import com.rti.dds.infrastructure.RETCODE_NO_DATA;
import com.rti.dds.infrastructure.RETCODE_TIMEOUT;
import com.rti.dds.infrastructure.ResourceLimitsQosPolicy;
import com.rti.dds.infrastructure.StatusKind;
import com.rti.dds.infrastructure.WaitSet;
import com.rti.dds.subscription.InstanceStateKind;
import com.rti.dds.subscription.ReadCondition;
import com.rti.dds.subscription.SampleInfo;
import com.rti.dds.subscription.SampleInfoSeq;
import com.rti.dds.subscription.SampleStateKind;
import com.rti.dds.subscription.Subscriber;
import com.rti.dds.subscription.ViewStateKind;
import com.rti.dds.topic.Topic;
public class HelloWorldSubscriber extends Application implements AutoCloseable {
    private DomainParticipant participant = null; // Usually one per application
```

```
    private static final String TAG = "RTIConnextLog";
    private HelloWorldDataReader reader = null;
    private final HelloWorldSeq dataSeq = new HelloWorldSeq();
    private final SampleInfoSeq infoSeq = new SampleInfoSeq();
    private int processData() {
        int samplesRead = 0;
        try {
            // Take available data from DataReader's queue
            reader.take(dataSeq, infoSeq,
            ResourceLimitsQosPolicy.LENGTH_UNLIMITED,
            SampleStateKind.ANY_SAMPLE_STATE,
            ViewStateKind.ANY_VIEW_STATE,
            InstanceStateKind.ANY_INSTANCE_STATE);
            for (int i = 0; i < dataSeq.size(); ++i) {
                SampleInfo info = infoSeq.get(i);
                if (info.valid_data) {
                    System.out.println("Received" + dataSeq.get(i));
                }
                samplesRead++;
            }
        } catch (RETCODE_NO_DATA noData) {
            // No data to process, not a problem
        } finally {
            // Data loaned from Connext for performance. Return loan when done.
            reader.return_loan(dataSeq, infoSeq);
        }
        return samplesRead;
    }
    private void runApplication() {
        // Start communicating in a domain
        participant = Objects.requireNonNull(
            DomainParticipantFactory.get_instance().create_participant(
                getDomainId(),
                DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // A Subscriber allows an application to create one or more DataReaders
        Subscriber subscriber = Objects.requireNonNull(
            participant.create_subscriber(
                DomainParticipant.SUBSCRIBER_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // Register the datatype to use when creating the Topic
        String typeName = HelloWorldTypeSupport.get_type_name();
        HelloWorldTypeSupport.register_type(participant, typeName);
        // Create a Topic with a name and a datatype
        Topic topic = Objects.requireNonNull(
            participant.create_topic(
                "Example HelloWorld",
                typeName,
                DomainParticipant.TOPIC_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // This DataReader reads data on "Example HelloWorld" Topic
        reader = (HelloWorldDataReader) Objects.requireNonNull(
            subscriber.create_datareader(
                topic,
                Subscriber.DATAREADER_QOS_DEFAULT,
                null, // listener
                StatusKind.STATUS_MASK_NONE));
        // Create ReadCondition that triggers when data in reader's queue
        ReadCondition condition = reader.create_readcondition(
            SampleStateKind.ANY_SAMPLE_STATE,
            ViewStateKind.ANY_VIEW_STATE,
            InstanceStateKind.ANY_INSTANCE_STATE);
        // WaitSet will be woken when the attached condition is triggered, or timeout
        WaitSet waitset = new WaitSet();
        waitset.attach_condition(condition);
        final Duration_t waitTimeout = new Duration_t(1, 0);
        int samplesRead = 0;
        ConditionSeq activeConditions = new ConditionSeq();
        // Main loop. Wait for data to arrive and process when it arrives
        while (!isShutdownRequested() && samplesRead < getMaxSampleCount()) {
            try {
                // Wait fills in activeConditions or times out
                waitset.wait(activeConditions, waitTimeout);
                // Read condition triggered, process data
                samplesRead += processData();
            } catch (RETCODE_TIMEOUT timeout) {
                // No data received, not a problem
                System.out.printf("No data after %d seconds.%n", waitTimeout.sec);
```

```
            }
        }
    }
    @Override
    public void close() {
        // Delete all entities (DataReader, Topic, Subscriber, DomainParticipant)
        if (participant != null) {
            participant.delete_contained_entities();
            DomainParticipantFactory.get_instance()
            .delete_participant(participant);
        }
    }
    public static void main(String[] args) {
        // Create example and run: Uses try-with-resources,
        // subscriberApplication.close() automatically called
        try (HelloWorldSubscriber subscriberApplication = new HelloWorldSubscriber()) {
            subscriberApplication.parseArguments(args);
            subscriberApplication.addShutdownHook();
            subscriberApplication.runApplication();
        }
        // Releases the memory used by the participant factory. Optional at application exit.
        DomainParticipantFactory.finalize_instance();
    }
}
```

Index