RTI Connext Java API

Version 4.5f

Generated by Doxygen 1.5.5

Sat Mar 17 21:18:59 2012

Contents

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. The most recent publication of this specification can be found in the Catalog of OMG Specifications under "Middleware Specifications".

1.1 Feedback and Support for this Release.

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

By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send e-mail to support@rti.com. We can only guarantee a response for customers with a current maintenance contract or subscription. To purchase a maintenance contract or subscription, contact your local RTI representative

(see `http://www.rti.com/company/contact.html`), send an email request to `sales@rti.com`, or call +1 (408) 990-7400.

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

1.2 Available Documentation.

The documentation for this release is provided in two forms: the HTML API reference documentation and PDF documents. If you are new to RTI Connext, the **Documentation Roadmap** (p. 269) will provide direction on how to learn about this product.

1.2.1 The PDF documents 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.
- `Getting Started Guide.` 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. Developers should read this document first.
- `Getting Started Guide, Database Addendum.` Additional installation and setup information for database usage.
- `Getting Started Guide, Embedded Systems Addendum.` Additional installation and setup information for embedded systems.
- `User's Manual.` Introduction to RTI Connext, product tour and conceptual presentation of the functionality of RTI Connext.
- `Platform Notes.` Specific details, such as compilation setting and libraries, related to building and using RTI Connext on the various supported platforms.
- `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, an experimental feature in this release.

- `C API Reference Manual.` A consolidated PDF version of the HTML C API reference documentation.
- `C++ API Reference Manual.` A consolidated PDF version of the HTML C++ API reference documentation.
- `Java API Reference Manual.` A consolidate PDF version of the HTML Java API reference documentation.
- `.NET API Reference Manual.` A consolidated PDF version of the HTML .Net API reference documentation.

1.2.2 The HTML API Reference documentation contains:

- **DDS API Reference** (p. 272) - The DDS API reference.
- **RTI Connext API Reference** (p. 286) - RTI Connext API's independent of the DDS standard.
- **Programming How-To's** (p. 287) - Describes and shows the common tasks done using the API.
- **Programming Tools** (p. 289) - RTI Connext helper tools.

The HTML API Reference 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".

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Namespace Index

3.1 Package List

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

Chapter 4

Class 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 ASYNCHRONOUS_PUBLISHER

<<***eXtension***>> *(p. 270)* Specifies the asynchronous publishing settings of the **com.rti.dds.publication.Publisher** (p. 1277) instances.

Classes

ˆ class **AsynchronousPublisherQosPolicy**

Configures the mechanism that sends user data in an external middleware thread.

Variables

ˆ static final QosPolicyId_t **ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID**

<<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy*** *(p. 387)*

6.1.1 Detailed Description

<<***eXtension***>> *(p. 270)* Specifies the asynchronous publishing settings of the **com.rti.dds.publication.Publisher** (p. 1277) instances.

6.1.2 Variable Documentation

6.1.2.1 final QosPolicyId_t ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.AsynchronousPublisherQosP**
(p. 387)

6.2 AVAILABILITY

<<*eXtension*>> *(*p. *270)* Configures the availability of data.

Classes

- ˆ class **AvailabilityQosPolicy**

  Configures the availability of data.

- ˆ class **EndpointGroup_t**

  Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

- ˆ class **EndpointGroupSeq**

  A sequence of ***com.rti.dds.infrastructure.EndpointGroup_t*** *(*p. *909).*

Variables

- ˆ static final QosPolicyId_t **AVAILABILITY_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.AvailabilityQosPolicy*** *(*p. *392)*

6.2.1 Detailed Description

<<*eXtension*>> *(*p. *270)* Configures the availability of data.

6.2.2 Variable Documentation

6.2.2.1 final QosPolicyId_t AVAILABILITY_QOS_POLICY_ID

`[static, inherited]`

<<*eXtension*>> *(*p. *270)* Identifier for **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 392)

6.3 BATCH

<<*eXtension*>> *(p. 270)* Batch QoS policy used to enable batching in **com.rti.dds.publication.DataWriter** (p. 538) 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. 270) *Identifier for* ***com.rti.dds.infrastructure.BatchQosPolicy*** *(p. 401)*

6.3.1 Detailed Description

<<*eXtension*>> *(p. 270)* Batch QoS policy used to enable batching in **com.rti.dds.publication.DataWriter** (p. 538) instances.

6.3.2 Variable Documentation

6.3.2.1 final QosPolicyId_t BATCH_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.BatchQosPolicy** (p. 401)

6.4 Conditions and WaitSets

com.rti.dds.infrastructure.Condition (p. 451) and **com.rti.dds.infrastructure.WaitSet** (p. 1695) and related items.

Classes

- interface **Condition**

  <<**interface**>> (p. 271) *Root class for all the conditions that may be attached to a* ***com.rti.dds.infrastructure.WaitSet*** *(p. 1695).*

- class **ConditionSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) *<* ***com.rti.dds.infrastructure.Condition*** *(p. 451) >.*

- class **GuardCondition**

  <<**interface**>> (p. 271) *A specific* ***com.rti.dds.infrastructure.Condition*** *(p. 451) whose* `trigger_value` *is completely under the control of the application.*

- interface **StatusCondition**

  <<**interface**>> (p. 271) *A specific* ***com.rti.dds.infrastructure.Condition*** *(p. 451) that is associated with each* ***com.rti.dds.infrastructure.Entity*** *(p. 912).*

- class **WaitSet**

  <<**interface**>> (p. 271) *Allows an application to wait until one or more of the attached* ***com.rti.dds.infrastructure.Condition*** *(p. 451) objects has a* `trigger_value` *of true or else until the timeout expires.*

- class **WaitSetProperty_t**

  <<**eXtension**>> (p. 270) *Specifies the* ***com.rti.dds.infrastructure.WaitSet*** *(p. 1695) behavior for multiple trigger events.*

6.4.1 Detailed Description

com.rti.dds.infrastructure.Condition (p. 451) and **com.rti.dds.infrastructure.WaitSet** (p. 1695) and related items.

6.5 DATABASE

*<<**eXtension**>> (p. 270)* 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. 270) *Identifier for **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 468)*

6.5.1 Detailed Description

*<<**eXtension**>> (p. 270)* Various threads and resource limits settings used by RTI Connext to control its internal database.

6.5.2 Variable Documentation

6.5.2.1 final QosPolicyId_t DATABASE_QOS_POLICY_ID [static, inherited]

*<<**eXtension**>> (p. 270)* Identifier for **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 468)

6.6 DATA_READER_PROTOCOL

<<*eXtension*>> *(p. 270)* Specifies the DataReader-specific protocol QoS.

Classes

- class **DataReaderProtocolQosPolicy**

  Along with ***com.rti.dds.infrastructure.WireProtocolQosPolicy*** *(p. 1709) and* ***com.rti.dds.infrastructure.DataWriterProtocolQosPolicy*** *(p. 571), this QoS policy configures the DDS on-the-network protocol (RTPS).*

Variables

- static final QosPolicyId_t **DATAREADERPROTOCOL_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataReaderProtocolQosPolicy*** *(p. 504)*

6.6.1 Detailed Description

<<*eXtension*>> *(p. 270)* Specifies the DataReader-specific protocol QoS.

6.6.2 Variable Documentation

6.6.2.1 final QosPolicyId_t DATAREADERPROTOCOL_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504)

6.7 DATA_READER_RESOURCE_LIMITS

*<<**eXtension**>>* *(p. 270)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Classes

- class **DataReaderResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 473) allocates and uses physical memory for internal resources.*

Variables

- static final int **AUTO_MAX_TOTAL_INSTANCES**

  <<**eXtension**>> (p. 270) *This value is used to make **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_total_instances** (p. 533) equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360).*

- static final QosPolicyId_t **DATAREADERRESOURCELIMITS_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524)*

6.7.1 Detailed Description

*<<**eXtension**>>* *(p. 270)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

6.7.2 Variable Documentation

6.7.2.1 final int AUTO_MAX_TOTAL_INSTANCES [static, inherited]

*<<**eXtension**>>* *(p. 270)* This value is used to make **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_total_instances** (p. 533) equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360).

6.7.2.2 final QosPolicyId_t DATAREADERRESOURCELIMITS_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524)

6.8 DATA_WRITER_PROTOCOL

<<*eXtension*>> *(p. 270)* Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709) and **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504), 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. 538) instances.*

Variables

- static final QosPolicyId_t **DATAWRITERPROTOCOL_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataWriterProtocolQosPolicy*** *(p. 571)*

6.8.1 Detailed Description

<<*eXtension*>> *(p. 270)* Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709) and **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504), this QoS policy configures the DDS on-the-network protocol (RTPS).

6.8.2 Variable Documentation

6.8.2.1 final QosPolicyId_t DATAWRITERPROTOCOL_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DataWriterProtocolQosPolic** (p. 571)

6.9 DATA_WRITER_RESOURCE_LIMITS

<<*eXtension*>> *(p. 270)* Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) allocates and uses physical memory for internal resources.*

Variables

- static final QosPolicyId_t **DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy** (p. 598)*

6.9.1 Detailed Description

<<*eXtension*>> *(p. 270)* Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 538) allocates and uses physical memory for internal resources.

6.9.2 Variable Documentation

6.9.2.1 final QosPolicyId_t DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy** (p. 598)

6.10 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. 604).*

6.10.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

6.10.2 Variable Documentation

6.10.2.1 final QosPolicyId_t DEADLINE_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604).

6.11 DESTINATION_ORDER

Controls the criteria used to determine the logical order among changes made by **com.rti.dds.publication.Publisher** (p. 1277) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1545) and key).

Classes

- class **DestinationOrderQosPolicy**

  *Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 538) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1545) and key).*

- class **DestinationOrderQosPolicyKind**

  Kinds of destination order.

Variables

- static final QosPolicyId_t **DESTINATIONORDER_QOS_POLICY_-ID**

  *Identifier for **com.rti.dds.infrastructure.DestinationOrderQosPolicy** (p. 607).*

6.11.1 Detailed Description

Controls the criteria used to determine the logical order among changes made by **com.rti.dds.publication.Publisher** (p. 1277) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1545) and key).

6.11.2 Variable Documentation

6.11.2.1 final QosPolicyId_t DESTINATIONORDER_QOS_-POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.DestinationOrderQosPolicy** (p. 607).

6.12 DISCOVERY_CONFIG

*<<**eXtension**>>* *(p. 270)* Specifies the discovery configuration QoS.

Classes

- class **BuiltinTopicReaderResourceLimits_t**

  *Built-in **topic** (p. 350) reader's resource limits.*

- 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 **SDP**

  Built-in discovery plugins that can be used.

- static final int **MASK_NONE** = 0

  A bit-mask (list) of built-in discovery plugins.

- static final int **MASK_ALL** = 0xefff

  A bit-mask (list) of built-in discovery plugins.

- static final QosPolicyId_t **DISCOVERYCONFIG_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy** (p. 615)*

6.12.1 Detailed Description

*<<**eXtension**>>* *(p. 270)* Specifies the discovery configuration QoS.

6.12.2 Variable Documentation

6.12.2.1 final int SDP [static, inherited]

Built-in discovery plugins that can be used.

See also:

com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKindMask .**SDP** (p. 53)

6.12.2.2 final int MASK_NONE = 0 [static, inherited]

A bit-mask (list) of built-in discovery plugins.

The bit-mask is an efficient and compact representation of a fixed-length list of **com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind** (p. 614) values.

QoS:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615) .MASK_NONE

6.12.2.3 final int MASK_ALL = 0xefff [static, inherited]

A bit-mask (list) of built-in discovery plugins.

The bit-mask is an efficient and compact representation of a fixed-length list of **com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKind** (p. 614) values.

QoS:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615) .MASK_ALL

6.12.2.4 final QosPolicyId_t DISCOVERYCONFIG_QOS_POLICY_ID [static, inherited]

<<***eXtension***>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy** (p. 615)

6.13 DISCOVERY

<<***eXtension***>> *(p. 270)* 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. 626) and* ***com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses*** *(p. 625) contained in the* ***com.rti.dds.domain.DomainParticipantQos.discovery*** *(p. 739) qos policy.*

Classes

- class **DiscoveryQosPolicy**

  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. 270) *Identifier for* ***com.rti.dds.infrastructure.DiscoveryQosPolicy*** *(p. 624)*

6.13.1 Detailed Description

<<***eXtension***>> *(p. 270)* Specifies the attributes required to discover participants in the domain.

6.13.2 Variable Documentation

6.13.2.1 final QosPolicyId_t DISCOVERY_QOS_POLICY_ID [static, inherited]

<<***eXtension***>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 624)

6.14 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 626) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625) contained in the **com.rti.dds.domain.DomainParticipantQos.discovery** (p. 739) qos policy.

The default value of the **com.rti.dds.domain.DomainParticipantQos** (p. 736) is obtained by calling **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos()** (p. 716).

NDDS_DISCOVERY_PEERS specifies the default value of the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 626) and **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625) 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. 716) has never been called or was called using DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625) 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. 625), 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. 625).

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. 56)) or in a file. These formats are described below.

6.14.1 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. 626) field and the **com.rti.dds.domain.DomainParticipant.add_peer()** (p. 692) operation.

The anatomy of a `peer` descriptor is illustrated below using a special "StarFabric" transport example.

A peer descriptor consists of:

optional **Participant ID**. 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.

ˆ **Locator**. See **Locator Format** (p. 56).

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).

6.14.1.1 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. 368)) 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. 57).

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. 912)) may be used to parse the address string.

6.14.1.2 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.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625), and **com.rti.dds.infrastructure.TransportMulticastSettings_t.receive_address** (p. 1595) 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. 371)).

optional **Transport Address**. A string that is passed to the transport for processing. The transport maps this string into Transport.Property_t.address_bit_count 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.

The bits resulting from the transport address string are prepended with the network address. The least significant Transport.Property_t.address_bit_count bits of the network address are ignored (**Transport Network Address** (p. 371)).

If the separator is omitted and the string is not a valid IPv4 or IPv6 address, it is treated as a transport address with an implicit network address (of the transport plugin).

6.14.2 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 FAA0::0#localhost (could be a UDPv4 transport plugin registered at network address of FAA0::0) (IPv6)

- FAA0::0#localhost

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 all unicast addresses accessed via the "udpv4" (UDPv4) transport plugin(s)

- udpv4://

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)

- shmem://

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

6.14.3 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 - Default 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 RTI Connext 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)
```

6.14.4 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.

6.14.5 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 RTI Connext applications anywhere on the local network to
;; discover each other over UDPv4.

builtin.udpv4://239.255.0.1  ; RTI Connext's default discovery multicast address


;; Unicast - UDPv4 (on all platforms)
;;
;; This allows two RTI Connext 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 RTI Connext 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://
```

6.14.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - ShmemTransport
- `udpv4` - UDPv4Transport
- `udpv6` - UDPv6Transport

These may be used as the transport names in the **Locator Format** (p. 56).

6.14.7 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. 1580) of the **com.rti.dds.domain.DomainParticipant** (p. 629). 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 document.)

See also:

com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses (p. 625)
com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 626)
com.rti.dds.domain.DomainParticipant.add_peer() (p. 692)
DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT
com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos() (p. 716)
Transport Aliases (p. 368)
Transport Network Address (p. 371)

6.15 DOMAIN_PARTICIPANT_RESOURCE_-LIMITS

<<*eXtension*>> *(p. 270)* Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Classes

- class **AllocationSettings_t**

  Resource allocation settings.

- class **DomainParticipantResourceLimitsQosPolicy**

  *Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

Variables

- static final QosPolicyId_t **DOMAINPARTICIPANTRESOURCE-LIMITS_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 741)*

6.15.1 Detailed Description

<<*eXtension*>> *(p. 270)* Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

6.15.2 Variable Documentation

6.15.2.1 final QosPolicyId_t DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.DomainParticipantResourceL** (p. 741)

6.16 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. 473) 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. 473) entities that join the network later.*

- class **DurabilityQosPolicyKind**

  Kinds of durability.

Variables

- static final QosPolicyId_t **DURABILITY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 765).*

6.16.1 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. 473) entities that join the network later.

6.16.2 Variable Documentation

6.16.2.1 final QosPolicyId_t DURABILITY_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 765).

6.17 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. 765) setting of DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or 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. 765) setting of* ***DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS*** *(p. 772) or* ***DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS*** *(p. 771).*

Variables

- static final QosPolicyId_t **DURABILITY_SERVICE_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.DurabilityServiceQosPolicy*** *(p. 773).*

6.17.1 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. 765) setting of DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.

6.17.2 Variable Documentation

6.17.2.1 final QosPolicyId_t DURABILITY_SERVICE_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.DurabilityServiceQosPolicy** (p. 773).

6.18 Time Support

Time and duration types and defines.

Classes

- class **Duration_t**

  Type for duration *representation.*

- class **Time_t**

  Type for time *representation.*

6.18.1 Detailed Description

Time and duration types and defines.

6.19 Entity Support

com.rti.dds.infrastructure.Entity (p. 912), **com.rti.dds.infrastructure.Listener** (p. 1154) and related items.

Classes

- interface **DomainEntity**

  <<**interface**>> (p. 271) *Abstract base class for all DDS entities except for the **com.rti.dds.domain.DomainParticipant** (p. 629).*

- interface **Entity**

  <<**interface**>> (p. 271) *Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.*

- interface **Listener**

  <<**interface**>> (p. 271) *Abstract base class for all **Listener** (p. 1154) interfaces.*

6.19.1 Detailed Description

com.rti.dds.infrastructure.Entity (p. 912), **com.rti.dds.infrastructure.Listener** (p. 1154) and related items.

com.rti.dds.infrastructure.Entity (p. 912) subtypes are created and destroyed by factory objects. With the exception of **com.rti.dds.domain.DomainParticipant** (p. 629), whose factory is **com.rti.dds.domain.DomainParticipantFactory** (p. 708), all **com.rti.dds.infrastructure.Entity** (p. 912) factory objects are themselves **com.rti.dds.infrastructure.Entity** (p. 912) subtypes as well.

Important: all **com.rti.dds.infrastructure.Entity** (p. 912) delete operations are inherently thread-unsafe. The user must take extreme care that a given **com.rti.dds.infrastructure.Entity** (p. 912) 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. 912) is undefined.

6.20 ENTITY_FACTORY

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) types.

Classes

- class **EntityFactoryQosPolicy**

  *A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) types.*

Variables

- static final QosPolicyId_t **ENTITYFACTORY_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 919).*

6.20.1 Detailed Description

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) types.

6.20.2 Variable Documentation

6.20.2.1 final QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 919).

6.21 ENTITY_NAME

<<*eXtension*>> *(p. 270)* Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 629). This name will be 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. 629), **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473). These names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

6.21.1 Detailed Description

<<*eXtension*>> *(p. 270)* Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 629). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

6.22 EVENT

<<*eXtension*>> *(p. 270)* 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. 270) *Identifier for* ***com.rti.dds.infrastructure.EventQosPolicy*** *(p. 930)*

6.22.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures the internal thread in a DomainParticipant that handles timed events.

6.22.2 Variable Documentation

6.22.2.1 final QosPolicyId_t EVENT_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.EventQosPolicy** (p. 930)

6.23 EXCLUSIVE_AREA

<<*eXtension*>> *(p. 270)* Configures multi-thread concurrency and deadlock prevention capabilities.

Classes

- class **ExclusiveAreaQosPolicy**

  Configures multi-thread concurrency and deadlock prevention capabilities.

Variables

- static final QosPolicyId_t **EXCLUSIVEAREA_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.ExclusiveAreaQosPolicy*** *(p. 933)*

6.23.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures multi-thread concurrency and deadlock prevention capabilities.

6.23.2 Variable Documentation

6.23.2.1 final QosPolicyId_t EXCLUSIVEAREA_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy** (p. 933)

6.24 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Classes

- class **GroupDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.*

Variables

- static final QosPolicyId_t **GROUPDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1064).*

6.24.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

6.24.2 Variable Documentation

6.24.2.1 final QosPolicyId_t GROUPDATA_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1064).

6.25 GUID Support

<<*eXtension*>> *(*p. *270)* GUID type and defines.

Classes

- ˆ class **GUID_t**

  Type for GUID *(Global Unique Identifier) representation.*

6.25.1 Detailed Description

<<*eXtension*>> *(*p. *270)* GUID type and defines.

6.26 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.

- class **RefilterQosPolicyKind**

  <<**eXtension**>> (p. 270) *Kinds of Refiltering*

Variables

- static final QosPolicyId_t **HISTORY_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.HistoryQosPolicy*** *(p. 1071).*

6.26.1 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.

6.26.2 Variable Documentation

6.26.2.1 final QosPolicyId_t HISTORY_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071).

6.27 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. 1148).*

6.27.1 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.

6.27.2 Variable Documentation

6.27.2.1 final QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1148).

6.28 LIFESPAN

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 538) is considered valid.

Classes

- class **LifespanQosPolicy**

  *Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 538) is considered valid.*

Variables

- static final QosPolicyId_t **LIFESPAN_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1152).*

6.28.1 Detailed Description

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 538) is considered valid.

6.28.2 Variable Documentation

6.28.2.1 final QosPolicyId_t LIFESPAN_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.LifespanQosPolicy** (p. 1152).

6.29 LIVELINESS

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 473) entities to detect when **com.rti.dds.publication.DataWriter** (p. 538) entities become disconnected or "dead.".

Classes

- class **LivelinessQosPolicy**

  *Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 473) entities to detect when **com.rti.dds.publication.DataWriter** (p. 538) 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. 1164).*

6.29.1 Detailed Description

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 473) entities to detect when **com.rti.dds.publication.DataWriter** (p. 538) entities become disconnected or "dead.".

6.29.2 Variable Documentation

6.29.2.1 final QosPolicyId_t LIVELINESS_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164).

6.30 LOCATORFILTER

<<***eXtension***>> *(p. 270)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of builtin.PublicationBuiltinTopicData.

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 builtin.PublicationBuiltinTopicData.

- class **LocatorFilterSeq**

  Declares IDL `sequence<` ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1178)* `>`.

Variables

- static final QosPolicyId_t **LOCATORFILTER_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.LocatorFilterQosPolicy*** *(p. 1181)*

6.30.1 Detailed Description

<<***eXtension***>> *(p. 270)* The QoS policy used to report the configuration of a MultiChannel DataWriter as part of builtin.PublicationBuiltinTopicData.

6.30.2 Variable Documentation

6.30.2.1 final QosPolicyId_t LOCATORFILTER_QOS_POLICY_ID

`[static, inherited]`

<<***eXtension***>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.LocatorFilterQosPolicy** (p. 1181)

6.31 LOGGING

<<*eXtension*>> *(p. 270)* 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. 270) *Identifier for* ***com.rti.dds.infrastructure.LoggingQosPolicy*** *(p. 1190)*

6.31.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures the RTI Connext logging facility.

6.31.2 Variable Documentation

6.31.2.1 final QosPolicyId_t LOGGING_QOS_POLICY_ID

`[static, inherited]`

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.LoggingQosPolicy** (p. 1190)

6.32 MULTICHANNEL

<<*eXtension*>> *(p. 270)* 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. 441)* `>`.

- 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. 270) *Identifier for* ***com.rti.dds.infrastructure.MultiChannelQosPolicy*** *(p. 1205)*

6.32.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

6.32.2 Variable Documentation

6.32.2.1 final QosPolicyId_t MULTICHANNEL_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205)

6.33 Object Support

<<*eXtension*>> *(p. 270)* Object related items.

Classes

- interface **Copyable**

  <<**eXtension**>> (p. 270) <<**interface**>> (p. 271) *Interface for all the user-defined data type classes that support copy.*

- class **ObjectHolder**

  <<**eXtension**>> (p. 270) *Holder of object instance*

6.33.1 Detailed Description

<<*eXtension*>> *(p. 270)* 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.dds.topic.example.FooDataReader.take_next_sample or com.rti.dds.topic.example.FooDataReader.read_next_sample.

6.34 OWNERSHIP

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 538) (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. 538) (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. 1216).*

6.34.1 Detailed Description

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 538) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

6.34.2 Variable Documentation

6.34.2.1 final QosPolicyId_t OWNERSHIP_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.OwnershipQosPolicy** (p. 1216).

6.35 OWNERSHIP_STRENGTH

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 538) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1545) + key).

Classes

- class **OwnershipStrengthQosPolicy**

  *Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 538) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1545) + key).*

Variables

- static final QosPolicyId_t **OWNERSHIPSTRENGTH_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1225).*

6.35.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 538) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1545) + key).

6.35.2 Variable Documentation

6.35.2.1 final QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1225).

6.36 PARTITION

Set of strings that introduces a logical partition among the topics visible by a **com.rti.dds.publication.Publisher** (p. 1277) and a **com.rti.dds.subscription.Subscriber** (p. 1478).

Classes

- class **PartitionQosPolicy**

  *Set of strings that introduces a logical partition among the topics visible by a **com.rti.dds.publication.Publisher** (p. 1277) and a **com.rti.dds.subscription.Subscriber** (p. 1478).*

Variables

- static final QosPolicyId_t **PARTITION_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1233).*

6.36.1 Detailed Description

Set of strings that introduces a logical partition among the topics visible by a **com.rti.dds.publication.Publisher** (p. 1277) and a **com.rti.dds.subscription.Subscriber** (p. 1478).

6.36.2 Variable Documentation

6.36.2.1 final QosPolicyId_t PARTITION_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1233).

6.37 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. 1237).*

6.37.1 Detailed Description

Specifies how the samples representing changes to data instances are presented to a subscribing application.

6.37.2 Variable Documentation

6.37.2.1 final QosPolicyId_t PRESENTATION_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1237).

6.38 PROFILE

*<<**eXtension**>> (p. 270)* 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. 270) *Identifier for* ***com.rti.dds.infrastructure.ProfileQosPolicy*** *(p. 1247)*

6.38.1 Detailed Description

*<<**eXtension**>> (p. 270)* Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

6.38.2 Variable Documentation

6.38.2.1 final QosPolicyId_t PROFILE_QOS_POLICY_ID [static, inherited]

*<<**eXtension**>> (p. 270)* Identifier for **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1247)

6.39 PROPERTY

*<<**eXtension**>>* *(p. 270)* 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.

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 which facilitate management of the properties in the input policy.

- class **PropertySeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.Property_t*** *(p. 1250) >.*

6.39.1 Detailed Description

*<<**eXtension**>>* *(p. 270)* 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.

See **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252)

6.40 PUBLISH_MODE

*<<**eXtension**>> (p. 270)* Specifies how RTI Connext sends application data on the network. This QoS policy can be used to tell RTI Connext to use its *own* thread to send data, instead of the user thread.

Classes

- class **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 QosPolicyId_t **PUBLISHMODE_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.PublishModeQosPolicy*** *(p. 1308)*

6.40.1 Detailed Description

*<<**eXtension**>> (p. 270)* 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.

6.40.2 Variable Documentation

6.40.2.1 final QosPolicyId_t PUBLISHMODE_QOS_POLICY_ID

`[static, inherited]`

*<<**eXtension**>> (p. 270)* Identifier for **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1308)

6.41 QoS Policies

Quality of Service (QoS) policies.

Modules

- **ASYNCHRONOUS_PUBLISHER**

  <<**eXtension**>> (p. 270) *Specifies the asynchronous publishing settings of the* ***com.rti.dds.publication.Publisher*** *(p. 1277) instances.*

- **AVAILABILITY**

  <<**eXtension**>> (p. 270) *Configures the availability of data.*

- **BATCH**

  <<**eXtension**>> (p. 270) *Batch QoS policy used to enable batching in* ***com.rti.dds.publication.DataWriter*** *(p. 538) instances.*

- **DATABASE**

  <<**eXtension**>> (p. 270) *Various threads and resource limits settings used by RTI Connext to control its internal database.*

- **DATA_READER_PROTOCOL**

  <<**eXtension**>> (p. 270) *Specifies the DataReader-specific protocol QoS.*

- **DATA_READER_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 270) *Various settings that configure how DataReaders allocate and use physical memory for internal resources.*

- **DATA_WRITER_PROTOCOL**

  <<**eXtension**>> (p. 270) *Along with* ***com.rti.dds.infrastructure.WireProtocolQosPolicy*** *(p. 1709) and* ***com.rti.dds.infrastructure.DataReaderProtocolQosPolicy*** *(p. 504), this QoS policy configures the DDS on-the-network protocol (RTPS).*

- **DATA_WRITER_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 270) *Various settings that configure how a* ***com.rti.dds.publication.DataWriter*** *(p. 538) 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. 1277) entities to the same instance of data (i.e., matching **com.rti.dds.topic.Topic** (p. 1545) and key).*

ˆ **DISCOVERY_CONFIG**

<<**eXtension**>> (p. 270) *Specifies the discovery configuration QoS.*

ˆ **DISCOVERY**

<<**eXtension**>> (p. 270) *Specifies the attributes required to discover participants in the domain.*

ˆ **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**

<<**eXtension**>> (p. 270) *Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

ˆ **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. 473) entities that join the network later.*

ˆ **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. 765) setting of DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS or DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS.*

ˆ **ENTITY_FACTORY**

*A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) types.*

ˆ **ENTITY_NAME**

<<**eXtension**>> (p. 270) *Assigns a name to a **com.rti.dds.domain.DomainParticipant** (p. 629). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

ˆ **EVENT**

<<**eXtension**>> (p. 270) *Configures the internal thread in a DomainParticipant that handles timed events.*

^ **EXCLUSIVE_AREA**

<<**eXtension**>> (p. 270) *Configures multi-thread concurrency and deadlock prevention capabilities.*

^ **GROUP_DATA**

Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 153) 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. 538) is considered valid.*

^ **LIVELINESS**

Specifies and configures the mechanism that allows ***com.rti.dds.subscription.DataReader*** *(p. 473) entities to detect when* ***com.rti.dds.publication.DataWriter*** *(p. 538) entities become disconnected or "dead.".*

^ **LOCATORFILTER**

<<**eXtension**>> (p. 270) *The QoS policy used to report the configuration of a MultiChannel DataWriter as part of builtin.PublicationBuiltinTopicData.*

^ **LOGGING**

<<**eXtension**>> (p. 270) *Configures the RTI Connext logging facility.*

^ **MULTICHANNEL**

<<**eXtension**>> (p. 270) *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. 538) (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. 538) objects that attempt to modify the same instance of a data type (identified by* ***com.rti.dds.topic.Topic*** *(p. 1545) + key).*

ˆ **PARTITION**

Set of strings that introduces a logical partition among the topics visible by a ***com.rti.dds.publication.Publisher*** *(p. 1277) and a* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

ˆ **PRESENTATION**

Specifies how the samples representing changes to data instances are presented to a subscribing application.

ˆ **PROFILE**

<<**eXtension**>> (p. 270) *Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.*

ˆ **PROPERTY**

<<**eXtension**>> (p. 270) *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.*

ˆ **PUBLISH_MODE**

<<**eXtension**>> (p. 270) *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.*

ˆ **READER_DATA_LIFECYCLE**

Controls how a DataReader manages the lifecycle of the data that it has received.

ˆ **RECEIVER_POOL**

<<**eXtension**>> (p. 270) *Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).*

ˆ **RELIABILITY**

Indicates the level of reliability offered/requested by RTI Connext.

ˆ **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.

^ SYSTEM_RESOURCE_LIMITS

<<**eXtension**>> (p. 270) *Configures DomainParticipant-independent resources used by RTI Connext.*

^ TIME_BASED_FILTER

Filter that allows a ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 153) during discovery.*

^ TRANSPORT_BUILTIN

<<**eXtension**>> (p. 270) *Specifies which built-in transports are used.*

^ TRANSPORT_MULTICAST

<<**eXtension**>> (p. 270) *Specifies the multicast address on which a* ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 629) level) transports with which to receive the multicast data.*

^ TRANSPORT_PRIORITY

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

^ TRANSPORT_SELECTION

<<**eXtension**>> (p. 270) *Specifies the physical transports that a* ***com.rti.dds.publication.DataWriter*** *(p. 538) or* ***com.rti.dds.subscription.DataReader*** *(p. 473) may use to send or receive data.*

^ TRANSPORT_UNICAST

<<**eXtension**>> (p. 270) *Specifies a subset of transports and a port number that can be used by an Entity to receive data.*

^ TYPESUPPORT

<<**eXtension**>> (p. 270) *Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.*

^ USER_DATA

Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 153) during discovery.*

- **WIRE_PROTOCOL**

  <<**eXtension**>> (p. 270) *Specifies the wire protocol related attributes for the* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- **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. 1318).*

- class **QosPolicyCountSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.QosPolicyCount*** *(p. 1315) >.*

- class **QosPolicyId_t**

  Type to identify `QosPolicies`*.*

6.41.1 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).

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. 912) objects in the system such as **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478), and **com.rti.dds.domain.DomainParticipant** (p. 629).

6.41.2 Specifying QoS on entities

QosPolicies can be set programmatically when an **com.rti.dds.infrastructure.Entity** (p. 912) is created, or modified with the **com.rti.dds.infrastructure.Entity** (p. 912)'s **set_qos (abstract)** (p. 913) 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. 225).

To customize a **com.rti.dds.infrastructure.Entity** (p. 912)'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. 913) operation, or when the **com.rti.dds.infrastructure.Entity** (p. 912) 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. 912) creation time or else prior to calling the **com.rti.dds.infrastructure.Entity.enable** (p. 915) operation on the entity.

Each **com.rti.dds.infrastructure.Entity** (p. 912) can be configured with a list of `QosPolicy` objects. However, not all QosPolicies are supported by each **com.rti.dds.infrastructure.Entity** (p. 912). For instance, a **com.rti.dds.domain.DomainParticipant** (p. 629) supports a different set of QosPolicies than a **com.rti.dds.topic.Topic** (p. 1545) or a **com.rti.dds.publication.Publisher** (p. 1277).

6.41.3 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. 1478) requests to receive data reliably while the corresponding **com.rti.dds.publication.Publisher** (p. 1277) 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. 912) objects and will record this fact by means of the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS on the publisher end and 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. 1154) or a **com.rti.dds.infrastructure.Condition** (p. 451).

The following **properties** are defined on a QosPolicy.

- **RxO (p. 97) 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. 97) property:

  - **RxO** (p. 97) = **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. 97) = **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. 97) = **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.

ˆ **Changeable** (p. 98) **property**

Determines whether a QosPolicy can be changed.

NO (p. 98) – policy can only be specified at **com.rti.dds.infrastructure.Entity** (p. 912) creation time.

UNTIL ENABLE (p. 98) – policy can only be changed before the **com.rti.dds.infrastructure.Entity** (p. 912) is enabled.

YES (p. 98) – policy can be changed at any time.

6.42 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. 1328).*

6.42.1 Detailed Description

Controls how a DataReader manages the lifecycle of the data that it has received.

6.42.2 Variable Documentation

6.42.2.1 final QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy** (p. 1328).

6.43 RECEIVER_POOL

<<*eXtension*>> *(p. 270)* 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. 270) *Identifier for* ***com.rti.dds.infrastructure.ReceiverPoolQosPolicy*** *(p. 1331)*

6.43.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

6.43.2 Variable Documentation

6.43.2.1 final QosPolicyId_t RECEIVERPOOL_QOS_POLICY_ID

`[static, inherited]`

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1331)

6.44 RELIABILITY

Indicates the level of reliability offered/requested by RTI Connext.

Classes

- class **ReliabilityQosPolicy**

  Indicates the level of reliability offered/requested by RTI Connext.

- class **ReliabilityQosPolicyKind**

  Kinds of reliability.

Variables

- static final QosPolicyId_t **RELIABILITY_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.ReliabilityQosPolicy*** *(p. 1336).*

6.44.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Connext.

6.44.2 Variable Documentation

6.44.2.1 final QosPolicyId_t RELIABILITY_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1336).

6.45 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. 1356*).*

- static final int **LENGTH_UNLIMITED**

  A special value indicating an unlimited quantity.

6.45.1 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.

6.45.2 Variable Documentation

6.45.2.1 final QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356).

6.45.2.2 final int LENGTH_UNLIMITED [static, inherited]

A special value indicating an unlimited quantity.

6.46 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_ENABLED**

  *Operation invoked on a **com.rti.dds.infrastructure.Entity*** (p. 912) *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. Can only returned by operations that are unsupported.

6.46.1 Detailed Description

Types of return codes.

6.46.2 Standard Return Codes

Any void operation that documents that it may throw an exception of type RETCODE_ERROR may throw exactly RETCODE_ERROR or RETCODE_ILLEGAL_OPERATION. Any such operation that takes one or more input parameters may additionally throw the subclass RETCODE_BAD_PARAMETER. Any operation on an object created from any of the factories may additionally throw the subclass RETCODE_ALREADY_DELETED. Any operation that is stated as optional may additionally throw the subclass RETCODE_UNSUPPORTED.

Thus, the standard return codes are:

- RETCODE_OK
- RETCODE_ERROR
- RETCODE_ILLEGAL_OPERATION
- RETCODE_ALREADY_DELETED
- RETCODE_BAD_PARAMETER
- RETCODE_UNSUPPORTED

Operations that may throw any other exception type will state so explicitly.

6.47 Sequence Number Support

<<*eXtension*>> *(*p. 270*)* Sequence number type and defines.

Classes

- class **SequenceNumber_t**

  Type for sequence *number representation.*

6.47.1 Detailed Description

<<*eXtension*>> *(*p. 270*)* Sequence number type and defines.

6.48 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.

6.48.1 Detailed Description

Kinds of communication status.

Entity:

com.rti.dds.infrastructure.Entity (p. 912)

QoS:

QoS Policies (p. 90)

Listener:

com.rti.dds.infrastructure.Listener (p. 1154)

Each concrete **com.rti.dds.infrastructure.Entity** (p. 912) 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. 912).

When these status values change, the corresponding **com.rti.dds.infrastructure.StatusCondition** (p. 1452) objects are activated and the proper **com.rti.dds.infrastructure.Listener** (p. 1154) 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. 1154) or the **com.rti.dds.infrastructure.WaitSet** (p. 1695) / **com.rti.dds.infrastructure.Condition** (p. 451) mechanism. The two mechanisms may be combined in the application (e.g., using **com.rti.dds.infrastructure.WaitSet** (p. 1695) (s) / **com.rti.dds.infrastructure.Condition** (p. 451) (s) to access the data and **com.rti.dds.infrastructure.Listener** (p. 1154) (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. 1154) mechanism is used first and then the **com.rti.dds.infrastructure.WaitSet** (p. 1695) objects are signalled.

The statuses may be classified into:

- *read communication statuses*: i.e., those that are related to arrival of data, namely StatusKind.DATA_ON_READERS_STATUS and 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 (StatusKind.DATA_ON_READERS_STATUS and 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).

6.48.2 Changes in Status

Associated with each one of an **com.rti.dds.infrastructure.Entity** (p. 912)'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.

6.48.2.1 Changes in plain communication status

For the plain communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever the plain communication status changes and it is reset to false each time the application accesses the plain communication status via the proper `get_<plain communication status>()` operation on the **com.rti.dds.infrastructure.Entity** (p. 912).

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 StatusKind.REQUESTED_DEADLINE_MISSED_STATUS will become TRUE each time new deadline occurs (which increases the **com.rti.dds.subscription.RequestedDeadlineMissedStatus.total_count** (p. 1353) field). 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. 484) method on the proper Entity

6.48.2.2 Changes in read communication status

For the read communication status, the `StatusChangedFlag` flag is initially set to FALSE. The `StatusChangedFlag` becomes TRUE when either a data-sample arrives or else the **com.rti.dds.subscription.ViewStateKind** (p. 1689), **com.rti.dds.subscription.SampleStateKind** (p. 1430), or **com.rti.dds.subscription.InstanceStateKind** (p. 1086) of any existing sample changes for any reason other than a call to com.rti.dds.topic.example.FooDataReader.read, com.rti.dds.topic.example.FooDataReader.take 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. 1086) 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. 538) that owns it if **OWNERSHIP** (p. 83) QoS kind= OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS
    * or by any **com.rti.dds.publication.DataWriter** (p. 538) if **OWNERSHIP** (p. 83) QoS kind= OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS
  - The loss of liveliness of the **com.rti.dds.publication.DataWriter** (p. 538) of an instance for which there is no other **com.rti.dds.publication.DataWriter** (p. 538).
  - The arrival of the notification that an instance has been unregistered by the only **com.rti.dds.publication.DataWriter** (p. 538) that is known to be writing the instance.

Depending on the kind of `StatusChangedFlag`, the flag transitions to FALSE again as follows:

- The 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. 473).
- The 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. 473) belonging to the **com.rti.dds.subscription.Subscriber** (p. 1478).
  - The read or take operation (or their variants) is called on any **com.rti.dds.subscription.DataReader** (p. 473) belonging to the **com.rti.dds.subscription.Subscriber** (p. 1478).

See also:

com.rti.dds.infrastructure.Listener (p. 1154)
com.rti.dds.infrastructure.WaitSet (p. 1695),
com.rti.dds.infrastructure.Condition (p. 451)

6.48.3 Variable Documentation

6.48.3.1 final int STATUS_MASK_NONE [static, inherited]

No bits are set.

6.48.3.2 final int STATUS_MASK_ALL [static, inherited]

All bits are set.

6.49 SYSTEM_RESOURCE_LIMITS

<<*eXtension*>> *(p. 270)* Configures DomainParticipant-independent resources used by RTI Connext.

Classes

- class **SystemResourceLimitsQosPolicy**

  Configures ***com.rti.dds.domain.DomainParticipant*** *(p. 629)-independent resources used by RTI Connext. Mainly used to change the maximum number of* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) entities that can be created within a single process (address space).*

Variables

- static final QosPolicyId_t **SYSTEMRESOURCELIMITS_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy*** *(p. 1524)*

6.49.1 Detailed Description

<<*eXtension*>> *(p. 270)* Configures DomainParticipant-independent resources used by RTI Connext.

6.49.2 Variable Documentation

6.49.2.1 final QosPolicyId_t SYSTEMRESOURCELIMITS_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy** (p. 1524)

6.50 Thread Settings

The properties of a thread of execution.

Classes

- class **ThreadSettings_t**

  The properties of a thread of execution.

- class **ThreadSettingsCpuRotationKind**

  Determines how ***com.rti.dds.infrastructure.ThreadSettings_t.cpu_list*** *(p. 1532) 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.

6.50.1 Detailed Description

The properties of a thread of execution.

6.50.2 Variable Documentation

6.50.2.1 final int THREAD_SETTINGS_KIND_MASK_DEFAULT [static, inherited]

The mask of default thread options.

6.51 TIME_BASED_FILTER

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 1541).*

6.51.1 Detailed Description

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 473) to specify that it is interested only in (potentially) a subset of the values of the data.

6.51.2 Variable Documentation

6.51.2.1 final QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_-ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy** (p. 1541).

6.52 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Classes

- class **TopicDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.*

Variables

- static final QosPolicyId_t **TOPICDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.TopicDataQosPolicy** (p. 1559).*

6.52.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

6.52.2 Variable Documentation

6.52.2.1 final QosPolicyId_t TOPICDATA_QOS_POLICY_ID

`[static, inherited]`

Identifier for **com.rti.dds.infrastructure.TopicDataQosPolicy** (p. 1559).

6.53 TRANSPORT_BUILTIN

<<*eXtension*>> *(*p. *270)* 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. 270) *Identifier for* ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy*** *(*p. *1580)*

- static final String **UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport.

- static final String **SHMEM_ALIAS**

  Alias name for the shared memory built-in transport.

- static final String **UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport.

- static final int **MASK_NONE**

  None of the built-in transports will be registered automatically when the ***com.rti.dds.domain.DomainParticipant*** *(*p. *629) is enabled. The user must explictly register transports using TransportSupport.register_transport.*

- static final int **MASK_DEFAULT**

  The default value of ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask*** *(*p. *1581).*

- static final int **MASK_ALL**

  All the available built-in transports are registered automatically when the ***com.rti.dds.domain.DomainParticipant*** *(*p. *629) is enabled.*

6.53.1 Detailed Description

<<*eXtension*>> *(p. 270)* Specifies which built-in transports are used.

See also:

Changing the automatically registered built-in transports (p. 255)

6.53.2 Variable Documentation

6.53.2.1 final QosPolicyId_t TRANSPORTBUILTIN_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580)

6.53.2.2 final String UDPv4_ALIAS [static, inherited]

Alias name for the UDPv4 built-in transport.

6.53.2.3 final String SHMEM_ALIAS [static, inherited]

Alias name for the shared memory built-in transport.

6.53.2.4 final String UDPv6_ALIAS [static, inherited]

Alias name for the UDPv6 built-in transport.

6.53.2.5 final int MASK_NONE [static, inherited]

None of the built-in transports will be registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled. The user must explictly register transports using TransportSupport.register_transport.

See also:

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.53.2.6 final int MASK_DEFAULT [static, inherited]

The default value of **com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask** (p. 1581).

The set of builtin transport plugins that will be automatically registered with the participant by default. The user can register additional transports using TransportSupport.register_transport.

See also:

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.53.2.7 final int MASK_ALL [static, inherited]

All the available built-in transports are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled.

See also:

com.rti.dds.infrastructure.TransportBuiltinKindMask

6.54 TRANSPORT_MULTICAST

<<*eXtension*>> *(p. 270)* Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) 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. 473) 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. 629) level) transports with which to receive the multicast data.*

- class **TransportMulticastQosPolicy**

  *Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) 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. 270) *Identifier for **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1590)*

- 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.

6.54.1 Detailed Description

<<*eXtension*>> *(p. 270)* Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) level) transports with which to receive the multicast data.

6.54.2 Variable Documentation

6.54.2.1 final QosPolicyId_t TRANSPORTMULTICAST_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1590)

6.54.2.2 final TransportMulticastQosPolicyKind AUTOMATIC_TRANSPORT_MULTICAST_QOS [static, inherited]

Initial value:

```
new TransportMulticastQosPolicyKind(
                                    "AUTOMATIC_TRANSPORT_MULTICAST_QOS", 0)
```

Transport Multicast Policy Kind.

See also:

com.rti.dds.infrastructure.TransportMulticastQosPolicy.AUTOMATIC_TRANSPORT_MULTICAST_QOS

6.54.2.3 final TransportMulticastQosPolicyKind UNICAST_ONLY_TRANSPORT_MULTICAST_QOS = new TransportMulticastQosPolicyKind("UNICAST_ONLY_TRANSPORT_MULTICAST_QOS", 1) [static, inherited]

Transport Multicast Policy Kind.

See also:

com.rti.dds.infrastructure.TransportMulticastQosPolicy.UNICAST_-ONLY_TRANSPORT_MULTICAST_QOS

6.55 TRANSPORT_PRIORITY

This QoS policy allows the application to take advantage of transports 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. 1598).*

6.55.1 Detailed Description

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

6.55.2 Variable Documentation

6.55.2.1 final QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.TransportPriorityQosPolicy** (p. 1598).

6.56 TRANSPORT_SELECTION

<<*eXtension*>> *(p. 270)* Specifies the physical transports that a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) may use to send or receive data.

Classes

- class **TransportSelectionQosPolicy**

  *Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) may use to send or receive data.*

Variables

- static final QosPolicyId_t **TRANSPORTSELECTION_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1600)*

6.56.1 Detailed Description

<<*eXtension*>> *(p. 270)* Specifies the physical transports that a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) may use to send or receive data.

6.56.2 Variable Documentation

6.56.2.1 final QosPolicyId_t TRANSPORTSELECTION_QOS_POLICY_ID [static, inherited]

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1600)

6.57 TRANSPORT_UNICAST

*<<**eXtension**>> (p. 270)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Classes

ˆ class **TransportUnicastQosPolicy**

*Specifies a subset of transports and a port number that can be used by an **Entity** (p. 912) to receive data.*

Variables

ˆ static final QosPolicyId_t **TRANSPORTUNICAST_QOS_POLICY_ID**

<<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1605)*

6.57.1 Detailed Description

*<<**eXtension**>> (p. 270)* Specifies a subset of transports and a port number that can be used by an Entity to receive data.

6.57.2 Variable Documentation

6.57.2.1 final QosPolicyId_t TRANSPORTUNICAST_QOS_POLICY_ID [static, inherited]

*<<**eXtension**>> (p. 270)* Identifier for **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1605)

6.58 TYPESUPPORT

*<<**eXtension**>> (p. 270)* Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

Classes

- class **TypeSupportQosPolicy**

  Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

Variables

- static final QosPolicyId_t **TYPESUPPORT_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1652)*

- static final QosPolicyId_t **ENTITYNAME_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1652)*

6.58.1 Detailed Description

*<<**eXtension**>> (p. 270)* Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

6.58.2 Variable Documentation

6.58.2.1 final QosPolicyId_t TYPESUPPORT_QOS_POLICY_ID

`[static, inherited]`

*<<**eXtension**>> (p. 270)* Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1652)

6.58.2.2 final QosPolicyId_t ENTITYNAME_QOS_POLICY_ID

`[static, inherited]`

<<*eXtension*>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.TypeSupportQosPolicy** (p. 1652)

6.59 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Classes

- class **UserDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.*

Variables

- static final QosPolicyId_t **USERDATA_QOS_POLICY_ID**

  *Identifier for **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680).*

6.59.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

6.59.2 Variable Documentation

6.59.2.1 final QosPolicyId_t USERDATA_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680).

6.60 Exception Codes

<<*eXtension*>> *(p. 270)* Exception codes.

Classes

- ^ class **BAD_PARAM**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- ^ class **BAD_TYPECODE**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- ^ class **BadKind**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- ^ class **BadMemberId**

  The specified TypeCode member ID is invalid.

- ^ class **BadMemberName**

  The specified TypeCode member name is invalid.

- ^ class **Bounds**

  A user exception thrown when a parameter is not within the legal bounds.

- ^ class **SystemException**

  System exception.

- ^ class **UserException**

  User exception.

6.60.1 Detailed Description

<<*eXtension*>> *(p. 270)* Exception codes.

These exceptions are used for error handling by the **Type Code Support** (p. 162) API.

6.61 WIRE_PROTOCOL

*<<**eXtension**>> (p. 270)* Specifies the wire protocol related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 629).

Classes

- class **RtpsReservedPortKind**

  *RTPS reserved port kind, used to identify the types of ports that can be reserved on **domain** (p. 317) 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. 629).*

- class **WireProtocolQosPolicyAutoKind**

  Kind of auto mechanism used to calculate the GUID prefix.

Variables

- static final QosPolicyId_t **WIREPROTOCOL_QOS_POLICY_ID**

  <<**eXtension**>> (p. 270) *Identifier for **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709)*

- 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. 1717).*

- 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 WireProtocolQosPolicyAutoKind("RTPS_AUTO_ID_FROM_IP", 0)

  Kind of auto mechanism used to calculate the GUID prefix.

- static final WireProtocolQosPolicyAutoKind **RTPS_AUTO_ID_FROM_MAC** = new WireProtocolQosPolicyAutoKind("RTPS_AUTO_ID_FROM_MAC", 1)

  Kind of auto mechanism used to calculate the GUID prefix.

6.61.1 Detailed Description

<<***eXtension***>> *(p. 270)* Specifies the wire protocol related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 629).

6.61.2 Variable Documentation

6.61.2.1 final QosPolicyId_t WIREPROTOCOL_QOS_POLICY_ID [static, inherited]

<<***eXtension***>> *(p. 270)* Identifier for **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709)

6.61.2.2 final int MASK_DEFAULT = BUILTIN_UNICAST | BUILTIN_MULTICAST | USER_UNICAST [static, inherited]

The default value of **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask** (p. 1717).

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. 1396) 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

6.61.2.3 final int MASK_NONE [static, inherited]

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

6.61.2.4 final int MASK_ALL [static, inherited]

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. 1396) 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

6.61.2.5 final RtpsWellKnownPorts_t RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS [static, inherited]

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. 1715) 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 **RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 131):

`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 **RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 131).

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports (p. 1715)
RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 131)

6.61.2.6 final RtpsWellKnownPorts_t INTEROPERABLE_RTPS_WELL_KNOWN_PORTS [static, inherited]

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. 1715) 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 **RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 131):

`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** (p. 317).

These settings are *not* backwards compatible with previous versions of the RTI Connext middleware that used fixed port mappings. For backwards compability, please use **RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 131).

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_well_known_ports (p. 1715)
RtpsWellKnownPorts_t.RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS (p. 131)

6.61.2.7 final WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_IP = new WireProtocolQosPolicyAutoKind("RTPS_AUTO_ID_FROM_IP", 0) [static, inherited]

Kind of auto mechanism used to calculate the GUID prefix.

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind.RTPS_AUTO_ID_FROM_IP

6.61.2.8 final WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_MAC = new WireProtocolQosPolicyAutoKind("RTPS_AUTO_ID_FROM_MAC", 1) [static, inherited]

Kind of auto mechanism used to calculate the GUID prefix.

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind.RTPS_AUTO_ID_FROM_MAC

6.62 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. 538) 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. 1722).*

6.62.1 Detailed Description

Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

6.62.2 Variable Documentation

6.62.2.1 final QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID [static, inherited]

Identifier for **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 1722).

6.63 String Built-in Type

Built-in type consisting of a single character string.

Classes

- ˆ class **StringSeq**

  Declares IDL `sequence` < *String* > .

- ˆ class **StringDataReader**

  <<**interface**>> (p. 271) *Instantiates* `DataReader` < *String* >.

- ˆ class **StringDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < *String* >.

- ˆ class **StringTypeSupport**

  <<**interface**>> (p. 271) *String type support.*

6.63.1 Detailed Description

Built-in type consisting of a single character string.

6.64 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. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* >.

- class **KeyedStringDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* >.

- class **KeyedStringSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* > .

- class **KeyedStringTypeSupport**

  <<**interface**>> (p. 271) *Keyed string type support.*

6.64.1 Detailed Description

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

6.65 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. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* >.

- class **BytesDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* >.

- class **BytesSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* > .

- class **BytesTypeSupport**

  <<**interface**>> (p. 271) ***com.rti.dds.type.builtin.Bytes*** *(p. 417) type support.*

6.65.1 Detailed Description

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

6.66 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. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- ^ class **KeyedBytesDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- ^ class **KeyedBytesSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- ^ class **KeyedBytesTypeSupport**

  <<**interface**>> (p. 271) ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) type support.*

6.66.1 Detailed Description

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

6.67 Sequence Support

The **com.rti.dds.util.Sequence** (p. 1432) 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. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 955).

- interface **Sequence**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo`.

- class **FooSeq**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 956).

6.67.1 Detailed Description

The **com.rti.dds.util.Sequence** (p. 1432) 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. 202) provide extension APIs for working with collections of primitive elements without the overhead of boxing the unboxing.

When you use the **rtiddsgen** (p. 290) 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

6.68 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. 695).

6.68.1 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".

6.68.2 Clock Selection Strategy

By default, both the internal and external clocks use the realtime 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.

6.68.3 Configuring Clock Selection

To configure the clock selection, use the **PROPERTY** (p. 88) QoS policy associated with the **com.rti.dds.domain.DomainParticipant** (p. 629).

See also:

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1252)

The following table lists the supported 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"

Table 6.1: *Clock Selection Properties*

6.69 Domain Module

Contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up RTI Connext.

Modules

- **DomainParticipantFactory**

  com.rti.dds.domain.DomainParticipantFactory *(p. 708) entity and associated elements*

- **DomainParticipants**

  com.rti.dds.domain.DomainParticipant *(p. 629) entity and associated elements*

- **Built-in Topics**

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

Variables

- 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. 712).*

6.69.1 Detailed Description

Contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up RTI Connext.

6.69.2 Variable Documentation

6.69.2.1 DomainParticipantFactory ThePartipantFactory = create_singletonI() [static, inherited]

Can be used as an alias for the singleton factory returned by the operation **com.rti.dds.domain.DomainParticipantFactory.get_instance()** (p. 712).

See also:

com.rti.dds.domain.DomainParticipantFactory.get_instance (p. 712)

6.70 DomainParticipantFactory

com.rti.dds.domain.DomainParticipantFactory (p. 708) entity and associated elements

Classes

- class **DomainParticipantFactory**

  <<**singleton**>> (p. 271) <<**interface**>> (p. 271) *Allows creation and destruction of* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) objects.*

- class **DomainParticipantFactoryQos**

  QoS policies supported by a ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

Variables

- static final DomainParticipantQos **PARTICIPANT_QOS_DEFAULT**

  Special value for creating a ***DomainParticipant*** *(p. 629) with default QoS.*

6.70.1 Detailed Description

com.rti.dds.domain.DomainParticipantFactory (p. 708) entity and associated elements

6.70.2 Variable Documentation

6.70.2.1 final DomainParticipantQos PARTICIPANT_QOS_DEFAULT [static, inherited]

Initial value:

```
new DomainParticipantQos()
```

Special value for creating a **DomainParticipant** (p. 629) with default QoS.

When used in **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 714), this special value is used to indicate that the **com.rti.dds.domain.DomainParticipant** (p. 629) should be created with

the default **com.rti.dds.domain.DomainParticipant** (p. 629) QoS by means of the operation **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos()** (p. 716) and using the resulting QoS to create the **com.rti.dds.domain.DomainParticipant** (p. 629).

When used in **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 716), 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. 716) operation had never been called.

When used in **com.rti.dds.domain.DomainParticipant.set_qos** (p. 677), this special value is used to indicate that the QoS of the **com.rti.dds.domain.DomainParticipant** (p. 629) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 708) that the **com.rti.dds.domain.DomainParticipant** (p. 629) belongs to.

RTI Connext treats this special value as a constant.

Note: You cannot use this value to *get* the default QoS values from the **DomainParticipant** (p. 629) factory; for this purpose, use **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos** (p. 716).

See also:

NDDS_DISCOVERY_PEERS (p. 55)
com.rti.dds.domain.DomainParticipantFactory.create_participant() (p. 714)
com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos() (p. 716)
com.rti.dds.domain.DomainParticipant.set_qos() (p. 677)

6.71 DomainParticipants

com.rti.dds.domain.DomainParticipant (p. 629) entity and associated elements

Classes

- interface **DomainParticipant**

  <<**interface**>> (p. 271) *Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 628) objects.*

- class **DomainParticipantAdapter**

  <<**eXtension**>> (p. 270) *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. 271) *Listener for participant status.*

- class **DomainParticipantQos**

  *QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 629) entity.*

Variables

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

  *Special value for creating a **com.rti.dds.topic.Topic** (p. 1545) with default QoS.*

- static final PublisherQos **PUBLISHER_QOS_DEFAULT** = new PublisherQos()

  *Special value for creating a **com.rti.dds.publication.Publisher** (p. 1277) with default QoS.*

- static final SubscriberQos **SUBSCRIBER_QOS_DEFAULT**

  *Special value for creating a **com.rti.dds.subscription.Subscriber** (p. 1478) with default QoS.*

- static final FlowControllerProperty_t **FLOW_CONTROLLER_PROPERTY_DEFAULT**

  <<**eXtension**>> (p. 270) *Special value for creating a **com.rti.dds.publication.FlowController** (p. 942) with default property.*

- static final String **SQLFILTER_NAME**

  <<**eXtension**>> (p. 270) *The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

- static final String **STRINGMATCHFILTER_NAME**

  <<**eXtension**>> (p. 270) *The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

6.71.1 Detailed Description

com.rti.dds.domain.DomainParticipant (p. 629) entity and associated elements

6.71.2 Variable Documentation

6.71.2.1 final TopicQos TOPIC_QOS_DEFAULT = new TopicQos() [static, inherited]

Special value for creating a **com.rti.dds.topic.Topic** (p. 1545) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670), this special value is used to indicate that the **com.rti.dds.topic.Topic** (p. 1545) should be created with the default **com.rti.dds.topic.Topic** (p. 1545) QoS by means of the operation get_default_topic_qos and using the resulting QoS to create the **com.rti.dds.topic.Topic** (p. 1545).

When used in **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642), 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. 642) operation had never been called.

When used in **com.rti.dds.topic.Topic.set_qos** (p. 1547), this special value is used to indicate that the QoS of the **com.rti.dds.topic.Topic** (p. 1545) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipant** (p. 629) that the **com.rti.dds.topic.Topic** (p. 1545) belongs to.

Note: 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. 641).

See also:

com.rti.dds.domain.DomainParticipant.create_topic (p. 670)
com.rti.dds.domain.DomainParticipant.set_default_topic_qos (p. 642)
com.rti.dds.topic.Topic.set_qos (p. 1547)

6.71.2.2 final PublisherQos PUBLISHER_QOS_DEFAULT = new PublisherQos() [static, inherited]

Special value for creating a **com.rti.dds.publication.Publisher** (p. 1277) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656), this special value is used to indicate that the **com.rti.dds.publication.Publisher** (p. 1277) should be created with the default **com.rti.dds.publication.Publisher** (p. 1277) QoS by means of the operation get_default_publisher_qos and using the resulting QoS to create the **com.rti.dds.publication.Publisher** (p. 1277).

When used in **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 644), 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. 644) operation had never been called.

When used in **com.rti.dds.publication.Publisher.set_qos** (p. 1289), this special value is used to indicate that the QoS of the **com.rti.dds.publication.Publisher** (p. 1277) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipant** (p. 629) that the **com.rti.dds.publication.Publisher** (p. 1277) belongs to.

Note: 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. 644).

See also:

com.rti.dds.domain.DomainParticipant.create_publisher (p. 656)
com.rti.dds.domain.DomainParticipant.set_default_publisher_qos (p. 644)
com.rti.dds.publication.Publisher.set_qos (p. 1289)

6.71.2.3 final SubscriberQos SUBSCRIBER_QOS_DEFAULT [static, inherited]

Initial value:

```
new SubscriberQos()
```

Special value for creating a **com.rti.dds.subscription.Subscriber** (p. 1478) with default QoS.

When used in **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659), this special value is used to indicate that the **com.rti.dds.subscription.Subscriber** (p. 1478) should be created with the default **com.rti.dds.subscription.Subscriber** (p. 1478) QoS by means of the operation get_default_subscriber_qos and using the resulting QoS to create the **com.rti.dds.subscription.Subscriber** (p. 1478).

When used in **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 649), 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. 649) operation had never been called.

When used in **com.rti.dds.subscription.Subscriber.set_qos** (p. 1493), this special value is used to indicate that the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1478) should be changed to match the current default QoS set in the **com.rti.dds.domain.DomainParticipant** (p. 629) that the **com.rti.dds.subscription.Subscriber** (p. 1478) belongs to.

Note: 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. 652).

See also:

com.rti.dds.domain.DomainParticipant.create_subscriber (p. 659)
com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 652)
com.rti.dds.subscription.Subscriber.set_qos (p. 1493)

6.71.2.4 final FlowControllerProperty_t FLOW_CONTROLLER_PROPERTY_DEFAULT [static, inherited]

Initial value:

```
new FlowControllerProperty_t()
```

*<<**eXtension**>> (p. 270)* Special value for creating a **com.rti.dds.publication.FlowController** (p. 942) with default property.

When used in **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 654), this special value is used to indicate that the

com.rti.dds.publication.FlowController (p. 942) should be created with the default **com.rti.dds.publication.FlowController** (p. 942) property by means of the operation get_default_flowcontroller_property and using the resulting QoS to create the **com.rti.dds.publication.FlowControllerProperty_t** (p. 946).

When used in **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 640), 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_flowcontroller_property** (p. 640) operation had never been called.

When used in **com.rti.dds.publication.FlowController.set_property** (p. 943), this special value is used to indicate that the property of the **com.rti.dds.publication.FlowController** (p. 942) should be changed to match the current default property set in the **com.rti.dds.domain.DomainParticipant** (p. 629) that the **com.rti.dds.publication.FlowController** (p. 942) belongs to.

Note: 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. 639).

See also:

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 654)
com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property (p. 640)
com.rti.dds.publication.FlowController.set_property (p. 943)

6.71.2.5 final String SQLFILTER_NAME [static, inherited]

<<***eXtension***>> *(p. 270)* The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.

See also:

Queries and Filters Syntax (p. 278)

6.71.2.6 final String STRINGMATCHFILTER_NAME [static, inherited]

<<***eXtension***>> *(p. 270)* 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. 278)

6.72 Built-in Topics

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

Packages

- package **com.rti.dds.domain.builtin**

  *Builtin **topic** (p. 350) for accessing information about the DomainParticipants discovered by RTI Connext.*

- package **com.rti.dds.publication.builtin**

  *Builtin **topic** (p. 350) for accessing information about the Publications discovered by RTI Connext.*

- package **com.rti.dds.subscription.builtin**

  *Builtin **topic** (p. 350) for accessing information about the Subscriptions discovered by RTI Connext.*

- package **com.rti.dds.topic.builtin**

  *Builtin **topic** (p. 350) for accessing information about the Topics discovered by RTI Connext.*

Classes

- class **ContentFilterProperty_t**

  <<**eXtension**>> (p. 270) *Type used to provide all the required information to enable content filtering.*

- class **Locator_t**

  <<**eXtension**>> (p. 270) *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. 1174)* >.

- class **ProductVersion_t**

  <<**eXtension**>> (p. 270) *Type used to represent the current version of RTI Connext.*

- class **ProtocolVersion_t**

<<**eXtension**>> (p. 270) *Type used to represent the version of the RTPS protocol.*

ˆ class **VendorId_t**

<<**eXtension**>> (p. 270) *Type used to represent the vendor of the service implementing the RTPS protocol.*

ˆ class **BuiltinTopicKey_t**

The key type of the built-in ***topic*** *(p. 350) types.*

6.72.1 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. 473) objects are introduced to be used by the application to access these discovery information.

The information can be accessed as if it was 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. 1154) or the **com.rti.dds.infrastructure.Condition** (p. 451) mechanisms.

The built-in data-readers all belong to a built-in **com.rti.dds.subscription.Subscriber** (p. 1478), which can be retrieved by using the method **com.rti.dds.domain.DomainParticipant.get_builtin_subscriber** (p. 684). The built-in **com.rti.dds.subscription.DataReader** (p. 473) objects can be retrieved by using the operation **com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1490), 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. 1478) 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. This 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. 473) will not provide data pertaining to entities created from the same **com.rti.dds.domain.DomainParticipant** (p. 629) under the assumption that such entities are already known to the application that created them.

Refer to builtin.ParticipantBuiltinTopicData, builtin.TopicBuiltinTopicData, builtin.SubscriptionBuiltinTopicData and builtin.PublicationBuiltinTopicData for a description of all the built-in topics and their contents.

The QoS of the built-in **com.rti.dds.subscription.Subscriber** (p. 1478) and **com.rti.dds.subscription.DataReader** (p. 473) objects is given by the following table:

QoS	Value
com.rti.dds.infrastructure.UserDataQosPolicy (p. 1680)	0-length sequence
com.rti.dds.infrastructure.TopicDataQosPolicy (p. 1559)	0-length sequence
com.rti.dds.infrastructure.GroupDataQosPolicy (p. 1064)	0-length sequence
com.rti.dds.infrastructure.DurabilityQosPolicy (p. 765)	DurabilityQosPolicyKind = TRANSIENT_LOCAL_DURABILITY_QOS
com.rti.dds.infrastructure.DurabilityServiceQosPolicy (p. 773)	Does not apply as durability.kind = com.rti.dds.infrastructure.DurabilityQosPolicyKind (p. 770) is DurabilityQosPolicyKind.TRANSIENT_LOCAL_DURABILITY_QOS
com.rti.dds.infrastructure.PresentationQosPolicy (p. 1237)	access_scope = PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS coherent_access = false ordered_access = false
com.rti.dds.infrastructure.DeadlineQosPolicy (p. 604)	Period = infinite
com.rti.dds.infrastructure.LatencyBudgetQosPolicy (p. 1148)	duration = 0
com.rti.dds.infrastructure.OwnershipQosPolicy (p. 1216)	OwnershipQosPolicyKind = SHARED_OWNERSHIP_QOS
com.rti.dds.infrastructure.OwnershipStrengthQosPolicy (p. 1225)	value = 0
com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1164)	kind = LivelinessQosPolicyKind = AUTOMATIC_LIVELINESS_QOS lease_duration = 0
com.rti.dds.infrastructure.TimeBasedFilterQosPolicy (p. 1541)	minimum_separation = 0
com.rti.dds.infrastructure.PartitionQosPolicy (p. 1233)	0-length sequence
com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1330)	kind = ReliabilityQosPolicyKind = RELIABLE_RELIABILITY_QOS max_blocking_time = 100 milliseconds
com.rti.dds.infrastructure.DestinationOrderQosPolicy (p. 607)	DestinationOrderQosPolicyKind = BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
com.rti.dds.infrastructure.HistoryQosPolicy	kind = HistoryQosPolicyKind.KEEP

6.73 Topic Module

Contains the **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) classes, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.

Modules

- **Topics**

  com.rti.dds.topic.Topic *(p. 1545) entity and associated elements*

- **User Data Type Support**

  Defines generic classes and macros to support user data types.

- **Type Code Support**

  <<**eXtension**>> (p. 270) *A* TypeCode *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. 170) capability or to inspect the type information you receive from remote readers and writers.*

- **Built-in Types**

  <<**eXtension**>> (p. 270) *RTI Connext provides a set of very simple data types for you to use with the topics in your application.*

- **Dynamic Data**

  <<**eXtension**>> (p. 270) *The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.*

6.73.1 Detailed Description

Contains the **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) classes, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.

6.74 Topics

com.rti.dds.topic.Topic (p. 1545) entity and associated elements

Classes

- interface **ContentFilter**

  <<**interface**>> (p. 271) *Interface to be used by a custom filter of a* ***com.rti.dds.topic.ContentFilteredTopic*** *(p. 458)*

- interface **ContentFilteredTopic**

  <<**interface**>> (p. 271) *Specialization of* ***com.rti.dds.topic.TopicDescription*** *(p. 1561) that allows for content-based subscriptions.*

- class **InconsistentTopicStatus**

  StatusKind.INCONSISTENT_TOPIC_STATUS.

- interface **MultiTopic**

  *[***Not supported (optional)***]* <<**interface**>> (p. 271) *A specialization of* ***com.rti.dds.topic.TopicDescription*** *(p. 1561) that allows subscriptions that combine/filter/rearrange data coming from several topics.*

- interface **Topic**

  <<**interface**>> (p. 271) *The most basic description of the data to be published and subscribed.*

- class **TopicAdapter**

  <<**eXtension**>> (p. 270) *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. 1545) entity and associated elements*

- interface **TopicListener**

  <<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for* ***com.rti.dds.topic.Topic*** *(p. 1545) entities.*

- class **TopicQos**

  QoS policies supported by a ***com.rti.dds.topic.Topic*** *(p. 1545) entity.*

6.74.1 Detailed Description

com.rti.dds.topic.Topic (p. 1545) entity and associated elements

6.75 User Data Type Support

Defines generic classes and macros to support user data types.

Classes

- class **InstanceHandle_t**

  Type definition for an instance handle.

- class **InstanceHandleSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.infrastructure.InstanceHandle_t*** (p. 1080) > .

- class **Foo**

  A representative user-defined data type.

- class **FooTypeSupport**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific interface.*

- interface **TypeSupport**

  <<**interface**>> (p. 271) *An abstract* marker *interface that has to be specialized for each concrete user data type that will be used by the application.*

- class **Foo**

  A representative user-defined data type.

6.75.1 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** (p. 290) utility. The complete set of automatic classes created for a hypothetical user data type named `Foo` are shown below.

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

See also:

rtiddsgen (p. 290)

6.76 Type Code Support

*<<**eXtension**>> (p. 270)* A *TypeCode* 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. 170) 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 **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 **TypeCode** (p. 1611) 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** (p. 290) 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.*

6.76.1 Detailed Description

<<***eXtension***>> *(*p. 270*)* A *TypeCode* 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. 170) 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 TypeCode class, which has an analogue in CORBA.

A TypeCode value consists of a type code *kind* (represented by the TCKind enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own TypeCode, 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 TypeCode.equals method is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

6.76.2 Accessing a Local TypeCode

When generating types with **rtiddsgen** (p. 290), type codes are enabled by default. (The *-notypecode* option can be used to disable generation of TypeCode information.) For these types, a TypeCode may be accessed via the **FooTypeCode.VALUE** member.

This API also includes support for dynamic creation of TypeCode values, typically for use with the **Dynamic Data** (p. 170) API. You can create a TypeCode using the TypeCodeFactory class. You will construct the TypeCode 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:

- TypeCodeFactory.get_primitive_tc, which provides the TypeCode instances corresponding to the primitive types (e.g. TCKind.TK_LONG, TCKind.TK_SHORT, and so on).
- TypeCodeFactory.create_string_tc and TypeCodeFactory.create_wstring_tc create a TypeCode representing a text string with a certain *bound* (i.e. maximum length).
- TypeCodeFactory.create_array_tc and TypeCodeFactory.create_sequence_tc create a TypeCode for a collection based on the TypeCode for its elements.
- TypeCodeFactory.create_struct_tc, TypeCodeFactory.create_value_tc, and TypeCodeFactory.create_sparse_tc create a TypeCode for a structured type.

6.76.3 Accessing a Remote TypeCode

In addition to being used locally, RTI Connext can transmit TypeCode 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. 170)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote TypeCode 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. 153). Discovered TypeCode 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. 551) or **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 487).

The space available locally to deserialize a discovered remote TypeCode is specified by the **com.rti.dds.domain.DomainParticipant** (p. 629)'s **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_code_max_serialized_length** (p. 755) QoS parameter. To support especially complex type codes, it may be necessary for you to increase the value of this parameter.

See also:

TypeCode
Dynamic Data (p. 170)
rtiddsgen (p. 290)
builtin.SubscriptionBuiltinTopicData
builtin.PublicationBuiltinTopicData

6.77 Built-in Types

<<*eXtension*>> *(p. 270)* 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.

6.77.1 Detailed Description

<<*eXtension*>> *(p. 270)* 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.KeyedString** (p. 1123): 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.Bytes** (p. 417): A payload consisting of an opaque variable-length array of bytes. This type has no key.
- **com.rti.dds.type.builtin.KeyedBytes** (p. 1095): 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.KeyedString** (p. 1123) types are appropriate for simple text-based applications. The **com.rti.dds.type.builtin.Bytes** (p. 417) and **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) 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. 458)).

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** (p. 290) utility
- At runtime, by using the **Dynamic Data** (p. 170) API

6.77.2 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. 1252) 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 DomainParticipant unless they are overwrittem in the DataWriters and DataReaders.

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.

6.77.3 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.

For more information about the built-in types, including how to control memory usage and maximum lengths, please see chapter 3, *Data Types and Data Samples*, in the RTI Connext User's Manual.

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the **com.rti.dds.type.builtin.StringDataWriter** (p. 1468) or received the **com.rti.dds.type.builtin.StringDataReader** (p. 1465) (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.KeyedStringDataWriter** (p. 1133) or **com.rti.dds.type.builtin.KeyedStringDataReader** (p. 1125) (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.KeyedStringDataWriter** (p. 1133) or received by the **com.rti.dds.type.builtin.KeyedStringDataReader** (p. 1125) (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.BytesDataWriter** (p. 424) or received by the **com.rti.dds.type.builtin.BytesDataReader** (p. 420). Default: dds.builtin_type.octets.max_size if defined. Otherwise, 2048.
dds.builtin_type.keyed_octets.alloc_-key_size	Max-imum size of the key published by the **com.rti.dds.type.builtin.KeyedBytesDataWriter** (p. 1106) or received by the **com.rti.dds.type.builtin.KeyedBytesDataReader** (p. 1098) (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.KeyedBytesDataWriter** (p. 1106) or received by a **com.rti.dds.type.builtin.KeyedBytesDataReader** (p. 1098).

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

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

6.78 Dynamic Data

*<<**eXtension**>>* *(p. 270)* 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. 780) 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. 780) objects.*

- class **DynamicDataReader**

  Reads (subscribes to) objects of type ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780).*

- class **DynamicDataSeq**

  An ordered collection of ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) elements.*

- class **DynamicDataTypeProperty_t**

  A collection of attributes used to configure ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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. 780) objects.*

- class **DynamicDataWriter**

  Writes (publishes) objects of type ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780).*

Functions

- **DynamicDataInfo** ()

  A descriptor for a ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) object.*

- **DynamicDataInfo** (int member_count, int stored_size, boolean is_optimized_storage)

  A descriptor for a ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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. 849).*

- static final DynamicDataTypeProperty_t **TYPE_PROPERTY_DEFAULT**

  Sentinel constant indicating default values for ***com.rti.dds.dynamicdata.DynamicDataTypeProperty_t*** *(p. 883).*

6.78.1 Detailed Description

<<***eXtension***>> *(p. 270)* 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 TypeCode (see Type Code Support (p. 162)) that defines the type definition you want to use.

A TypeCode includes a type's *kind* (TCKind), *name*, and *members* (that is, fields). You can create your own TypeCode using the TypeCodeFactory class – see, for example, the TypeCodeFactory.create_struct_tc method. Alternatively, you can use a remote TypeCode that you discovered on the network (see **Built-in Topics** (p. 153)) or one generated by **rtiddsgen** (p. 290).

2. Wrap the TypeCode in a com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 887) object.

See the constructor DynamicDataTypeSupport.DynamicDataTypeSupport. This object lets you connect the type definition to a **com.rti.dds.domain.DomainParticipant** (p. 629) and manage data samples (of type **com.rti.dds.dynamicdata.DynamicData** (p. 780)).

3. Register the com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 887) with one or more domain participants.

See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 889). 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. 1545) from the com.rti.dds.domain.DomainParticipant (p. 629).

Use the name under which you registered your data type – see **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670). This **com.rti.dds.topic.Topic** (p. 1545) is what you will use to produce and consume data.

5. Create a com.rti.dds.dynamicdata.DynamicDataWriter (p. 893) and/or com.rti.dds.dynamicdata.DynamicDataReader (p. 851).

These objects will produce and/or consume data (of type **com.rti.dds.dynamicdata.DynamicData** (p. 780)) on the **com.rti.dds.topic.Topic** (p. 1545). You can create these objects directly from the **com.rti.dds.domain.DomainParticipant** (p. 629) – see **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 661) and **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 666) – or by first creating intermediate **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.subscription.Subscriber** (p. 1478) objects – see **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) and **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659).

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. 629) is optional; all types are automatically unregistered when the **com.rti.dds.domain.DomainParticipant** (p. 629) itself is deleted.

6.78.2 Function Documentation

6.78.2.1 DynamicDataInfo () [inherited]

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

See also:

com.rti.dds.dynamicdata.DynamicData.get_info (p. 798)

6.78.2.2 DynamicDataInfo (int *member_count*, int *stored_size*, boolean *is_optimized_storage*) [inherited]

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

See also:

com.rti.dds.dynamicdata.DynamicData.get_info (p. 798)

6.78.2.3 DynamicDataMemberInfo () [inherited]

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. 805)

6.78.2.4 DynamicDataMemberInfo (int *member_id*, String *member_name*, boolean *member_exists*, TCKind *member_kind*, int *representation_count*, int *element_count*, TCKind *element_kind*) [inherited]

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. 805)

6.78.3 Variable Documentation

6.78.3.1 final DynamicDataProperty_t PROPERTY_DEFAULT [static, inherited]

Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 849).

Pass this object instead of your own **com.rti.dds.dynamicdata.DynamicDataProperty_t** (p. 849) 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. 849)

6.78.3.2 final DynamicDataTypeProperty_t TYPE_PROPERTY_DEFAULT [static, inherited]

Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 883).

Pass this object instead of your own **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 883) 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. 883)

6.79 Publication Module

Contains the **com.rti.dds.publication.FlowController** (p. 942), **com.rti.dds.publication.Publisher** (p. 1277), and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**

  com.rti.dds.publication.Publisher *(p. 1277) entity and associated elements*

- **Data Writers**

  com.rti.dds.publication.DataWriter *(p. 538) entity and associated elements*

- **Flow Controllers**

  <<eXtension>> (p. 270) ***com.rti.dds.publication.FlowController*** *(p. 942) and associated elements*

6.79.1 Detailed Description

Contains the **com.rti.dds.publication.FlowController** (p. 942), **com.rti.dds.publication.Publisher** (p. 1277), and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the publication side.

6.80 Publishers

com.rti.dds.publication.Publisher (p. 1277) entity and associated elements

Classes

- interface **Publisher**

  <<**interface**>> (p. 271) *A publisher is the object responsible for the actual dissemination of publications.*

- class **PublisherAdapter**

  <<**eXtension**>> (p. 270) *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. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for* ***com.rti.dds.publication.Publisher*** *(p. 1277) status.*

- class **PublisherQos**

  QoS policies supported by a ***com.rti.dds.publication.Publisher*** *(p. 1277) entity.*

- class **PublisherSeq**

  Declares IDL `sequence` < ***com.rti.dds.publication.Publisher*** *(p. 1277)* > .

Variables

- static final DataWriterQos **DATAWRITER_QOS_DEFAULT**

  Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 538) with default QoS.*

- static final DataWriterQos **DATAWRITER_QOS_USE_TOPIC_QOS** = new DataWriterQos()

  Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 538) with a combination of the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1566).*

6.80.1 Detailed Description

com.rti.dds.publication.Publisher (p. 1277) entity and associated elements

6.80.2 Variable Documentation

6.80.2.1 final DataWriterQos DATAWRITER_QOS_DEFAULT [static, inherited]

Initial value:

```
new DataWriterQos()
```

Special value for creating **com.rti.dds.publication.DataWriter** (p. 538) with default QoS.

When used in **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284), this special value is used to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) should be created with the default **com.rti.dds.publication.DataWriter** (p. 538) QoS by means of the operation get_default_datawriter_qos and using the resulting QoS to create the **com.rti.dds.publication.DataWriter** (p. 538).

When used in **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282), 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. 1282) operation had never been called.

When used in **com.rti.dds.publication.DataWriter.set_qos** (p. 543), this special value is used to indicate that the QoS of the **com.rti.dds.publication.DataWriter** (p. 538) should be changed to match the current defualt QoS set in the **com.rti.dds.publication.Publisher** (p. 1277) that the **com.rti.dds.publication.DataWriter** (p. 538) belongs to.

Note: You cannot use this value to *get* the default QoS values for a **DataWriter** (p. 538); for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 647).

See also:

com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.publication.Publisher.set_default_datawriter_qos (p. 1282)
com.rti.dds.publication.DataWriter.set_qos (p. 543)

6.80.2.2 final DataWriterQos DATAWRITER_QOS_USE_TOPIC_QOS = new DataWriterQos() [static, inherited]

Special value for creating **com.rti.dds.publication.DataWriter** (p. 538) with a combination of the default **com.rti.dds.publication.DataWriterQos** (p. 588) and the **com.rti.dds.topic.TopicQos** (p. 1566).

The use of this value is equivalent to the application obtaining the default **com.rti.dds.publication.DataWriterQos** (p. 588) and the **com.rti.dds.topic.TopicQos** (p. 1566) (by means of the operation **com.rti.dds.topic.Topic.get_qos** (p. 1548)) and then combining these two QoS using the operation **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1297) whereby any policy that is set on the **com.rti.dds.topic.TopicQos** (p. 1566) "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. 538).

This value should only be used in **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284).

See also:

com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1281)
com.rti.dds.topic.Topic.get_qos (p. 1548)
com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1297)

6.81 Data Writers

com.rti.dds.publication.DataWriter (p. 538) entity and associated elements

Classes

- interface **DataWriter**

  <<**interface**>> (p. 271) *Allows an application to set the value of the data to be published under a given* ***com.rti.dds.topic.Topic*** *(p. 1545).*

- class **DataWriterAdapter**

  <<**eXtension**>> (p. 270) *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. 270) *The status of the writer's cache.*

- interface **DataWriterListener**

  <<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for writer status.*

- class **DataWriterProtocolStatus**

  <<**eXtension**>> (p. 270) *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. 538) entity.*

- interface **FooDataWriter**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific data writer.*

- class **LivelinessLostStatus**

  StatusKind.LIVELINESS_LOST_STATUS.

- class **OfferedDeadlineMissedStatus**

  StatusKind.OFFERED_DEADLINE_MISSED_STATUS.

- class **OfferedIncompatibleQosStatus**

  StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS.

- class **PublicationMatchedStatus**

  StatusKind.PUBLICATION_MATCHED_STATUS.

- class **ReliableReaderActivityChangedStatus**

  <<**eXtension**>> (p. 270) *Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*

- class **ReliableWriterCacheChangedStatus**

  <<**eXtension**>> (p. 270) *A summary of the state of a data writer's cache of unacknowledged samples written.*

- class **ReliableWriterCacheEventCount**

  <<**eXtension**>> (p. 270) *The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.*

- class **FooDataWriter**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific data writer.*

6.81.1 Detailed Description

com.rti.dds.publication.DataWriter (p. 538) entity and associated elements

6.82 Flow Controllers

<<eXtension>> *(p. 270)* **com.rti.dds.publication.FlowController** (p. 942) and associated elements

Classes

- interface **FlowController**

  <<interface>> (p. 271) *A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***com.rti.dds.publication.DataWriter*** *(p. 538) instances are allowed to write data.*

- class **FlowControllerProperty_t**

  Determines the flow control characteristics of the ***com.rti.dds.publication.FlowController*** *(p. 942).*

- class **FlowControllerSchedulingPolicy**

  Kinds of flow controller scheduling policy.

- class **FlowControllerTokenBucketProperty_t**

  com.rti.dds.publication.FlowController *(p. 942) 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. 1310) 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. 1310) 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. 1310) that refers to the built-in on-demand flow controller.*

6.82.1 Detailed Description

<<eXtension>> *(p. 270)* **com.rti.dds.publication.FlowController** (p. 942) and associated elements

com.rti.dds.publication.FlowController (p. 942) provides the network traffic shaping capability to asynchronous **com.rti.dds.publication.DataWriter** (p. 538) instances. For use cases and advantages of publishing asnychronously, please refer to **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1308) of **com.rti.dds.publication.DataWriterQos** (p. 588).

See also:

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)
com.rti.dds.publication.DataWriterQos.publish_mode (p. 593)
com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)

6.82.2 Variable Documentation

6.82.2.1 final String DEFAULT_FLOW_CONTROLLER_NAME [static, inherited]

[default] Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1310) that refers to the built-in default flow controller.

RTI Connext provides several built-in **com.rti.dds.publication.FlowController** (p. 942) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 538). 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 **FlowController.DEFAULT_FLOW_CONTROLLER_NAME** (p. 182) 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. 538).

Essentially, this is equivalent to a user-created **com.rti.dds.publication.FlowController** (p. 942) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947) = **FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 949)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `max_tokens` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_added_per_period` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_leaked_per_period` = 0

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `period` = 1 second

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `bytes_per_token` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

See also:

com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 684)
com.rti.dds.publication.FlowController.set_property (p. 943)
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)
com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)

6.82.2.2 final String FIXED_RATE_FLOW_CONTROLLER_NAME [static, inherited]

Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1310) that refers to the built-in fixed-rate flow controller.

RTI Connext provides several **builtin** (p. 341) **com.rti.dds.publication.FlowController** (p. 942) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 538). 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 **FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 183) 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. 942) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947) = **FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 949)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `max_tokens` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_added_per_period` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_leaked_per_period` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `period` = 1 second

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `bytes_per_token` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

See also:

com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 684)
com.rti.dds.publication.FlowController.set_property (p. 943)
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)
com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)

6.82.2.3 final String ON_DEMAND_FLOW_CONTROLLER_NAME [static, inherited]

Special value of **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1310) that refers to the built-in on-demand flow controller.

RTI Connext provides several **builtin** (p. 341) **com.rti.dds.publication.FlowController** (p. 942) for use with an asynchronous **com.rti.dds.publication.DataWriter** (p. 538). 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 **FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 184) allows data to be sent only when the user calls **com.rti.dds.publication.FlowController.trigger_flow** (p. 945). With each trigger, all accumulated data since the previous trigger is sent (across all **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.publication.DataWriter** (p. 538) 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 Transport.Property_t.gather_send_buffer_count_max).

Essentially, this is equivalent to a user-created **com.rti.dds.publication.FlowController** (p. 942) with the following **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) settings:

- **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947) = **FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 949)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `max_tokens` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_added_per_period` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `tokens_leaked_per_period` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `period` = com.rti.dds.infrastructure.Duration_t.INFINITE

- **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) `bytes_per_token` = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

See also:

com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.domain.DomainParticipant.lookup_flowcontroller (p. 684)
com.rti.dds.publication.FlowController.trigger_flow (p. 945)
com.rti.dds.publication.FlowController.set_property (p. 943)
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)
com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)

6.83 Subscription Module

Contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**

  com.rti.dds.subscription.Subscriber *(p. 1478) entity and associated elements*

- **DataReaders**

  com.rti.dds.subscription.DataReader *(p. 473) entity and associated elements*

- **Data Samples**

  com.rti.dds.subscription.SampleInfo *(p. 1404),*
  com.rti.dds.subscription.SampleStateKind *(p. 1430),*
  com.rti.dds.subscription.ViewStateKind *(p. 1689),*
  com.rti.dds.subscription.InstanceStateKind *(p. 1086) and associated elements*

6.83.1 Detailed Description

Contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the subscription side.

6.83.2 Access to data samples

Data is made available to the application by the following operations on **com.rti.dds.subscription.DataReader** (p. 473) objects: com.rti.dds.topic.example.FooDataReader.read,

com.rti.dds.topic.example.FooDataReader.read_w_condition, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.take_w_condition, 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. 1404) 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. 75) QoS allow for it.

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.dds.topic.example.FooDataReader.return_loan.

See also:

Interpretation of the SampleInfo (p. 1405)

6.83.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **com.rti.dds.subscription.DataReader** (p. 473). These operations return an ordered collection of `DataSamples` consisting of a **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 473) and **com.rti.dds.subscription.Subscriber** (p. 1478), 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. 1430))
- the desired view states (any combination of **com.rti.dds.subscription.ViewStateKind** (p. 1689))
- the desired instance states (any combination of **com.rti.dds.subscription.InstanceStateKind** (p. 1086))

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. 1326) 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 true. These operations, in conjunction with **com.rti.dds.subscription.ReadCondition** (p. 1326) objects and a **com.rti.dds.infrastructure.WaitSet** (p. 1695), 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. 86) QoS must be set properly.

6.84 Subscribers

com.rti.dds.subscription.Subscriber (p. 1478) entity and associated elements

Classes

- interface **Subscriber**

  <<**interface**>> (p. 271) *A subscriber is the object responsible for actually receiving data from a* ***subscription*** *(p. 343).*

- 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. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for status about a subscriber.*

- class **SubscriberQos**

  QoS policies supported by a ***com.rti.dds.subscription.Subscriber*** *(p. 1478) entity.*

- class **SubscriberSeq**

  Declares IDL `sequence` < ***com.rti.dds.subscription.Subscriber*** *(p. 1478)* > .

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. 473) with a combination of the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1566).*

6.84.1 Detailed Description

com.rti.dds.subscription.Subscriber (p. 1478) entity and associated elements

6.84.2 Variable Documentation

6.84.2.1 final DataReaderQos DATAREADER_QOS_DEFAULT [static, inherited]

Initial value:

```
new DataReaderQos()
```

Special value for creating data reader with default QoS.

When used in **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485), this special value is used to indicate that the **com.rti.dds.subscription.DataReader** (p. 473) should be created with the default **com.rti.dds.subscription.DataReader** (p. 473) QoS by means of the operation get_default_datareader_qos and using the resulting QoS to create the **com.rti.dds.subscription.DataReader** (p. 473).

When used in **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1483), 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. 1483) operation had never been called.

When used in **com.rti.dds.subscription.DataReader.set_qos** (p. 480), this special value is used to indicate that the QoS of the **com.rti.dds.subscription.DataReader** (p. 473) should be changed to match the current default QoS set in the **com.rti.dds.subscription.Subscriber** (p. 1478) that the **com.rti.dds.subscription.DataReader** (p. 473) belongs to.

Note: You cannot use this value to *get* the default QoS values for a **DataReader** (p. 473); for this purpose, use **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 649).

See also:

com.rti.dds.subscription.Subscriber.create_datareader (p. 1485)
com.rti.dds.subscription.Subscriber.set_default_datareader_qos (p. 1483)
com.rti.dds.subscription.DataReader.set_qos (p. 480)

6.84.2.2 final DataReaderQos DATAREADER_QOS_USE_TOPIC_QOS = new DataReaderQos() [static, inherited]

Special value for creating **com.rti.dds.subscription.DataReader** (p. 473) with a combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 518) and the **com.rti.dds.topic.TopicQos** (p. 1566).

The use of this value is equivalent to the application obtaining the default **com.rti.dds.subscription.DataReaderQos** (p. 518) and the **com.rti.dds.topic.TopicQos** (p. 1566) (by means of the operation **com.rti.dds.topic.Topic.get_qos** (p. 1548)) and then combining these two QoS using the operation **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1500) whereby any policy that is set on the **com.rti.dds.topic.TopicQos** (p. 1566) "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. 473).

This value should only be used in **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485).

See also:

com.rti.dds.subscription.Subscriber.create_datareader (p. 1485)
com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1482)
com.rti.dds.topic.Topic.get_qos (p. 1548)
com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1500)

6.85 DataReaders

com.rti.dds.subscription.DataReader (p. 473) entity and associated elements

Modules

- **Read Conditions**

  com.rti.dds.subscription.ReadCondition *(p. 1326) and associated elements*

- **Query Conditions**

  com.rti.dds.subscription.QueryCondition *(p. 1324) and associated elements*

Classes

- interface **DataReader**

  <<**interface**>> (p. 271) *Allows the application to: (1) declare the data it wishes to receive (i.e. make a* ***subscription*** *(p. 343)) and (2) access the data received by the attached* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- class **DataReaderAdapter**

  <<**eXtension**>> (p. 270) *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. 270) *The status of the reader's cache.*

- interface **DataReaderListener**

  <<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for reader status.*

- class **DataReaderProtocolStatus**

  <<**eXtension**>> (p. 270) *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. 473) entity.*

- class **DataReaderSeq**

  Declares IDL `sequence <` ***com.rti.dds.subscription.DataReader*** *(p. 473)* `>` .

- interface **FooDataReader**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type-specific data reader.*

- class **LivelinessChangedStatus**

  StatusKind.LIVELINESS_CHANGED_STATUS.

- class **RequestedDeadlineMissedStatus**

  StatusKind.REQUESTED_DEADLINE_MISSED_STATUS.

- class **RequestedIncompatibleQosStatus**

  StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS.

- class **SampleLostStatus**

  StatusKind.SAMPLE_LOST_STATUS_STATUS.

- class **SampleLostStatusKind**

  Kinds of reasons why a sample was lost.

- class **SampleRejectedStatus**

  StatusKind.SAMPLE_REJECTED_STATUS.

- class **SampleRejectedStatusKind**

  Kinds of reasons for rejecting a sample.

- class **SubscriptionMatchedStatus**

  StatusKind.SUBSCRIPTION_MATCHED_STATUS.

- class **FooDataReader**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type-specific data reader.*

6.85.1 Detailed Description

com.rti.dds.subscription.DataReader (p. 473) entity and associated elements

6.86 Read Conditions

com.rti.dds.subscription.ReadCondition (p. 1326) and associated elements

Classes

- interface **ReadCondition**

  <<**interface**>> (p. 271) *Conditions specifically dedicated to read operations and attached to one* ***com.rti.dds.subscription.DataReader*** *(p. 473).*

6.86.1 Detailed Description

com.rti.dds.subscription.ReadCondition (p. 1326) and associated elements

6.87 Query Conditions

com.rti.dds.subscription.QueryCondition (p. 1324) and associated elements

Classes

- interface **QueryCondition**

  <<**interface**>> (p. 271) *These are specialised* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326) objects that allow the application to also specify a filter on the locally available data.*

6.87.1 Detailed Description

com.rti.dds.subscription.QueryCondition (p. 1324) and associated elements

6.88 Data Samples

com.rti.dds.subscription.SampleInfo (p. 1404), **com.rti.dds.subscription.SampleStateKind** (p. 1430), **com.rti.dds.subscription.ViewStateKind** (p. 1689), **com.rti.dds.subscription.InstanceStateKind** (p. 1086) and associated elements

Modules

- **Sample States**

  com.rti.dds.subscription.SampleStateKind *(p. 1430) and associated elements*

- **View States**

  com.rti.dds.subscription.ViewStateKind *(p. 1689) and associated elements*

- **Instance States**

  com.rti.dds.subscription.InstanceStateKind *(p. 1086) and associated elements*

Classes

- class **SampleInfo**

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

- class **SampleInfoSeq**

  Declares IDL `sequence` *<* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404) > .*

6.88.1 Detailed Description

com.rti.dds.subscription.SampleInfo (p. 1404), **com.rti.dds.subscription.SampleStateKind** (p. 1430), **com.rti.dds.subscription.ViewStateKind** (p. 1689), **com.rti.dds.subscription.InstanceStateKind** (p. 1086) and associated elements

6.89 Sample States

com.rti.dds.subscription.SampleStateKind (p. 1430) and associated elements

Classes

- ˆ class **SampleStateKind**

  Indicates whether or not a sample has ever been read.

Variables

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

  *Any sample state **SampleStateKind.READ_SAMPLE_STATE** (p. 1430) | **SampleStateKind.NOT_READ_SAMPLE_STATE** (p. 1431).*

6.89.1 Detailed Description

com.rti.dds.subscription.SampleStateKind (p. 1430) and associated elements

6.89.2 Variable Documentation

6.89.2.1 final int ANY_SAMPLE_STATE = 0xffff [static, inherited]

Any sample state **SampleStateKind.READ_SAMPLE_STATE** (p. 1430) | **SampleStateKind.NOT_READ_SAMPLE_STATE** (p. 1431).

6.90 View States

com.rti.dds.subscription.ViewStateKind (p. 1689) 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 **ViewStateKind.NEW_VIEW_STATE** (p. 1690) | **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690).*

6.90.1 Detailed Description

com.rti.dds.subscription.ViewStateKind (p. 1689) and associated elements

6.90.2 Variable Documentation

6.90.2.1 final int ANY_VIEW_STATE = 0xffff [static, inherited]

Any view state **ViewStateKind.NEW_VIEW_STATE** (p. 1690) | **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690).

6.91 Instance States

com.rti.dds.subscription.InstanceStateKind (p. 1086) and associated elements

Classes

- class **InstanceStateKind**

  *Indicates is the samples are from a live **com.rti.dds.publication.DataWriter** (p. 538) 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.

6.91.1 Detailed Description

com.rti.dds.subscription.InstanceStateKind (p. 1086) and associated elements

6.91.2 Variable Documentation

6.91.2.1 final int ANY_INSTANCE_STATE = 0xffff [static, inherited]

Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

6.91.2.2 final int NOT_ALIVE_INSTANCE_STATE = 0x006 [static, inherited]

Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

6.92 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

- **Conditions and WaitSets**

  com.rti.dds.infrastructure.Condition *(p. 451)* *and* ***com.rti.dds.infrastructure.WaitSet*** *(p. 1695) and related items.*

- **Time Support**

  Time and duration types and defines.

- **Entity Support**

  com.rti.dds.infrastructure.Entity *(p. 912),* ***com.rti.dds.infrastructure.Listener*** *(p. 1154) and related items.*

- **GUID Support**

  <<**eXtension**>> (p. 270) *GUID type and defines.*

- **Object Support**

  <<**eXtension**>> (p. 270) *Object related items.*

- **QoS Policies**

  Quality of Service (QoS) policies.

- **Return Codes**

  Types of return codes.

- **Sequence Number Support**

  <<**eXtension**>> (p. 270) *Sequence number type and defines.*

- **Status Kinds**

  Kinds of communication status.

- **Exception Codes**

  <<**eXtension**>> (p. 270) *Exception codes.*

- **Sequence Support**

  The ***com.rti.dds.util.Sequence*** *(p. 1432) interface allows you to work with variable-length collections of homogeneous data.*

Classes

- ˆ class **Enum**

  A superclass for all type-safe enumerated types.

6.92.1 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.

6.93 Built-in Sequences

Defines sequences of primitive data type.

Classes

- ^ class **BooleanSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *boolean* >.

- ^ class **ByteSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *byte* >.

- ^ class **CharSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *char* >.

- ^ class **DoubleSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *double* >.

- ^ class **FloatSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *float* >.

- ^ class **IntSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *int* >.

- ^ class **LongDoubleSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *com.rti.dds.infrastructure.LongDouble* >.

- ^ class **LongSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *long* >.

- ^ class **ShortSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *short* >.

- ^ class **StringSeq**

  Declares IDL `sequence` < *String* > .

- ^ class **WcharSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *char* >.

- ^ class **WstringSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *char** >.

6.93.1 Detailed Description

Defines sequences of primitive data type.

6.94 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

6.94.1 What is a Multi-channel DataWriter?

A Multi-channel **com.rti.dds.publication.DataWriter** (p. 538) is a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **com.rti.dds.publication.DataWriter** (p. 538) 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.

6.94.2 Configuration on the Writer Side

To configure a multi-channel **com.rti.dds.publication.DataWriter** (p. 538), simply define a list of all its channels in the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205).

The **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) is propagated along with discovery traffic. The value of this policy is available in builtin.PublicationBuiltinTopicData.locator_filter.

6.94.3 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. 538). If you want the **com.rti.dds.subscription.DataReader** (p. 473) to subscribe to only a subset of the channels, use a **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

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

6.94.4 Reliability with Multi-Channel DataWriters

6.94.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is set to PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS and the filters in **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) 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.

6.94.4.2 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. 1359) and/or **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 600).

The protocol parameters described in **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 571) are applied per channel, with the following exceptions:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1381) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1381): 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. 1385): 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. 1359))

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. 1359) instead of max_batches **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolic batches** (p. 600) in order to take advantage of heartbeats_per_max_samples.

6.95 Pluggable Transports

APIs related to RTI Connext pluggable transports.

Modules

- **Using Transport Plugins**

  Configuring transports used by RTI Connext.

- **Built-in Transport Plugins**

  Transport plugins delivered with RTI Connext.

6.95.1 Detailed Description

APIs related to RTI Connext pluggable transports.

6.95.2 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. 216)) 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. 1580)). 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.

6.95.3 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. 629). 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. 629). An **alias** symbolically refers to one or more transport plugins registered with the **com.rti.dds.domain.DomainParticipant** (p. 629). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 115)).

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. 625)), and for **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) entities (see **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1600)).

- the **multicast** addresses on which to receive discovery messages (see **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625)), and the multicast addresses and ports on which to receive user data (see **com.rti.dds.subscription.DataReaderQos.multicast** (p. 522)).

- the **unicast ports** used for user data (see **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1605)) on both **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) entities.

- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 56) and **NDDS_DISCOVERY_PEERS** (p. 55)).

A **com.rti.dds.domain.DomainParticipant** (p. 629) (and contained its entities) start using a transport plugin after the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled (see **com.rti.dds.infrastructure.Entity.enable** (p. 915)). 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.

6.95.4 Transport Lifecycle

A transport plugin is owned by whoever created it. Thus, if you create and register a transport plugin with a **com.rti.dds.domain.DomainParticipant** (p. 629), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 115)) and transport plugins that are loaded through the **PROPERTY** (p. 88) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 213)) 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. 629). This generally means that a user-created transport plugin instance can only be deleted after the **com.rti.dds.domain.DomainParticipant** (p. 629) with which it was registered is deleted (see **com.rti.dds.domain.DomainParticipantFactory.delete_participant** (p. 715)). Note that a transport plugin *cannot* be "unregistered" from a **com.rti.dds.domain.DomainParticipant** (p. 629).

A transport plugin instance cannot be registered with more than one **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) 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. 629) 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.

6.95.5 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 Transport.Property_t.classid) 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.

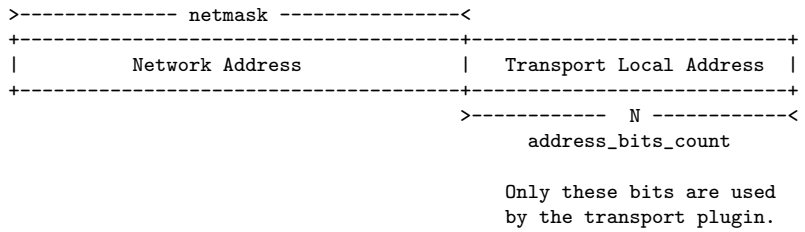
Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Connext application, provided they use different transport class IDs.

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 Transport.Property_t.address_bit_count)

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. In general, this mapping is likely to use only `N` least significant bits (LSB); these are specified by this attribute.



The remaining bits of an address using the 128-bit address representation will be considered as part of the "network address" (see **Transport Network Address** (p. 371)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher (128 - Transport.Property_t.address_bit_count) 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.

6.95.6 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 Transport.Property_t.message_size_max and Transport.Property_t.gather_send_buffer_count_max.

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.

6.95.7 Transport Network Address

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

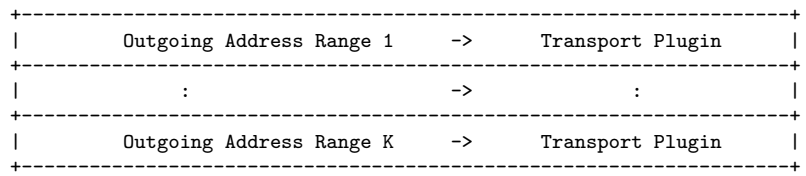
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a transport plugin via the TransportSupport.register_transport() API.

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. 629), 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).

6.95.8 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 TransportSupport.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.

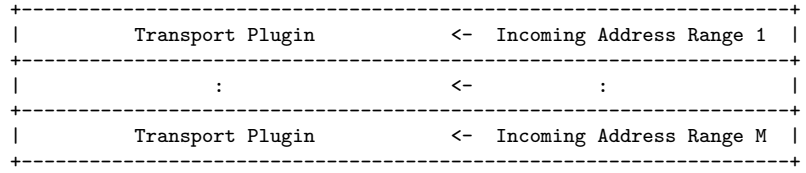


The user can set up a routing table to restrict the use of a transport plugin to send messages to selected addresses ranges.

6.95.9 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 TransportSupport.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).

6.96 Using Transport Plugins

Configuring transports used by RTI Connext.

Classes

- class **TransportSupport**

  <<**interface**>> (p. 271) *The utility class used to configure RTI Connext pluggable transports.*

6.96.1 Detailed Description

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.TransportBuiltinQosPolicy** (p. 1580) (see **Built-in Transport Plugins** (p. 216))
- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 88) QoS policy settings of **com.rti.dds.domain.DomainParticipant** (p. 629) (on UNIX, Solaris and Windows systems only) (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 213))

The lifecycle of the transport plugin is automatically managed by RTI Connext. See **Transport Lifecycle** (p. 369) for details.

6.96.2 Loading Transport Plugins through Property QoS Policy of Domain Participant

On UNIX, Solaris and Windows operating systems, 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. 88) QoS policy settings of the **com.rti.dds.domain.DomainParticipant** (p. 629). The dynamic-link library (and all the dependent libraries) need to be in the path during runtime (in **LD_LIBRARY_PATH** environment variable on Linux/Solaris systems,and in **PATH** environment variable for 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 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. 88) QoS policy of the **com.rti.dds.domain.DomainParticipant** (p. 629).

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

A transport plugin is dynamically created and registered to the **com.rti.dds.domain.DomainParticipant** (p. 629) by RTI Connext when:

- the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1490)),

whichever happens first.

Any changes to the transport plugin related properties in **PROPERTY** (p. 88) QoS policy after the transport plugin has been registered with the **com.rti.dds.domain.DomainParticipant** (p. 629) will have no effect.

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.WAN.wan1, dds.transport.DTLS.dtls1", 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 Unix/Solaris, and *.dll for Windows) that contains the transport plugin implementation. This library (and all the other dependent dynamic libraries) needs to be in the path during run time for used by RTI Connext (in the **LD_LIBRARY_PATH** environment variable on UNIX/Solaris systems, in **PATH** for Windows systems).	**YES**
<TRANSPORT_PREFIX>.create_function	Should be set to the name of the function with the prototype of 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	**YES**

6.97 Built-in Transport Plugins

Transport plugins delivered with RTI Connext.

Classes

- interface **ShmemTransport**

  *Built-in **transport** (p. 367) plug-in for inter-process communications using shared memory.*

- interface **UDPv4Transport**

  *Built-in **transport** (p. 367) plug-in using UDP/IPv4.*

- interface **UDPv6Transport**

  *Built-in **transport** (p. 367) plug-in using UDP/IPv6.*

6.97.1 Detailed Description

Transport plugins delivered with RTI Connext.

The **TRANSPORT_BUILTIN** (p. 115) 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. 1580), 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. 1580), the plugin is implicitly created and registered to the corresponding **com.rti.dds.domain.DomainParticipant** (p. 629) by RTI Connext when:

- the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**com.rti.dds.subscription.Subscriber.lookup_datareader** (p. 1490)),

whichever happens first.

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

The builtin transport plugin properties can be changed by the method TransportSupport.set_builtin_transport_property() or by using the **PROPERTY** (p. 88) QoS policy associated with the **com.rti.dds.domain.DomainParticipant** (p. 629). Builtin transport plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) always overwrite the ones specified through TransportSupport.set_builtin_transport_property(). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 88) QoS policy.

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

See also:

TransportSupport.set_builtin_transport_property()
com.rti.dds.infrastructure.PropertyQosPolicy (p. 1252)

6.98 Configuration Utilities

Utility API's independent of the DDS standard.

Classes

- class **LibraryVersion_t**

  The version of a single library shipped as part of an RTI Connext distribution.

- class **LogCategory**

  Categories of logged messages.

- class **Logger**

  <<**interface**>> (p. 271) *The singleton type used to configure RTI Connext logging.*

- class **LogPrintFormat**

  The format used to output RTI Connext diagnostic information.

- class **LogVerbosity**

  The verbosities at which RTI Connext diagnostic information is logged.

- class **Version**

  <<**interface**>> (p. 271) *The version of an RTI Connext distribution.*

6.98.1 Detailed Description

Utility API's independent of the DDS standard.

6.99 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. 219)
- **Durable Reader State** (p. 219)
- **Data Durability** (p. 220)

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

These three features can be used separately or in combination.

6.99.1 Durable Writer History

This feature allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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. 538) had been temporarily disconnected from the network and then reappeared.

See also:

Configuring Durable Writer History (p. 221)

6.99.2 Durable Reader State

This feature allows a **com.rti.dds.subscription.DataReader** (p. 473) to locally persists its state and remember the data it has already received. When an application restarts, each **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) before the restart will be suppressed so it is not sent over the network.

6.99.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 65) QoS lets an application configure a **com.rti.dds.publication.DataWriter** (p. 538) such that the information written by the **com.rti.dds.publication.DataWriter** (p. 538) survives beyond the lifetime of the **com.rti.dds.publication.DataWriter** (p. 538). In this manner, a late-joining **com.rti.dds.subscription.DataReader** (p. 473) can subscribe and receive the information even after the **com.rti.dds.publication.DataWriter** (p. 538) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

6.99.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) or **com.rti.dds.subscription.DataReader** (p. 473); 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. 538).

Several **com.rti.dds.publication.DataWriter** (p. 538) entities can be configured with the same virtual GUID. If each of these **com.rti.dds.publication.DataWriter** (p. 538) entities publishes a sample with sequence number '0', the sample will only be received once by the **com.rti.dds.subscription.DataReader** (p. 473) entities subscribing to the content published by the **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) will be persisted in a database table with a name generated from the virtual GUID of the **com.rti.dds.publication.DataWriter** (p. 538). If the **com.rti.dds.publication.DataWriter** (p. 538) 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. 473) will be persisted in a database table whose name is generated from the **com.rti.dds.subscription.DataReader** (p. 473) virtual GUID

A **com.rti.dds.publication.DataWriter** (p. 538)'s virtual GUID can be configured using **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.virtual_guid** (p. 572). Similarly, a **com.rti.dds.subscription.DataReader** (p. 473)'s virtual GUID can be configured using **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 505).

The builtin.PublicationBuiltinTopicData and builtin.SubscriptionBuiltinTopicData 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. 572) **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_guid** (p. 505).

6.99.5 Configuring Durable Writer History

To configure a **com.rti.dds.publication.DataWriter** (p. 538) to have durable writer history, use the **PROPERTY** (p. 88) QoS policy associated with the **com.rti.dds.publication.DataWriter** (p. 538) or the **com.rti.dds.domain.DomainParticipant** (p. 629).

Properties defined for the **com.rti.dds.domain.DomainParticipant** (p. 629) will be applied to all the **com.rti.dds.publication.DataWriter** (p. 538) objects belonging to the **com.rti.dds.domain.DomainParticipant** (p. 629), unless the property is overwritten by the **com.rti.dds.publication.DataWriter** (p. 538).

See also:

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1252)

The following table lists the supported durable writer history properties.

6.99.6 Configuring Durable Reader State

To configure a **com.rti.dds.subscription.DataReader** (p. 473) with durable reader state, use the **PROPERTY** (p. 88) QoS policy associated with the **com.rti.dds.subscription.DataReader** (p. 473) or **com.rti.dds.domain.DomainParticipant** (p. 629).

A property defined in the **com.rti.dds.domain.DomainParticipant** (p. 629) will be applicable to all the **com.rti.dds.subscription.DataReader** (p. 473) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 629) unless it is overwritten by the **com.rti.dds.subscription.DataReader** (p. 473).

See also:

com.rti.dds.infrastructure.PropertyQosPolicy (p. 1252)

The following table lists the supported durable reader state properties.

6.99.7 Configuring Data Durability

RTI Connext implements DurabilityQosPolicyKind.TRANSIENT_-DURABILITY_QOS and 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 online documentation.

See also:

DURABILITY (p. 65)

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.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.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 UNIX/Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems).
dds.data_writer.history.odbc_plugin.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.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.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. 538) will create its own database connection. Default: 0 (false)
dds.data_writer.history.odbc_plugin.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.ResourceLimitsQosPolicy** (p. 1356) are used to configure the maximum number of samples and instances that can be stored in the relational database. Default:

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. 473) 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 UNIX/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. 473) state must be restored or not once the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) will restore its previous state from the database. Default: 1
dds.data_reader.state.checkpoint_frequency	This property controls how often the reader state is stored in the database. A value of N means to store the state once every N samples. A high frequency will provide better performance. However, if the reader

6.100 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

6.100.1 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 when the following operations are called:

- **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 714)
- **com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile** (p. 730)
- **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 717)
- **com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos** (p. 716)
- **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 721)
- **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 722)
- **com.rti.dds.domain.DomainParticipantFactory.get_participant_qos_from_profile** (p. 723)
- **com.rti.dds.domain.DomainParticipantFactory.get_topic_qos_from_profile** (p. 728)
- **com.rti.dds.domain.DomainParticipantFactory.get_topic_qos_from_profile_w_topic_name** (p. 728)
- **com.rti.dds.domain.DomainParticipantFactory.get_publisher_qos_from_profile** (p. 724)
- **com.rti.dds.domain.DomainParticipantFactory.get_subscriber_qos_from_profile** (p. 724)
- **com.rti.dds.domain.DomainParticipantFactory.get_datawriter_qos_from_profile** (p. 725)
- **com.rti.dds.domain.DomainParticipantFactory.get_datawriter_qos_from_profile_w_topic_name** (p. 726)

- **com.rti.dds.domain.DomainParticipantFactory.get_datareader_qos_from_profile** (p. 726)
- **com.rti.dds.domain.DomainParticipantFactory.get_datareader_qos_from_profile_w_topic_name** (p. 727)
- **com.rti.dds.domain.DomainParticipantFactory.get_qos_profile_libraries** (p. 729)
- **com.rti.dds.domain.DomainParticipantFactory.get_qos_profiles** (p. 729)
- **com.rti.dds.domain.DomainParticipantFactory.load_profiles** (p. 720)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **com.rti.dds.domain.DomainParticipantFactory.set_qos** (p. 719)
- **com.rti.dds.domain.DomainParticipantFactory.reload_profiles** (p. 720)

The **com.rti.dds.domain.DomainParticipantFactory.unload_profiles()** (p. 720) 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/qos_profiles_4.5f/xml is loaded if it exists and **com.rti.dds.infrastructure.ProfileQosPolicy.ignore_resource_profile** (p. 1249) in **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1247) is set to false (first to be loaded). An example file, `NDDS_QOS_PROFILES.example.xml`, is available for reference.
- The URL groups separated by semicolons referenced by the enviroment variable NDDS_QOS_PROFILES are loaded if they exist and **com.rti.dds.infrastructure.ProfileQosPolicy.ignore_environment_profile** (p. 1248) in **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1247) is set to 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. 1248) in **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1247) is set to false.

- The URL groups referenced by **com.rti.dds.infrastructure.ProfileQosPolicy.url_profile** (p. 1248) in **com.rti.dds.infrastructure.ProfileQosPolicy** (p. 1247) will be loaded if specified.

- The sequence of XML strings referenced by **com.rti.dds.infrastructure.ProfileQosPolicy.string_profile** (p. 1248) will be loaded if specified (last to be loaded).

The above methods can be combined together.

6.100.2 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

6.100.2.1 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.

6.100.2.2 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 enviroment variable are loaded if they exist and **com.rti.dds.infrastructure.ProfileQosPolicy.ignore_environment_profile** (p. 1248) is set to false

For more information on XML Configuration, refer to the User's Manual.

6.101 Publication Example

A data publication example.

6.101.1 A typical publication example

Prep

- **Create user data types using rtiddsgen** (p. 290)

Set up

- **Get the factory** (p. 231)
- **Set up participant** (p. 231)
- **Set up publisher** (p. 239)
- **Register user data type(s)** (p. 233)
- **Set up topic(s)** (p. 233)
- **Set up data writer(s)** (p. 240)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 249)

Send data

- **Send data** (p. 241)

Tear down

- **Tear down data writer(s)** (p. 241)
- **Tear down topic(s)** (p. 233)
- **Tear down publisher** (p. 239)
- **Tear down participant** (p. 232)

6.102 Subscription Example

A data subscription example.

6.102.1 A typical subscription example

Prep

- **Create user data types using rtiddsgen** (p. 290)

Set up

- **Get the factory** (p. 231)
- **Set up participant** (p. 231)
- **Set up subscriber** (p. 242)
- **Register user data type(s)** (p. 233)
- **Set up topic(s)** (p. 233)
- **Set up data reader(s)** (p. 245)
- **Set up data reader** (p. 246) **OR Set up subscriber** (p. 242) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 249)

Receive data

- Access received data either **via a reader** (p. 246) **OR via a subscriber** (p. 243) (possibly in a **ordered or coherent** (p. 244) manner)

Tear down

- **Tear down data reader(s)** (p. 248)
- **Tear down topic(s)** (p. 233)
- **Tear down subscriber** (p. 244)
- **Tear down participant** (p. 232)

6.103 Participant Use Cases

Working with domain partcipants. Working with domain participants.

6.103.1 Turning off auto-enable of newly created participant(s)

- **Get the factory** (p. 231)

- Change the value of the **ENTITY_FACTORY** (p. 69) for the **com.rti.dds.domain.DomainParticipantFactory** (p. 708)

```
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");
}
```

6.103.2 Getting the factory

- Get the DDSDomainParticipantFactory instance:

```
DomainParticipantFactory factory = null;

factory = DomainParticipantFactory.get_instance();
```

6.103.3 Setting up a participant

- **Get the factory** (p. 231)

- 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
   }
```

6.103.4 Tearing down a participant

ˆ **Get the factory** (p. 231)

ˆ Delete DDSDomainParticipant:

```
try {
    factory.delete_participant(participant);
} catch (RETCODE_ERROR err) {
    // unable to delete
}
```

6.104 Topic Use Cases

Working with topics.

6.104.1 Registering a user data type

- ^ **Set up participant** (p. 231)
- ^ Register user data type of type `Foo` under the name "My_Type"

```
String type_name = "My_Type";

FooTypeSupport.register_type(participant, type_name);
```

6.104.2 Setting up a topic

- ^ **Set up participant** (p. 231)
- ^ **Ensure user data type is registered** (p. 233)
- ^ Create a **com.rti.dds.topic.Topic** (p. 1545) 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
}
```

6.104.3 Tearing down a topic

- ^ Delete Topic:

```
try {
    participant.delete_topic(topic);
```

```
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.105 FlowController Use Cases

Working with flow controllers.

6.105.1 Creating a flow controller

- ^ **Set up participant** (p. 231)

- ^ 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
}
```

6.105.2 Flow controlling a data writer

- ^ **Set up participant** (p. 231)

- ^ **Create flow controller** (p. 235)

- ^ 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
}
```

6.105.3 Using the built-in flow controllers

RTI Connext provides several built-in flow controllers.

The FlowController.DEFAULT_FLOW_CONTROLLER_NAME built-in flow controller provides the basic asynchronous writer behavior. When calling com.rti.dds.topic.example.FooDataWriter.write, the call signals the **com.rti.dds.publication.Publisher** (p. 1277) asynchronous publishing thread (**com.rti.dds.publication.PublisherQos.asynchronous_publisher** (p. 1304)) to send the actual data. As with any PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 538), the com.rti.dds.topic.example.FooDataWriter.write call returns immediately afterwards. The data is sent immediately in the context of the **com.rti.dds.publication.Publisher** (p. 1277) asynchronous publishing thread.

When using the FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME flow controller, data is also sent in the context of the **com.rti.dds.publication.Publisher** (p. 1277) asynchronous publishing thread, but at a regular fixed interval. The thread accumulates samples from different **com.rti.dds.publication.DataWriter** (p. 538) instances and generates data on the wire only once per **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 953).

In contrast, the FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME flow controller permits flow only when **com.rti.dds.publication.FlowController.trigger_flow** (p. 945) is called. The data is still sent in the context of the **com.rti.dds.publication.Publisher** (p. 1277) asynchronous publishing thread. The thread accumulates samples from different **com.rti.dds.publication.DataWriter** (p. 538) instances

(across any **com.rti.dds.publication.Publisher** (p. 1277)) and sends all data since the previous trigger.

The properties of the built-in **com.rti.dds.publication.FlowController** (p. 942) instances can be adjusted.

- **Set up participant** (p. 231)
- 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. 235)

6.105.4 Shaping the network traffic for a particular transport

- **Set up participant** (p. 231)
- **Create the transports** (p. 255)
- **Create a separate flow controller for each transport** (p. 235)
- Configure **com.rti.dds.publication.DataWriter** (p. 538) instances to only use a single transport

- **Associate all data writers using the same transport to the corresponding flow controller** (p. 235)

- For each transport, the corresponding flow controller limits the network traffic based on the token bucket properties

6.105.5 Coalescing multiple samples in a single network packet

- **Set up participant** (p. 231)

- **Create a flow controller with a desired token bucket period** (p. 235)

- **Associate the data writer with the flow controller** (p. 235)

- 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).

6.106 Publisher Use Cases

Working with publishers.

6.106.1 Setting up a publisher

- **Set up participant** (p. 231)

- 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
}
```

6.106.2 Tearing down a publisher

- Delete DDSPublisher:

```
try {
    participant.delete_publisher(publisher);
} catch (RETCODE_ERROR err) {
    // respond to exception
}
```

6.107 DataWriter Use Cases

Working with data writers.

6.107.1 Setting up a data writer

- **Set up publisher** (p. 239)
- **Set up a topic** (p. 233)
- 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
}
```

6.107.2 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
}
```

6.107.3 Sending data

- **Set up data writer** (p. 240)

- **Register instance** (p. 240)

- 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
}
```

6.107.4 Tearing down a data writer

- Delete DataWriter:

```
try {
    publisher.delete_datawriter(writer);
} catch (RETCODE_ERR err) {
    // ... check for cause of failure
}
```

6.108 Subscriber Use Cases

Working with subscribers.

6.108.1 Setting up a subscriber

- **Set up participant** (p. 231)

- 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
}
```

6.108.2 Set up subscriber to access received data

- **Set up subscriber** (p. 242)

- 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. 250)
  - The processing to handle the status change is done in the **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1505) method of the attached listener.
  - Typical processing will **access the received data** (p. 243), either in arbitrary order or in a **coherent and ordered manner** (p. 244).

- **Enable DATA_ON_READERS_STATUS for the StatusCondition associated with the subscriber** (p. 251)

- The processing to handle the status change is done **when the subscriber's attached status condition is triggered** (p. 253) and the DATA_ON_READERS_STATUS status on the subscriber is changed.
- Typical processing will **access the received data** (p. 243), either in an arbitrary order or in a **coherent and ordered manner** (p. 244).

6.108.3 Access received data via a subscriber

ˆ **Ensure subscriber is set up to access received data** (p. 242)

ˆ 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
}
```

ˆ Upon successfully getting the list of readers with data, process the data readers to either:

- **Read the data in each reader** (p. 247), **OR**
- **Take the data in each reader** (p. 246)

If the intent is to access the data **coherently or in order** (p. 244), 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
}
```

ˆ **Alternatively**, call **com.rti.dds.subscription.Subscriber.notify_datareaders** (p. 1493) to invoke the DataReaderListener for each of the data readers.

```
subscriber.notify_datareaders();
```

6.108.4 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. 1237) attached to a **com.rti.dds.subscription.Subscriber** (p. 1478):

- **Ensure subscriber is set up to access received data** (p. 242)
- 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. 243)
- Indicate that the data access via the subscriber is done:

```
subscriber.end_access();
```

6.108.5 Tearing down a subscriber

- Delete Subscriber:

```
try {
    participant.delete_subscriber(subscriber);
} catch (RETCODE_ERROR err) {
    // handle exception
}
```

6.109 DataReader Use Cases

Working with data readers.

6.109.1 Setting up a data reader

- **Set up subscriber** (p. 242)

- **Set up a topic** (p. 233)

- 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
}
```

6.109.2 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
}
```

6.109.3 Set up reader to access received data

- **Set up data reader** (p. 245)

- 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. 250)
  - The processing to handle the status change is done in the **com.rti.dds.subscription.DataReaderListener.on_data_available** (p. 503) method of the attached listener.
  - Typical processing will **access the received data** (p. 246).

- **Enable DATA_AVAILABLE_STATUS for the StatusCondition associated with the data reader** (p. 251)
  - The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 253) and the DATA_AVAILABLE_STATUS status on the data reader is changed.
  - Typical processing will **access the received data** (p. 246).

6.109.4 Access received data via a reader

- **Ensure reader is set up to access received data** (p. 246)

- Access the received data, by either:
  - **Taking the received data in the reader** (p. 246), **OR**
  - **Reading the received data in the reader** (p. 247)

6.109.5 Taking data

- **Ensure reader is set up to access received data** (p. 246)

- 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);
```

6.109.6 Reading data

^ **Ensure reader is set up to access received data** (p. 246)

^ 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);
```

6.109.7 Tearing down a data reader

- Delete DDSDataReader:

```
try {
    subscriber.delete_datareader(reader);
} catch (RETCODE_ERROR err) {
    // ... check for cause of failure
}
```

6.110 Entity Use Cases

Working with entities.

6.110.1 Enabling an entity

- To enable an **com.rti.dds.infrastructure.Entity** (p. 912)

```
try {
    entity.enable();
} catch (RETCODE_ERROR err) {
    System.out.println(
        "*** Error: failed to enable entity");
}
```

6.110.2 Checking if a status changed on an entity.

- Given an **com.rti.dds.infrastructure.Entity** (p. 912) and a **com.rti.dds.infrastructure.StatusKind** (p. 1455) 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. 107) for details].

```
if ((status_changes_list & status_kind) != 0) {
    return true;  /* ... YES, status_kind changed ... */
} else {
    return false;  /* ... NO, status_kind did NOT change ... */
}
```

6.110.3 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. 914)

```
try {
    entity.get_qos(qos);
} catch (RETCODE_ERROR err) {
    System.out.println("***Error: failed to get qos\n");
}
```

ˆ Change the desired qos policy fields

```
/* Change the desired qos policies */
/* qos.policy.field = ... */
```

ˆ Set the qos using **set_qos (abstract)** (p. 913).

```
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");
}
```

6.110.4 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.

ˆ User defines entity listener methods

```
/* ... methods defined by EntityListener ... */
public class MyEntityListener implements Listener {
    // ... methods defined by EntityListener ...
}
```

ˆ Get an entity's listener using **get_listener (abstract)** (p. 915)

```
entity_listener = entity.get_listener();
```

ˆ 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. 914). 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
}
```

6.110.5 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
}
```

6.111 Waitset Use Cases

Using wait-sets and conditions.

6.111.1 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);
```

6.111.2 Waiting for condition(s) to trigger

- **Set up a wait-set** (p. 253)

- 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;
}
```

6.111.3 Tearing down a wait-set

^ Delete the wait-set

```
waitset.delete();
waitset = null;
// let the wait set be garbage collected
```

6.112 Transport Use Cases

Working with pluggable transports.

6.112.1 Changing the automatically registered built-in transports

- The TransportBuiltinKind.MASK_DEFAULT specifies the transport plugins that will be automatically registered with a newly created **com.rti.dds.domain.DomainParticipant** (p. 629) by default.

- This default can be changed by changing the value of the value of **TRANSPORT_BUILTIN** (p. 115) Qos Policy on the **com.rti.dds.domain.DomainParticipant** (p. 629)

- To change the **com.rti.dds.domain.DomainParticipantQos.transport_builtin** (p. 738) Qos Policy:

```
DomainParticipantQos participant_qos = new DomainParticipantQos();

factory.get_default_participant_qos(participant_qos);

participant_qos.transport_builtin.mask = TransportBuiltinKind.SHMEM |
                                         TransportBuiltinKind.UDPv4;
```

6.112.2 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. 708) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 231)

- Get the property of the desired builtin transport plugin, say UDPv4Transport

```
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 UDPv4Transport 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.

```
/* Increase the UDPv4 maximum message size to 64K (large messages). */
property.message_size_max        =  65535;
property.recv_socket_buffer_size =  65535;
property.send_socket_buffer_size =  65535;
```

- Set the property of the desired builtin transport plugin, say UDPv4Transport

```
TransportSupport.set_builtin_transport_property(participant, property);
```

- **Enable the participant** (p. 249) to turn on communications with other participants in the domain using the new properties for the automatically registered builtin transport plugins.

6.113 Filter Use Cases

Working with data filters.

6.113.1 Introduction

RTI Connext supports filtering data either during the exchange from **com.rti.dds.publication.DataWriter** (p. 538) to **com.rti.dds.subscription.DataReader** (p. 473), or after the data has been stored at the **com.rti.dds.subscription.DataReader** (p. 473).

Filtering during the exchange process is performed by a **com.rti.dds.topic.ContentFilteredTopic** (p. 458), which is created by the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) is performed by creating a **com.rti.dds.subscription.QueryCondition** (p. 1324), 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.dds.topic.example.FooDataReader.read_w_condition or com.rti.dds.topic.example.FooDataReader.take_w_condition functions. (Conditions may also be used with the APIs com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition and com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition.)

Filtering may be performed on any topic, either keyed or un-keyed, except the **Built-in Topics** (p. 153). 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. 170) API.

RTI Connext contains built in support for filtering using SQL syntax, described in the **Queries and Filters Syntax** (p. 278) module.

6.113.1.1 Overview of ContentFilteredTopic

Each **com.rti.dds.topic.ContentFilteredTopic** (p. 458) is created based on an existing **com.rti.dds.topic.Topic** (p. 1545). The **com.rti.dds.topic.Topic** (p. 1545) specifies the **field_names** and **field_types** of the data contained within the topic. The **com.rti.dds.topic.ContentFilteredTopic** (p. 458), by means of its `filter_expression` and `expression_parameters`, futher specifies the *values* of the data which the **com.rti.dds.subscription.DataReader** (p. 473) wishes to receive.

Custom filters may also be constructed and utilized as described in the **Creating Custom Content Filters** (p. 263) module.

Once the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) has been created, a **com.rti.dds.subscription.DataReader** (p. 473) can be created using the filtered topic. The filter's characteristics are exchanged between the **com.rti.dds.subscription.DataReader** (p. 473) and any matching **com.rti.dds.publication.DataWriter** (p. 538) during the discovery processs.

If the **com.rti.dds.publication.DataWriter** (p. 538) allows (by **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_remote_reader_filters** (p. 600)) and the number of filtered **com.rti.dds.subscription.DataReader** (p. 473) is less than or equal to 32, and the **com.rti.dds.subscription.DataReader** (p. 473) 's **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1590) is empty, then the **com.rti.dds.publication.DataWriter** (p. 538) will performing filtering and send to the **com.rti.dds.subscription.DataReader** (p. 473) only those samples that meet the filtering criteria.

If disallowed by the **com.rti.dds.publication.DataWriter** (p. 538), or if more than 32 **com.rti.dds.subscription.DataReader** (p. 473) require filtering, or the **com.rti.dds.subscription.DataReader** (p. 473) has set the **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1590), then the **com.rti.dds.publication.DataWriter** (p. 538) sends all samples to the **com.rti.dds.subscription.DataReader** (p. 473), and the **com.rti.dds.subscription.DataReader** (p. 473) discards any samples that do not meet the filtering criteria.

Although the `filter_expression` cannot be changed once the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) has been created, the `expression_parameters` can be modified using **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 460). Any changes made to the filtering criteria by means of **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 460), will be conveyed to any connected **com.rti.dds.publication.DataWriter** (p. 538). 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. 473) connects to a **com.rti.dds.publication.DataWriter** (p. 538) that *re-sends* its data, the re-sent samples will be subjected to the new filtering criteria.

6.113.1.2 Overview of QueryCondition

com.rti.dds.subscription.QueryCondition (p. 1324) combine aspects of the content filtering capabilities of **com.rti.dds.topic.ContentFilteredTopic** (p. 458) with state filter-

ing capabilities of **com.rti.dds.subscription.ReadCondition** (p. 1326) to create a reconfigurable means of filtering or searching data in the **com.rti.dds.subscription.DataReader** (p. 473) queue.

com.rti.dds.subscription.QueryCondition (p. 1324) may be created on a disabled **com.rti.dds.subscription.DataReader** (p. 473), or after the **com.rti.dds.subscription.DataReader** (p. 473) has been enabled. If the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1324) filter criteria at the time that the **com.rti.dds.subscription.QueryCondition** (p. 1324) is created. (Note that an exclusive lock is held on the **com.rti.dds.subscription.DataReader** (p. 473) sample queue for the duration of the **com.rti.dds.subscription.QueryCondition** (p. 1324) creation).

Once created, incoming samples are filtered against all **com.rti.dds.subscription.QueryCondition** (p. 1324) filter criteria at the time of their arrival and storage into the **com.rti.dds.subscription.DataReader** (p. 473) queue.

The number of **com.rti.dds.subscription.QueryCondition** (p. 1324) filters that an individual **com.rti.dds.subscription.DataReader** (p. 473) may create is set by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_query_condition_filters** (p. 535), to an upper maximum of 32.

6.113.2 Filtering with ContentFilteredTopic

- **Set up subscriber** (p. 242)

- **Set up a topic** (p. 233)

- 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. 458) is exactly the same as normal reads or takes, as decribed in **DataReader Use Cases** (p. 245).

ˆ Changing filter crieria using set_expression_parameters:

```
cft_parameters.set(0, "5");
cft_parameters.set(1, "9");
cft.set_expression_parameters(cft_parameters);
```

6.113.3 Filtering with Query Conditions

ˆ Given a data reader of type Foo

```
FooDataReader reader = ...;
```

ˆ Creating a QueryCondition

```
QueryCondition queryCondition = null;
String qc_param_list[] = {"1", "100"};

StringSeq qc_parameters = new StringSeq(java.util.Arrays.asList(cft_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;
}
```

ˆ Reading matching samples with a **com.rti.dds.subscription.QueryCondition** (p. 1324)

```
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. 1325) is used similarly to **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 460), and the same coding techniques can be used.

- Any **com.rti.dds.subscription.QueryCondition** (p. 1324) that have been created must be deleted before the **com.rti.dds.subscription.DataReader** (p. 473) can be deleted. This can be done using **com.rti.dds.subscription.DataReader.delete_contained_entities** (p. 489) or manually as in:

```
retcode = reader.delete_readcondition(queryCondition);
```

6.113.4 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 field only filters, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), the filter results are retrieved from 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. 473), for either **com.rti.dds.topic.ContentFilteredTopic** (p. 458) or **com.rti.dds.subscription.QueryCondition** (p. 1324). This does *not*

apply to filtering perfomed for **com.rti.dds.topic.ContentFilteredTopic** (p. 458) by the **com.rti.dds.publication.DataWriter** (p. 538).

6.114 Creating Custom Content Filters

Working with custom content filters.

6.114.1 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. 263)
- **Example Using C format strings** (p. 264)

6.114.2 The Custom Content Filter API

A custom content filter is created by calling the **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 698) function with a **com.rti.dds.topic.ContentFilter** (p. 454) that contains a **compile**, an **evaluate** function and a **finalize** function. **com.rti.dds.topic.ContentFilteredTopic** (p. 458) can be created with **com.rti.dds.domain.DomainParticipant.create_contentfilteredtopic_with_filter** (p. 675) 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. 458) (the function called is shown in parenthesis).

- When a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) is created (**compile** (p. 264))
- When the filter parameters are changed on the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) (**compile** (p. 264)) with **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 460)

- When a sample is filtered (**evaluate** (p. 264)). This function is called by the RTI Connext core with a de-serialized sample
- When a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) is deleted (**finalize** (p. 264))

6.114.2.1 The compile function

The **compile** (p. 264) 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. 455) for details.

6.114.2.2 The evaluate function

The **evaluate** (p. 265) 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. 456) for details.

6.114.2.3 The finalize function

The **finalize** (p. 265) 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. 457) for details.

6.114.3 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.

6.114.3.1 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. 264) 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);
}
```

6.114.3.2 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. 264) function.

```
public boolean evaluate(
    Object compile_data, Object sample) {

    String parameter = (String)compile_data;
    int x;

    Foo foo_sample = (Foo)sample;

    x = Integer.parseInt(parameter);

    return (foo_sample.x > x ? false : true);
}
```

6.114.3.3 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. 264) function.

```
public void finalize(
    Object compile_data) {
    /* nothing to do since no resource are allocated */
}
```

6.114.3.4 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);
```

6.114.3.5 Unregistering the Filter

When the filter is no longer needed, it can be unregistered from RTI Connext:

```
participant.unregister_contentfilter("MyCustomFilter");
```

6.115 Large Data Use Cases

Working with large data types.

6.115.1 Introduction

RTI Connext supports data types whose size exceeds the maximum message size of the underlying transports. A **com.rti.dds.publication.DataWriter** (p. 538) 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. 473) 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. 1359) and HistoryQosPolicyKind.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.

6.115.2 Writing Large Data

In order to use the large data feature with the ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS setting, the **com.rti.dds.publication.DataWriter** (p. 538) must be configured as an asynchronous writer (PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS) with associated **com.rti.dds.publication.FlowController** (p. 942).

While the use of an asynchronous writer and flow controller is optional when using the ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS 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. 240)
- **Add flow control** (p. 235)

6.115.3 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. 524) allows tuning the resources available to the **com.rti.dds.subscription.DataReader** (p. 473) for reassembling fragmented large data.

- **Set up reader** (p. 245)

6.116 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 **DDS API Reference** (p. 272) 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. 229) and **Subscription Example** (p. 230) 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** (p. 290).

- To integrate similar code into your own application and build system, you will likely need to refer to the `Platform Notes`.

6.117 Conventions

This section describes the conventions used in the API documentation.

6.117.1 Unsupported Features

[**Not supported (optional)**] This note means that the optional feature from the DDS specification is not supported in the current release.

6.117.2 API Documentation Terms

In the API documentation, the term module refers to a logical grouping of documentation and elements in the API.

6.117.3 Stereotypes

Commonly used stereotypes in the API documentation include the following.

6.117.3.1 Extensions

- ˆ *<<**eXtension**>>* *(p. 270)*
  - 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.

6.117.3.2 Experimental

- ˆ *<<**experimental**>>* *(p. 270)*
  - 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.

6.117.3.3 Types

ˆ *<<**interface**>>* *(p. 271)*

- 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.

ˆ *<<**generic**>>* *(p. 271)*

- 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.

ˆ *<<**singleton**>>* *(p. 271)*

- Singleton class. There is a single instance of the class.
- Generally acccessed via a `get_instance()` static method.

6.117.3.4 Method Parameters

ˆ *<<**in**>>* *(p. 271)*

- An *input* parameter.

ˆ *<<**out**>>* *(p. 271)*

- An *output* parameter.

ˆ *<<**inout**>>* *(p. 271)*

- An *input* and *output* parameter.

6.118 DDS API Reference

RTI Connext modules following the DDS module definitions.

Modules

ˆ Domain Module

*Contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up RTI Connext.*

ˆ Topic Module

*Contains the **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) classes, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.*

ˆ Publication Module

*Contains the **com.rti.dds.publication.FlowController** (p. 942), **com.rti.dds.publication.Publisher** (p. 1277), and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the publication side.*

ˆ Subscription Module

*Contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the subscription side.*

ˆ 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.

ˆ Queries and Filters Syntax

6.118.1 Detailed Description

RTI Connext modules following the DDS module definitions.

6.118.2 Overview

Information flows with the aid of the following constructs: **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.publication.DataWriter** (p. 538) on the sending side, **com.rti.dds.subscription.Subscriber** (p. 1478) and **com.rti.dds.subscription.DataReader** (p. 473) on the receiving side.

- A **com.rti.dds.publication.Publisher** (p. 1277) 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. 538) object is dedicated to one application data type) accessor to a publisher. A **com.rti.dds.publication.DataWriter** (p. 538) 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. 1478) 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. 1545) 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. 1545) 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. 538) associated with that Topic and the QoS of the **com.rti.dds.publication.Publisher** (p. 1277) associated to the **com.rti.dds.publication.DataWriter** (p. 538) control the behavior on the publisher's side, while the corresponding **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.subscription.Subscriber** (p. 1478) 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. 1277) (or reuse an already created one) and a **com.rti.dds.publication.DataWriter** (p. 538) 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. 1478) (or reuse an already created one) and a **com.rti.dds.subscription.DataReader** (p. 473) to define the subscription.

6.118.3 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. 912) supports its own specialized kind of QoS policies (see **QoS Policies** (p. 90)).

- Accepting a **com.rti.dds.infrastructure.Listener** (p. 1154); 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. 912) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 106)).

  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. 1452) (and a set of **com.rti.dds.subscription.ReadCondition** (p. 1326) ob-

jects for the **com.rti.dds.subscription.DataReader** (p. 473)); conditions (in conjunction with **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. 629). 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. 628) 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. 538) objects attached to the same **com.rti.dds.publication.Publisher** (p. 1277) and retrieved among **com.rti.dds.subscription.DataReader** (p. 473) objects attached to the same **com.rti.dds.subscription.Subscriber** (p. 1478).

By definition, a **com.rti.dds.topic.Topic** (p. 1545) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **com.rti.dds.topic.Topic** (p. 1545) 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. 1545) 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. 629).

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. 538) (this, in turn, will trigger its **com.rti.dds.publication.Publisher** (p. 1277)).

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. 1154) is used to provide a callback for synchronous access and a **com.rti.dds.infrastructure.WaitSet** (p. 1695) associated with one or several **com.rti.dds.infrastructure.Condition** (p. 451) 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. 106)). 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. 153) that the application can access as plain application data, using built-in data-readers.

6.118.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 200) 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. 143) contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up the Service.

- **Topic module** (p. 157) contains the **com.rti.dds.topic.Topic** (p. 1545) class, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by the application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.

- **Publication module** (p. 175) contains the **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the publication side.

- **Subscription module** (p. 186) contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the subscription side.

6.119 Queries and Filters Syntax

6.119.1 Syntax for DDS Queries and Filters

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

- The `filter_expression` in the **com.rti.dds.topic.ContentFilteredTopic** (p. 458)
- The `query_expression` in the **com.rti.dds.subscription.QueryCondition** (p. 1324)
- The `topic_expression` in the **com.rti.dds.topic.MultiTopic** (p. 1208)

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*.

6.119.2 SQL grammar in BNF

Expression ::= *FilterExpression*
| *TopicExpression*
| *QueryExpression*
.

FilterExpression ::= *Condition*
TopicExpression ::= *SelectFrom* { *Where* } ';'
QueryExpression ::= { *Condition* }{ 'ORDER BY' (**FIELDNAME** // ',') }
.

SelectFrom ::= 'SELECT' *Aggregation* 'FROM' *Selection*
.
Aggregation ::= '*'
| (*SubjectFieldSpec* // ',')

```
                .
SubjectFieldSpec ::= FIELDNAME
                |    FIELDNAME 'AS' IDENTIFIER
                |    FIELDNAME IDENTIFIER
                .
Selection   ::= TOPICNAME
           |    TOPICNAME NaturalJoin JoinItem
           .
JoinItem    ::= TOPICNAME
           |    TOPICNAME NaturalJoin JoinItem
           |    '(' TOPICNAME NaturalJoin JoinItem ')'
           .
NaturalJoin ::= 'INNER JOIN'
           |    'INNER NATURAL JOIN'
           |    'NATURAL JOIN'
           |    'NATURAL INNER JOIN'
           .
Where       ::= 'WHERE' 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
         |    PARAMETER
```

.

Note – INNER JOIN, INNER NATURAL JOIN, NATURAL JOIN, and NATURAL INNER JOIN are all aliases, in the sense that they have the same semantics. They are all supported because they all are part of the SQL standard.

6.119.3 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. 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 en or En, 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"]
```

- **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 $^{\text{th}}$ argument in the given context. Argument can only in primitive type value format. It cannot be a FIELDNAME.

Formal notation:

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

6.119.4 Type compatability in Predicate

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

	BOOLEAN-VALUE	INTEGER-VALUE	FLOAT-VALUE	CHAR-VALUE	STRING	ENUMERATED-VALUE
BOOLEAN	YES					
INTEGERVALUE		YES	YES			
FLOAT-VALUE		YES	YES			
CHAR-VALUE				YES	YES	YES
STRING				YES	YES(∗1)	YES
ENUMERATED-VALUE		YES		YES(∗2)	YES(∗2)	YES(∗3)

ˆ (∗1) See **SQL Extension: Regular Expression Matching** (p. 283)

ˆ (∗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 "MyEnum=MyString" is not allowed.

ˆ (∗3) Only for same type Enums.

6.119.5 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.

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 (not supported)

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.

6.119.6 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. 458)) or a **query_expression** (for **com.rti.dds.subscription.QueryCondition** (p. 1324)):

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

Examples of a **filter_expression** using **MATCH** (for

com.rti.dds.topic.ContentFilteredTopic (p. 458)) operator:

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

Example of a **topic_expression** (for **com.rti.dds.topic.MultiTopic** (p. 1208) [**Not supported (optional)**]):

-
```
"SELECT flight_id, x, y, z AS height FROM 'Location' NATURAL JOIN
      'FlightPlan' WHERE height < 1000 AND x <23"
```

6.120 RTI Connext API Reference

RTI Connext product specific API's.

Modules

- **Clock Selection**
  APIs related to clock selection.
- **Multi-channel DataWriters**
  APIs related to Multi-channel DataWriters.
- **Pluggable Transports**
  APIs related to RTI Connext pluggable transports.
- **Configuration Utilities**
  Utility API's independent of the DDS standard.
- **Durability and Persistence**
  APIs related to RTI Connext Durability and Persistence.
- **Configuring QoS Profiles with XML**
  APIs related to XML QoS Profiles.

6.120.1 Detailed Description

RTI Connext product specific API's.

6.121 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 partcipants.
- **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.

6.121.1 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.

6.122 Programming Tools

Modules

- **rtiddsgen**

  Generates source code from data types declared in IDL, XML, XSD, or WSDL files.

- **rtiddsping**

  Sends or receives simple messages using RTI Connext.

- **rtiddsspy**

  Debugging tool which receives all RTI Connext communication.

6.123 rtiddsgen

Generates source code from data types declared in IDL, XML, XSD, or WSDL files. Generates code necessary to allocate, send, receive, and print user-defined data types.

6.123.1 Usage

```
rtiddsgen [-d <outdir>]
     [-language <C|C++|Java|C++/CLI|C#|Ada>]
             [-namespace]
             [-package <packagePrefix>]
             [-example <arch>]
             [-replace]
     [-debug]
             [-corba [client header file] [-orb <CORBA ORB>]]
             [-optimization <level of optimization>]
             [-stringSize <Unbounded strings size>]
             [-sequenceSize <Unbounded sequences size>]
             [-notypecode]
             [-ppDisable]
             [-ppPath <preprocessor executable>]
             [-ppOption <option>]
             [-D <name>[=<value>]]
             [-U <name>]
             [-I <directory>]
             [-noCopyable]
             [-use42eAlignment]
             [-enableEscapeChar]
             [-typeSequenceSuffix <Suffix>]
             [-dataReaderSuffix <Suffix>]
             [-dataWriterSuffix <Suffix>]
             [-convertToXml |
              -convertToXsd |
              -convertToWsdl |
              -convertToIdl]
             [-convertToCcl]
             [-convertToCcs]
             [-expandOctetSeq]
             [-expandCharSeq]
             [-metp]
             [-version]
             [-help]
             [-verbosity [1-3]]
             [[-inputIdl] <IDLInputFile.idl> |
       [-inputXml] <XMLInputFile.xml> |
[-inputXsd] <XSDInputFile.xsd> |
[-inputWsdl] <WSDLInputFile.wsdl>]
```

-d Specifies where to put the generated files. If omitted, the input file's directory is used.

-language Generates output for only the language specified. The default is C++.

Use of generated Ada 2005 code requires installation of RTI Ada 2005 Language Support. Please contact `support@rti.com` for more information.

-namespace Specifies the use of C++ namespaces (for C++ only).

-package Specifies a packagePrefix to use as the root package (for Java only).

-example Generates example programs and makefiles (for UNIX-based systems) or workspace and project files (for Windows systems) based on the input types description file.

The <arch> parameter specifies the architecture for the example makefiles.

For -language C/C++, valid options for <arch> are:

sparcSol2.9gcc3.2, sparcSol2.9cc5.4, sparcSol2.10gcc3.4.2, sparc64Sol2.10gcc3.4.2, i86Sol2.9gcc3.3.2, i86Sol2.10gcc3.4.4, x64Sol2.10gcc3.4.3,

x64Darwin10gcc4.2.1,

i86Linux2.6gcc3.4.3, x64Linux2.6gcc3.4.5, i86Linux2.6gcc4.1.1, x64Linux2.6gcc4.1.1, i86Linux2.6gcc4.1.2, x64Linux2.6gcc4.1.2, i86Linux2.6gcc4.2.1, i86Linux2.6gcc4.4.3, x64Linux2.6gcc4.4.3, x64Linux2.6gcc4.3.4, i86Linux2.6gcc4.4.5, x64Linux2.6gcc4.4.5, i86Linux2.6gcc3.4.6, i86RedHawk5.1gcc4.1.2, i86RedHawk5.4gcc4.2.1, x64Linux2.6gcc4.4.4, x64Linux2.6gcc4.5.1, i86Suse10.1gcc4.1.0, x64Suse10.1gcc4.1.0, cell64Linux2.6gcc4.5.1, armv7leLinux2.6gcc4.4.1,

ppc4xxFPLinux2.6gcc4.3.3, ppc7400Linux2.6gcc3.3.3, ppc85xxLinux2.6gcc4.3.2, ppc85xxWRLinux2.6gcc4.3.2,

i86Win32VS2005, x64Win64VS2005, i86Win32VS2008, x64Win64VS2008, i86Win32VS2010, x64Win64VS2010,

ppc85xxInty5.0.11.xes-p2020, mpc8349Inty5.0.11.mds8349, pentiumInty10.0.0.pcx86,

ppc7400Lynx4.0.0gcc3.2.2, ppc7400Lynx4.2.0gcc3.2.2, ppc750Lynx4.0.0gcc3.2.2, ppc7400Lynx5.0.0gcc3.4.3, i86Lynx4.0.0gcc3.2.2,

ppc604Vx5.5gcc, ppc603Vx5.5gcc, ppc604Vx6.3gcc3.4.4, ppc604Vx6.3gcc3.4.4_rtp, ppc604Vx6.5gcc3.4.4, ppc604Vx6.5gcc3.4.4_rtp, pentiumVx6.6gcc4.1.2, pentiumVx6.6gcc4.1.2_rtp, ppc405Vx6.6gcc4.1.2, ppc405Vx6.6gcc4.1.2_rtp, ppc604Vx6.6gcc4.1.2, ppc604Vx6.6gcc4.1.2_rtp, pentiumVx6.7gcc4.1.2, pentiumVx6.7gcc4.1.2_rtp, ppc604Vx6.7gcc4.1.2, ppc604Vx6.7gcc4.1.2_smp, ppc604Vx6.7gcc4.1.2_rtp, ppc604Vx6.8gcc4.1.2, ppc604Vx6.8gcc4.1.2_rtp,

pentiumVx6.8gcc4.1.2, pentiumVx6.8gcc4.1.2_rtp, ppc604Vx6.9gcc4.3.3, ppc604Vx6.9gcc4.3.3_rtp, pentiumVx6.9gcc4.3.3, pentiumVx6.9gcc4.3.3_-rtp, pentium64Vx6.9gcc4.3.3, pentium64Vx6.9gcc4.3.3_rtp, ppc604VxT2.2.2gcc3.3.2, ppc604VxT2.2.2gcc3.3.2_v6, sbc8641Vx653-2.3gcc3.3.2, simpcVx653-2.3gcc3.3.2,

p5AIX5.3xlc9.0, 64p5AIX5.3xlc9.0,

i86QNX6.4.1qcc_gpp i86QNX6.5qcc_gpp4.4.2

For -language C++/CLI and C#, valid options for <arch> are:

i86Win32dotnet2.0, x64Win64dotnet2.0, i86Win32dotnet4.0, x64Win64dotnet4.0

For -language java, valid options for <arch> are:

i86Sol2.9jdk, i86Sol2.10jdk, x64Sol2.10jdk, sparcSol2.9jdk, sparcSol2.10jdk, sparc64Sol2.10jdk, x64Darwin10gcc4.2.1jdk, i86Linux2.6gcc3.4.3jdk, x64Linux2.6gcc3.4.5jdk, i86Linux2.6gcc4.1.1jdk, x64Linux2.6gcc4.1.1jdk, i86Linux2.6gcc4.4.3jdk, x64Linux2.6gcc4.4.3jdk, i86Linux2.6gcc4.4.5jdk, x64Linux2.6gcc4.4.5jdk, i86Linux2.6gcc4.2.1jdk, x64Linux2.6gcc4.3.4jdk, i86Linux2.6gcc4.1.2jdk, x64Linux2.6gcc4.1.2jdk, i86Linux2.6gcc3.4.6jdk, i86RedHawk5.1gcc4.1.2jdk, i86RedHawk5.4gcc4.2.1jdk, i86Suse10.1gcc4.1.0jdk, x64Suse10.1gcc4.1.0jdk, i86Win32jdk, x64Win64jdk, ppc7400Lynx4.0.0gcc3.2.2jdk, ppc750Lynx4.0.0gcc3.2.2jdk, ppc7400Lynx5.0.0gcc3.4.3jdk, i86Lynx4.0.0gcc3.2.2jdk, p5AIX5.3xlc9.0jdk, 64p5AIX5.3xlc9.0jdk

For -language Ada, valid option for <arch> is i86Linux2.6gcc4.1.2

-replace Overwrites any existing output files. Warning: This removes any changes you may have made to the original files.

-debug Generates intermediate files for debugging purposes.

-corba [client header file] [-orb <CORBA ORB>] Specifies that you want to produce CORBA-compliant code.

Use [client header file] and [-orb <CORBA ORB>] for C++ only. The majority of code generated is independent of the ORB. However, for some IDL features, the code generated depends on the ORB. This version of rtiddsgen generates code compatible with ACE-TAO or JacORB. To pick the ACE_TAO version, use the -orb parameter; the default is ACE_TAO1.6.

client header file: the name of the header file for the IDL types generated by the CORBA IDL compiler. This file will be included in the rtiddsgen type header file instead of generating type definitions.

CORBA support requires the RTI CORBA Compatibility Kit, an add-on product that provides a different version of rtiddsgen. Please contact `support@rti.com` for more information.

-optimization Sets the optimization level. (Only applies to C/C++)

- 0 (default): No optimization.
- 1: Compiler generates extra code for typedefs but optimizes its use. If the type that is used is a typedef that can be resolved either to a primitive type or to another type defined in the same file, the generated code will invoke the code of the most basic type to which the typedef can be resolved, unless the most basic type is an array or a sequence. This level can be used if the generated code is not expected to be modified.
- 2: Maximum optimization. Functionally the same as level 1, but extra code for typedef is not generated. This level can be used if the typedefs are only referred by types within the same file.

-typeSequenceSuffix Assigns a suffix to the name of the implicit sequence defined for IDL types. (Only applies to CORBA)

By default, the suffix is 'Seq'. For example, given the type 'Foo' the name of the implicit sequence will be 'FooSeq'.

-dataReaderSuffix Assigns a suffix to the name of the DataReader interface. (Only applies to CORBA)

By default, the suffix is 'DataReader'. For example, given the type 'Foo' the name of the DataReader interface will be 'FooDataReader'.

-dataWriterSuffix Assigns a suffix to the name of the DataWriter interface. (Only applies to CORBA)

By default, the suffix is 'DataWriter'. For example, given the type 'Foo' the name of the DataWriter interface will be 'FooDataWriter'.

-stringSize Sets the size for unbounded strings. Default: 255 bytes.

-sequenceSize Sets the size for unbounded sequences. Default: 100 elements.

-notypecode: Disables the generation of type code information.

-ppDisable: Disables the preprocessor.

-ppPath <preprocessor executable>: Specifies the preprocessor path. If you only specify the name of an executable (not a complete path to that executable), the executable must be found in your Path.

The default value is "cpp" for non-Windows architectures, "cl.exe" for Windows architectures.

If the default preprocessor is not found in your Path and you use -ppPath to provide its full path and filename, you must also use -ppOption (described below) to set the following preprocessor options:

- If you use a non-default path for cl.exe, you also need to set:

-ppOption /nologo -ppOption /C -ppOption /E -ppOption /X

ˆ If you use a non-default path for cpp, you also need to set:

-ppOption -C

-ppOption <option>: Specifies a preprocessor option. This parameter can be used multiple times to provide the command-line options for the specified preprocessor. See -ppPath (above).

-D <name>[=<value>]: Defines preprocessor macros.

-U <name>: Cancels any previous definition of name.

-I <directory>: Adds to the list of directories to be searched for type-definition files (IDL, XML, XSD or WSDL files). Note: A type-definition file in one format cannot include a file in another format.

-noCopyable: Forces rtiddsgen to put copy logic into the corresponding TypeSupport class rather than the type itself (for Java code generation only).

This option is not compatible with the use of ndds_standalone_type.jar.

-use42eAlignment: Generates code compliant with RTI Data Distribution Service 4.2e.

If your RTI Connext application's data type uses a 'double','long long','unsigned long long', or 'long double' it will not be backwards compatible with RTI Data Distribution Service 4.2e applications unless you use the -use42eAlignment flag when generating code with rtiddsgen.

-enableEscapeChar: Enables use of the escape character '_' in IDL identifiers. With CORBA this option is always enabled.

-convertToXml: Converts the input type-description file to XML format.

-convertToIdl: Converts the input type-description file to IDL format.

-convertToXsd: Converts the input type-description file to XSD format.

-convertToWsdl: Converts the input type-description file to WSDL format.

-convertToCcl: Converts the input type-description file to CCL format.

-convertToCcs: Converts the input type-description file to CCS format.

-expandOctetSeq: When converting to CCS or CCL files, expand octet sequences. The default is to use a blob type.

-expandCharSeq: When converting to CCS or CCL files, expand char sequences. The default is to use a string type.

-metp: Generates code for the Multi-Encapsulation Type Support (METP) library.

-version: Prints the version, such as 4.5x. (Does not show 'patch' revision number.)

-help: Prints this rtiddsgen usage help.

-verbosity: rtiddsgen verbosity.

- 1: exceptions
- 2: exceptions and warnings
- 3 (default): exceptions, warnings and information

-inputIdl: Indicates that the input file is an IDL file, regardless of the file extension.

-inputXml: Indicates that the input file is a XML file, regardless of the file extension.

-inputXsd: Indicates that the input file is a XSD file, regardless of the file extension.

-inputWsdl: Indicates that the input file is a WSDL file, regardless of the file extension.

IDLInputFile.idl: File containing IDL descriptions of your data types. If -inputIdl is not used, the file must have an .idl extension.

XMLInputFile.xml: File containing XML descriptions of your data types. If -inputXml is not used, the file must have an .xml extension.

XSDInputFile.xsd: File containing XSD descriptions of your data types. If -inputXsd is not used, the file must have an .xsd extension.

XSDInputFile.wsdl: WSDL file containing XSD descriptions of your data types. If -inputWsdl is not used, the file must have an .wsdl extension.

6.123.2 Description

rtiddsgen takes a language-independent specification of the data (in IDL, XML, XSD or WSDL notation) and generates supporting classes and code to distribute instances of the data over RTI Connext.

To use rtiddsgen, you must first write a description of your data types in IDL, XML, XSD or WSDL format.

6.123.3 C++ Example

The following is an example generating the RTI Connext type myDataType:

IDL notation

```
struct myDataType {
    long value;
};
```

XML notation

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="rti_dds_topic_types.xsd">
    <struct name="myDataType">
        <member name="value" type="long"/>
    </struct>
</types>
```

XSD notation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:dds="http://www.omg.org/dds"
     xmlns:tns="http://www.omg.org/IDL-Mapped/"
     targetNamespace="http://www.omg.org/IDL-Mapped/">
  <xsd:import namespace="http://www.omg.org/dds" schemaLocation="rti_dds_topic_types_common.xsd"/
  <xsd:complexType name="myDataType">
    <xsd:sequence>
      <xsd:element name="value" minOccurs="1" maxOccurs="1" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

WSDL notation

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:dds="http://www.omg.org/dds" xmlns:x
  <types>
    <xsd:schema targetNamespace="http://www.omg.org/IDL-Mapped/">
      <xsd:import namespace="http://www.omg.org/dds" schemaLocation="rti_dds_topic_types_common.x
      <xsd:complexType name="myDataType">
        <xsd:sequence>
          <xsd:element name="value" minOccurs="1" maxOccurs="1" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>
```

Assuming the name of the idl file is myFileName.(idl|xml|xsd|wsdl) then all you need to do is type:

rtiddsgen myFileName.(idl|xml|xsd|wsdl)

This generates myFileName.cxx, myFileName.h, myFileNamePlugin.cxx, myFileNamePlugin.h, myFileNameSupport.cxx and myFileNameSupport.h. By default, rtiddsgen will not overwrite these files. You must use the -replace argument to do that.

6.123.4 IDL Language

In the IDL language, data types are described in a fashion almost identical to structures in "C." The complete description of the language can be found at the OMG website.

rtiddsgen does not support the full IDL language.

For detailed information about the IDL support in RTI Connext Service see Chapter 3 of the user manual.

Below are the IDL types that are currently supported:

- char
- wchar
- octet
- short
- unsigned short
- long
- unsigned long
- long long
- unsigned long long
- float
- double
- long double
- boolean
- bounded string
- unbounded string
- bounded wstring
- unbounded wstring

- enum
- typedef
- struct
- valuetypes (limited support)
- union
- sequences
- unbounded sequences
- arrays
- array of sequences
- constant

The following non-IDL types are also supported by rtiddsgen:

- bitfield
- valued enum

Use of Unsupported Types in an IDL File

You may include unsupported data types in the IDL file. rtiddsgen does not consider this an error. This allows you to use types that are defined in non-IDL languages with either hand-written or non-rtiddsgen written plug-ins. For example, the following is allowable:

```
//@copy #include "Bar.h"
//@copy #include "BarHandGeneratedPlugin.h"
struct Foo {
short height;
Bar barMember;
};
```

In the above case, Bar is defined externally by the user.

Multiple Types in a Single File

You can specify multiple types in a single IDL file. This can simplify management of files in your distributed program.

Use of Directives in an IDL File

The following directives can be used in your IDL file: Note: Do not put a space between the slashes and the @ sign. Note: Directives are case-sensitive (for example: use key, not Key).

- //@key Specifies that the field declared just before this directive in the enclosing structure is part of the key. Any number of a structure's fields may be declared part of the key.

- //@copy Copies a line of text (verbatim) into the generated code (for all languages). The text is copied into all the type-specific files generated by rtiddsgen except the examples.

- //@copy-declaration Like //@copy, but only copies the text into the file where the type is declared (<type>.h for C++/C, or <type>.java for Java).

- //@copy-c Like //@copy, but for C++/C-only code.

- //@copy-c-declaration Like //@copy-declaration, but for C++/C-only code.

- //@copy-java Like //@copy, but for Java-only code.

- //@copy-java-begin Copies a line of text at the beginning of all the Java files generated for a type. The directive only applies to the first type that is immediately below in the IDL file.

- //@copy-java-declaration Like //@copy-declaration, but for Java-only code.

- //@copy-java-declaration-begin Like //@copy-java-begin but only copies the text into the file where the type is declared.

- //@copy-ada Like //@copy, but for Ada-only code.

- //@copy-ada-begin Like //@copy-java-begin, but for Ada-only code.

- //@copy-ada-declaration Like //@copy-declaration, but for Ada-only code.

- //@copy-ada-declaration-begin Like //@copy-java-declaration, but for Ada-only code.

- //@resolve-name [true|false] Specifies whether or not rtiddsgen should resolve the scope of a type. If this directive is not present or is set to true, rtiddsgen resolves the scope. Otherwise rtiddsgen delegates the resolution of a type to the user.

- //@top-level [true|false] Specifies whether or not rtiddsgen should generate type-support code for a particular struct or union. The default is true.

6.123.5 XML Language

The data types can be described using XML.

RTI Connext provides DTD and XSD files that describe the XML format.

The DTD definition of the XML elements can be found in ../../../resource/dtd/rti_dds_topic_types.dtd" under <NDDSHOME>/resource/rtiddsgen/schema.

The XSD definition of the XML elements can be found in ../../../resource/xsd/rti_dds_topic_types.xsd" under <NDDSHOME>/resource/rtiddsgen/schema.

The XML validation performed by rtiddsgen always uses the DTD definition. If the <!DOCTYPE> tag is not present in the XML file, rtiddsgen will look for the DTD document under <NDDSHOME>/resource/rtiddsgen/schema. Otherwise, it will use the location specified in <!DOCTYPE>.

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Connext User Manual.

6.123.6 XSD Language

The data types can be described using XML schemas (XSD files). The XSD specification is based on the standard IDL to WSDL mapping described in the OMG document `CORBA to WSDL/SOAP Interworking Specification`

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Connext User Manual.

6.123.7 WSDL Language

The data types can be described using XML schemas contained in WSDL files. The XSD specification is based on the standard IDL to WSDL mapping described in the OMG document `CORBA to WSDL/SOAP Interworking Specification`

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Connext User Manual.

6.123.8 Using Generated Types Without RTI Connext (Standalone)

You can use the generated type-specific source and header files without linking the RTI Connext libraries or even including the RTI Connext header files. That is, the generated files for your data types can be used standalone.

The directory <NDDSHOME>/resource/rtiddsgen/standalone contains the helper files required to work in standalone mode:

- include: header and templates files for C/C++.
- src: source files for C/C++.
- class: Java jar file.

Using Standalone Types in C

The generated files that can be used standalone are:

- <idl file name>.c : Types source file
- <idl file name>.h : Types header file

You *cannot* use the type plug-in (<idl file>Plugin.c <idl file>Plugin.h) or the type support (<idl file>Support.c <idl file>Support.h) code standalone.

To use the rtiddsgen-generated types in a standalone manner:

- Include the directory <NDDSHOME>/resource/rtiddsgen/standalone/include in the list of directories to be searched for header files.
- Add the source files ndds_standalone_type.c and <idl file name>.c to your project.
- Include the file <idl file name>.h in the source files that will use the generated types in a standalone way.
- Compile the project using the two following preprocessor definitions:
  - NDDS_STANDALONE_TYPE
  - The definition for your platform: RTI_VXWORKS, RTI_QNX, RTI_WIN32, RTI_INTY, RTI_LYNX or RTI_UNIX

Using Standalone Types in C++

The generated files that can be used standalone are:

- <idl file name>.cxx : Types source file
- <idl file name>.h : Types header file

You *cannot* use the type plugin (<idl file>Plugin.cxx <idl file>Plugin.h) or the type support (<idl file>Support.cxx <idl file>Support.h) code standalone.

To use the generated types in a standalone manner:

- Include the directory <NDDSHOME>/resource/rtiddsgen/standalone/include in the list of directories to be searched for header files.
- Add the source files ndds_standalone_type.cxx and <idl file name>.cxx to your project.
- Include the file <idl file name>.h in the source files that will use the generated types in a standalone way.
- Compile the project using the two following preprocessor definitions:
  - NDDS_STANDALONE_TYPE
  - The definition for your platform: RTI_VXWORKS, RTI_QNX, RTI_WIN32, RTI_INTY, RTI_LYNX or RTI_UNIX

Standalone Types in Java

The generated files that can be used standalone are:

- <idl type>.java
- <idl type>Seq.java

You *cannot* use the type code (<idl file>TypeCode.java), the type support (<idl type>TypeSupport.java), the data reader (<idl file>DataReader.java) or the data writer code (<idl file>DataWriter.java) standalone.

To use the generated types in a standalone manner:

- Include the file ndds_standalone_type.jar in the classpath of your project.
- Compile the project using the standalone types files (<idl type>.java <idl type>Seq.java).

6.124 rtiddsping

Sends or receives simple messages using RTI Connext. The rtiddsping utility uses RTI Connext to send and receive preconfigured "Ping" messages to other rtiddsping applications which can be running in the same or different computers.

The rtiddsping utility can used to test the network and/or computer configuration and the environment settings that affect the operation of RTI Connext.

Usage

```
  rtiddsping [-help] [-version]
    [-domainId <domainId>]     ... defaults to 0
    [-index <NN>]              ... defaults to -1 (auto)
    [-appId <ID>]              ... defaults to a middleware-selected value
    [-Verbosity <NN>]          ... can be 0..5
    [-peer <PEER>]             ... PEER format is NN@TRANSPORT://ADDRESS
    [-discoveryTTL <NN>]       ... can be 0..255
[-transport <MASK>]        ... defaults to DDS_TRANSPORTBUILTIN_MASK_DEFAULT
[-msgMaxSize <SIZE>]       ... defaults to -1 (transport default)
[-shmRcvSize <SIZE>]       ... defaults to -1 (transport default)
[-deadline   <SS>]         ... defaults to -1 (no deadline)
[-durability <TYPE>]       ... TYPE can be VOLATILE or TRANSIENT_LOCAL
    [-multicast  <ADDRESS>]    ... defaults to no multicast
[-numSamples <NN>]         ... defaults to infinite
    [-publisher]                 ... this is the default
[-queueSize  <NN>]         ... defaults to 1
[-reliable]                  ... defaults to best-efforts
[-sendPeriod <SS>]         ... SS is in seconds, defaults to 1
[-subscriber]
[-timeFilter <SS>]         ... defaults to  0 (no filter)
[-timeout    <SS>]         ... SS is in seconds, defaults to infinite
[-topicName  <NAME>]       ... defaults to PingTopic
[-typeName   <NAME>]       ... defaults to PingType
[-useKeys    <NN>]         ... defaults to PingType
    [-qosFile <file>]
    [-qosProfile  <lib.prof>]
```

Example: `rtiddsping -domainId 3 -publisher -numSamples 100`

VxWorks Usage

```
  rtiddsping "[<options>]"
       The options use the same syntax as above.
```

Example `rtiddsping "-domainId 3 -publisher -numSamples 100"`

If the stack of the shell is not large enough to run rtiddsping, use "taskSpawn":

taskSpawn <name>,<priority>,<taskspawn options>,<stack size in bytes>,rtiddsping,"[\<options
The options use the same syntax as above.

Example taskSpawn "rtiddsping",100,0x8,50000,rtiddsping,"-domainId 3 -publisher -numSamples

Options:

-help Prints a help message and exits.

-version Prints the version and exits.

-Verbosity <NN> Sets the verbosity level. The range is 0 to 5.

0 has minimal output and does not echo the fact that data is being sent or received.

1 prints the most relevant statuses, including the sending and receiving of data. This is the default.

2 prints a summary of the parameters that are in use and echoes more detailed status messages.

3-5 Mostly affects the verbosity used by the internal RTI Connext modules that implement rtiddsping. The output is not always readable; its main purpose is to provide information that may be useful to RTI's support team.

Example: rtiddsping -Verbosity 2

-domainId <NN>

Sets the domain ID. The valid range is 0 to 100.

Example: rtiddsping -domainId 31

-appId <ID>

Sets the application ID. If unspecified, the system will pick one automatically.

This option is rarely used.

Example: rtiddsping -appId 34556

-index <NN>

Sets the participantIndex. If participantIndex is not -1 (auto), it must be different than the one used by all other applications in the same computer and domainId. If this is not respected, rtiddsping (or the application that starts last) will get an initialization error.

Example: rtiddsping -index 2

-peer <PEER>

Specifies a PEER to be used for discovery. Like any RTI Connext application, it defaults to the setting of the environment variable NDDS_DISCOVERY_PEERS

or a preconfigured multicast address if the environment is not set.

The format used for PEER is the same one used for NDDS_DISCOVERY_PEERS and is described in detail in **NDDS_DISCOVERY_PEERS** (p. 55). A brief summary follows:

The general format is: NN@TRANSPORT://ADDRESS where:

- ADDRESS is an address (in name form or using the IP notation xxx.xxx.xxx.xxx). ADDRESS may be a multicast address.
- TRANSPORT represents the kind of transport to use and NN is the maximum participantIndex expected at that location. NN can be omitted and it is defaulted to '4'
- Valid settings for TRANSPORT are 'udpv4' and 'shmem'. The default setting if the transport is omitted is 'udpv4'.
- ADDRESS cannot be omitted if the '-peer' option is specified.

The -peer option may be repeated to specify multiple peers.

Example: rtiddsping -peer 10.10.1.192 -peer mars -peer 4@pluto

-discoveryTTL <TTL>

Sets the TTL (time-to-live) used for multicast discovery. If not specified, it defaults to the built-in RTI Connext default.

The valid range is 0 to 255. The value '0' limits multicast to the node itself (i.e., can only discover applications running on the same computer). The value '1' limits multicast discovery to computers on the same subnet. Values higher than 1 generally indicate the maximum number of routers that may be traversed (although some routers may be configured differently).

Example: rtiddsping -discoveryTTL 16

-transport <MASK>

A bit-mask that sets the enabled builtin transports. If not specified, the default set of transports is used (UDPv4 + shmem). The bit values are: 1=UDPv4, 2=shmem, 8=UDPv6.

-msgMaxSize <SIZE>

Configure the maximum message size allowed by the installed transports. This is needed if you are using rtiddsping to communicate with an application that has set these transport parameters to larger than default values.

-shmRcvSize <SIZE>

Increase the shared memory receive-buffer size. This is needed if you are using rtiddsping to communicate with an application that has set these transport parameters to larger than default values.

-deadline <SS>

This option only applies if the '-subscriber' option is also specified.

Sets the DEADLINE QoS for the subscriptions made by rtiddsping.

Note that this may cause the subscription QoS to be incompatible with the publisher if the publisher did not specify a sendPeriod greater than the deadline. If the QoS is incompatible, rtiddsping will not receive updates.

Each time a deadline is detected, rtiddsping will print a message indicating the number of deadlines received so far.

Example: rtiddsping -deadline 3.5

-durability <TYPE>

Sets the DURABILITY QoS used for publishing or subscribing. Valid settings are: VOLATILE and TRANSIENT_LOCAL (default). The effect of this setting can only be observed when it is used in in conjunction with reliability and a queueSize larger than 1. If all these conditions are met, a late-joining subscriber will be able to see up to queueSize samples that were previously written by the publisher.

Example: rtiddsping -durability VOLATILE

-multicast <ADDRESS>

This option only applies if the '-subscriber' option is also specified.

Configures ping to receive messages over multicast. The <ADDRESS> parameter indicates the address to use. ADDRESS must be in the valid range for multicast addresses. For IP version 4 the valid range is 224.0.0.1 to 239.255.255.255

Example: rtiddsping -multicast 225.1.1.1

-numSamples <NN>

Sets the number of samples that will be sent by rtiddsping. After those samples are sent, rtiddsping will exit. messages.

Example: rtiddsping -numSamples 10

-publisher

Causes rtiddsping to send ping messages. This is the default.

Example: rtiddsping -publisher

-queueSize <NN>

Specifies the maximal number of samples to hold in the queue. In the case of the publisher, it affects the samples that are available for a late-joining subscriber.

Example: rtiddsping -queueSize 100

-reliable

Configures the RELIABILITY QoS for publishing or subscribing. The default setting (if -reliable is not used) is BEST_EFFORT

Example: rtiddsping -reliable

-sendPeriod <SS>

Sets the period (in seconds) at which rtiddsping sends the messages.

Example: rtiddsping -sendPeriod 0.5

-subscriber

Causes rtiddsping to listen for ping messages. This option cannot be specified if '-publisher' is also specified.

Example: rtiddsping -subscriber

-timeFilter <SS>

This option only applies if the '-subscriber' option is also specified.

Sets the TIME_BASED_FILTER QoS for the subscriptions made by rtiddsping. This QoS causes RTI Connext to filter out messages that are published at a rate faster than what the filter duration permits. For example, if the filter duration is 10 seconds, messages will be printed no faster than once every 10 seconds.

Example: rtiddsping -timeFilter 5.5

-timeout <SS>

This option only applies if the '-subscriber' option is also specified.

Sets a timeout (in seconds) that will cause rtiddsping to exit if no samples are received for a duration that exceeds the timeout.

Example: rtiddsping -timeout 30

-topicName <NAME>

Sets the topic name used by rtiddsping. The default is 'RTIddsPingTopic'. To communicate, both the publisher and subscriber must specify the same topic name.

Example: rtiddsping -topicName Alarm

-typeName <NAME>

Sets the type name used by rtiddsping. The default is 'RTIddsPingType'. To communicate, both publisher and subscriber must specify the same type name.

Example: rtiddsping -typeName AlarmDescription

-useKeys <NN>

This option causes rtiddsping to use a topic whose data contains a key. The value of the NN parameter indicates the number of different data objects (each identified by a different value of the key) that will be published by rtiddsping.

The value of NN only affects the publishing behavior. However NN still needs to be specified when the -useKeys option is used with the -subscriber option.

For communication to occur, both the publisher and subscriber must agree on whether the topic that they publish/subscribe contains a key. Consequently, if you specify the -useKeys parameter for the publisher, you must do the same with the subscriber. Otherwise communication will not be stablished.

Example: rtiddsping -useKeys 20

-qosFile <file>

Allow you to specify additional QoS XML settings using url_profile. For more information on the syntax, see Chapter 15 in the RTI Connext User's Manual.

Example: rtiddsping -qosFile /home/user/QoSProfileFile.xml

-qosProfile <lib.prof>

This option specifies the library name and profile name that the tool should use.

QoS settings

rtiddsping is configured internally using a special set of QoS settings in a profile called InternalPingLibrary.InternalPingProfile. This is the default profile unless a profile called DefaultPingLibrary.DefaultPingProfile is found. You can use the command-line option -qosProfile to tell rtiddsping to use a different lib.profile instead of DefaultPingLibrary.DefaultPingProfile. Like all the other RTI Connext applications, rtiddsping loads all the profiles specified using the environment variable NDDS_QOS_PROFILES or the file named USER_QOS_PROFILES found in the current working directory.

The QoS settings used internally are available in the file RTIDDSSPING_QOS_PROFILES.example.xml.

Description

The usage depends on the operating system from which rtiddsping is executed.

Examples for UNIX, Linux, and Windows Systems

On UNIX, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'shell prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
shell prompt>  rtiddsping -domainId 3 -publisher -numSamples 100
shell prompt>  rtiddsping -domainId 5 -subscriber -timeout 20
shell prompt>  rtiddsping -help
```

VxWorks examples:

On VxWorks systems, the libraries libnddscore.so, libnddsc.so and libnddscpp.so must first be loaded. The rtiddsping command must be typed to the VxWorks

shell (either an rlogin shell, a target-server shell, or the serial line prompt). The arguments are passed embedded into a single string, but otherwise have the same syntax as for Unix/Windows. In the Unix, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'vxworks prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
vxworks prompt>  rtiddsping "-domainId 3 -publisher -numSamples 100"
vxworks prompt>  rtiddsping "-domainId 5 -subscriber -timeout 20"
vxworks prompt>  rtiddsping "-help"

or, alternatively (to avoid overflowing the stack):

vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-domainId 3 -publisher -numSamples 10
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-domainId 5 -subscriber -timeout 20"
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-help"
```

6.125 rtiddsspy

Debugging tool which receives all RTI Connext communication. The rtiddsspy utility allows the user to monitor groups of publications available on any RTI Connext domain.

Note: If you have more than one DataWriter for the same Topic, and these DataWriters have different settings for the Ownership QoS, then rtiddsspy will only receive (and thus report on) the samples from the first DataWriter.

To run rtiddsspy, like any RTI Connext application, you must have the NDDS_DISCOVERY_PEERS environment variable that defines your RTI Connext domain; otherwise you must specify the peers as command line parameters.

Usage

```
rtiddsspy [-help] [-version]
     [-domainId <domainId>]     ... defaults to 0
     [-index <NN>]              ... defaults to -1 (auto)
     [-appId <ID>]              ... defaults to a middleware-selected value
     [-Verbosity <NN>]          ... can be 0..5
     [-peer <PEER>]             ... PEER format is NN@TRANSPORT://ADDRESS
     [-discoveryTTL <NN>]       ... can be 0..255
     [-transport <MASK>]        ... defaults to DDS_TRANSPORTBUILTIN_MASK_DEFAULT
     [-msgMaxSize <SIZE>]       ... defaults to -1 (transport default)
     [-shmRcvSize <SIZE>]       ... defaults to -1 (transport default)
     [-tcMaxSize <SIZE>]        ... defaults to 4096
     [-hOutput]
     [-deadline   <SS>]         ... defaults to -1 (no deadline)
     [-history   <DEPTH>]       ... defaults to 8192
     [-timeFilter <SS>]         ... defaults to  0 (no filter)
     [-useFirstPublicationQos]
     [-showHandle]
     [-typeRegex <REGEX>]       ... defaults to "*"
     [-topicRegex <REGEX>]      ... defaults to "*"
     [-typeWidth <WIDTH>]       ... can be 1..255
     [-topicWidth <WIDTH>]      ... can be 1..255
     [-truncate]
     [-printSample]
     [-qosFile <file>]
     [-qosProfile  <lib.prof>]
```

Example: `rtiddsspy -domainId 3 -topicRegex "Alarm*"`

VxWorks Usage

```
rtiddsspy "[<options>]"
     The options use the same syntax as above.
```

Example `rtiddsspy "-domainId 3 -topicRegex Alarm*"`

`rtiddsspy requires about 25 kB of stack. If the stack size of the shell from which it is invoked is not la`

taskSpawn `<name>, <priority>, <taskspawn options>, <stack size in bytes>, rtiddsspy, "[\<options\>]"`

`The options use the same syntax as above.`

Example `taskSpawn "rtiddsspy", 100, 0x8, 50000, rtiddsspy, "-domainId 3 -topicRegex Alarm*"`

Options:

-help Prints a help message and exits.

-version Prints the version and exits.

-Verbosity <NN> Sets the verbosity level. The range is 0 to 5.

0 has minimal output and does not echo the fact that data is being sent or received.

1 prints the most relevant statuses, including the sending and receiving of data. This is the default.

2 prints a summary of the parameters being used and echoes more detailed status messages.

3-5 Mostly affect the verbosity used by the internal RTI Connext modules that implement rtiddsspy. The output is not always readable; its main purpose is to provide information that may be useful to RTI's support team.

Example: rtiddsspy -Verbosity 2

-domainId <NN>

Sets the domain ID. The valid range is 0 to 100.

Example: rtiddsspy -domainId 31

-appId <ID>

Sets the application ID. If unspecified, the system will pick one automatically.

This option is rarely used.

Example: rtiddsspy -appId 34556

-index <NN>

Sets the participantIndex. If participantIndex is not -1 (auto), it must be different than the one used by all other applications in the same computer and domainId. If this is not respected, rtiddsspy (or the application that starts last) will get an initialization error.

Example: rtiddsspy -index 2

-peer <PEER>

Specifies a PEER to be used for discovery. Like any RTI Connext application, it defaults to the setting of the environment variable NDDS_DISCOVERY_PEERS or a preconfigured multicast address if the environment is not set.

The format used for PEER is the same used for the NDDS_DISCOVERY_PEERS and is described in detail in **NDDS_DISCOVERY_PEERS** (p. 55). A brief summary follows:

The general format is: NN@TRANSPORT://ADDRESS where:

- ADDRESS is an address (in name form or using the IP notation xxx.xxx.xxx.xxx). ADDRESS may be a multicast address.
- TRANSPORT represents the kind of transport to use and NN is the maximum participantIndex expected at that location. NN can be omitted and it is defaulted to '4'
- Valid settings for TRANSPORT are 'udpv4' and 'shmem'. The default setting if the transport is omitted is 'udpv4'
- ADDRESS cannot be omitted if the '-peer' option is specified.

The -peer option may be repeated to specify multiple peers.

Example: rtiddsspy -peer 10.10.1.192 -peer mars -peer 4@pluto

-discoveryTTL <TTL>

Sets the TTL (time-to-live) used for multicast discovery. If not specified, it defaults to the built-in RTI Connext default.

The valid range is 0 to 255. The value '0' limits multicast to the node itself (i.e. can only discover applications running on the same computer). The value '1' limits multicast discovery to computers on the same subnet. Settings greater than 1 generally indicate the maximum number of routers that may be traversed (although some routers may be configured differently).

Example: rtiddsspy -discoveryTTL 16

-transport <MASK>

SPecifies a bit-mask that sets the enabled builtin transports. If not specified, the default set of transports is used (UDPv4 + shmem). The bit values are: 1=UDPv4, 2=shmem, 8=UDPv6.

-msgMaxSize <SIZE>

Configures the maximum message size allowed by the installed transports. This is needed if you are using rtiddsspy to communicate with an application that has set these transport parameters to larger than default values.

-shmRcvSize <SIZE>

Increases the shared memory receive-buffer size. This is needed if you are using rtiddsspy to communicate with an application that has set these transport parameters to larger than default values.

-tcMaxSize <SIZE>

Configures the maximum size, in bytes, of a received type code.

-hOutput

Prints information on the output format used by rtiddsspy.

This option prints an explanation of the output and then exits.

Example: rtiddsspy -hOutput

-deadline <SS>

Sets the requested DEADLINE QoS for the subscriptions made by rtiddsspy.

Note that this may cause the subscription QoS to be incompatible with the publisher if the publisher did not specify an offered deadline that is greater or equal to the one requested by rtiddsspy. If the QoS is incompatible rtiddsspy will not receive updates from that writer.

Each time a deadline is detected rtiddsspy will print a message that indicates the number of deadlines received so far.

Example: rtiddsspy -deadline 3.5

-timeFilter <SS>

Sets the TIME_BASED_FILTER QoS for the subscriptions made by rtiddsspy. This QoS causes RTI Connext to filter-out messages that are published at a rate faster than what the filter duration permits. For example if the filter duration is 10 seconds, messages will be printed no faster than once each 10 seconds.

Example: rtiddsspy -timeFilter 10.0

-history <DEPTH>

Sets the HISTORY depth QoS for the subscriptions made by rtiddsspy.

This may be relevant if the publisher has batching turned on, or if the -useFirstPublicationQos option is used that is causing a reliable or durable subscription to be created.

Example: rtiddsspy -history 1

-useFirstPublicationQos

Sets the RELIABILITY and DURABILITY QoS of the subscription based on the first discovered publication of that topic.

See also -history option.

Example: rtiddsspy -useFirstPublicationQos

-showHandle

Prints additional information about each sample received. The additional information is the 'instance_handle' field in the SampleHeader, which can be used to distinguish among multiple instances of data objects published under the same topic and type names.

Samples displayed that share the topic and type names and also have the same value for the instance_handle represent value updates to the same data object. On the other hand, samples that share the topic and type names but display different values for the instance_handle.

This option causes rtiddsspy to print an explanation of updates to the values of different data objects.

Example: rtiddsspy -showHandle

-typeRegex <REGEX>

Subscribe only to types that match the REGEX regular expression. The syntax of the regular expression is defined by the POSIX regex function.

When typing a regular expression to a command-line shell, some symbols may need to be escaped to avoid interpretation by the shell. In general, it is safest to include the expression in double quotes.

This option may be repeated to specify multiple topic expressions.

Example: rtiddsspy -typeRegex "SensorArray*"

-topicRegex <REGEX>

Subscribe only to topics that match the REGEX regular expression. The syntax of the regular expression is defined by the POSIX regex function.

When typing a regular expression to a command-line shell, some symbols may need to be escaped to avoid interpretation by the shell. In general, it is safest to include the expression in double quotes.

This option may be repeated to specify topic multiple expressions.

Example: rtiddsspy -topicRegex "Alarm*"

-typeWidth <WIDTH>

Sets the maximum width of the Type name column. Names wider than this will wrap around, unless `-truncate` is specified. Can be 1..255.

-topicWidth <WIDTH>

Sets the maximum width of the Topic name column. Names wider than this will wrap around, unless `-truncate` is specified. Can be 1..255.

-truncate

Specifies that names exceeding the maximum number of characters should be truncated.

-printSample

Prints the value of the received samples.

-qosFile <file>

Allows you to specify additional QoS XML settings using url_profile. For more information on the syntax, see Chapter 15 in the RTI Connext User's Manual.

Example: rtiddsspy -qosFile /home/user/QoSProfileFile.xml

-qosProfile <lib.prof>

Specifies the library name and profile name to be used.

QoS settings

rtiddsspy is configured to discover as many entities as possible. To do so, an internal profile is defined, called InternalSpyLibrary.InternalSpyProfile. This is the default profile, unless a profile called DefaultSpyLibrary.DefaultSpyProfile is found. You can use the command-line option -qosProfile to tell rtiddsspy to use a speficied lib.profile instead of DefaultSpyLibrary.DefaultSpyProfile. Like all the other RTI Connext applications, rtiddsspy loads all the profiles specified using the environment variable NDDS_QOS_PROFILES or the file named USER_QOS_PROFILES found in the current working directory.

The QoS settings used internally are available in the file RTIDDSSPY_QOS_PROFILES.example.xml.

Usage Examples

The usage depends on the operating system from which rtiddsspy is executed.

Examples for UNIX, Linux, Windows systems

On UNIX, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'shell prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
shell prompt>  rtiddsspy -domainId 3
shell prompt>  rtiddsspy -domainId 5 -topicRegex "Alarm*"
shell prompt>  rtiddsspy -help
```

Examples for VxWorks Systems

On VxWorks systems, the libraries libnddscore.so, libnddsc.so and libnddscpp.so must first be loaded. The rtiddsspy command must be typed to the VxWorks shell (either an rlogin shell, a target-server shell, or the serial line prompt). The arguments are passed embedded into a single string, but otherwise have the same syntax as for Unix/Windows. In UNIX, Linux, Windows and other

operating systems that have a shell, the syntax matches the one of the regular comamnds available in the shell. In the examples below, the string 'vxworks prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
vxworks prompt>  rtiddsspy "-domainId 3"
vxworks prompt>  rtiddsspy "-domainId 5  5 -topicRegex "Alarm*"
vxworks prompt>  rtiddsspy "-help"
```

Chapter 7

Namespace Documentation

7.1 Package com.rti.dds.domain

Contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up RTI Connext.

Classes

- ˆ interface **DomainParticipant**

  <<**interface**>> (p. 271) *Container for all* ***com.rti.dds.infrastructure.DomainEntity*** *(p. 628) objects.*

- ˆ class **DomainParticipantAdapter**

  <<**eXtension**>> (p. 270) *A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*

- ˆ class **DomainParticipantFactory**

  <<**singleton**>> (p. 271) <<**interface**>> (p. 271) *Allows creation and destruction of* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) objects.*

- ˆ class **DomainParticipantFactoryQos**

  QoS policies supported by a ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- ˆ interface **DomainParticipantListener**

<<**interface**>> (p. 271) *Listener for participant status.*

ˆ class **DomainParticipantQos**

QoS policies supported by a ***com.rti.dds.domain.DomainParticipant*** *(p. 629) entity.*

Packages

ˆ package **builtin**

Builtin ***topic*** *(p. 350) for accessing information about the DomainParticipants discovered by RTI Connext.*

7.1.1 Detailed Description

Contains the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) also acts as a container for the other objects that make up RTI Connext.

7.2 Package com.rti.dds.domain.builtin

Builtin **topic** (p. 350) for accessing information about the DomainParticipants discovered by RTI Connext.

Classes

- ˆ class **ParticipantBuiltinTopicData**

  *Entry created when a **DomainParticipant** (p. 629) object is discovered.*

- ˆ class **ParticipantBuiltinTopicDataDataReader**

  Instantiates `DataReader` < ***builtin.ParticipantBuiltinTopicData*** *(p. 1227)* > .

- ˆ class **ParticipantBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***builtin.ParticipantBuiltinTopicData*** *(p. 1227)* > .

- ˆ class **ParticipantBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` < ***builtin.ParticipantBuiltinTopicData*** *(p. 1227)* > .

7.2.1 Detailed Description

Builtin **topic** (p. 350) for accessing information about the DomainParticipants discovered by RTI Connext.

7.3 Package com.rti.dds.dynamicdata

*<<**eXtension**>>* *(p. 270)* 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. 780) 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. 780) objects.*

- class **DynamicDataReader**

  Reads (subscribes to) objects of type ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780).*

- class **DynamicDataSeq**

  An ordered collection of ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) elements.*

- class **DynamicDataTypeProperty_t**

  A collection of attributes used to configure ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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. 780) objects.*

ˆ class **DynamicDataWriter**

Writes (publishes) objects of type ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780).*

7.3.1 Detailed Description

*<<**eXtension**>> (p. 270)* 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 TypeCode (see Type Code Support (p. 162)) that defines the type definition you want to use.

A TypeCode includes a type's *kind* (TCKind), *name*, and *members* (that is, fields). You can create your own TypeCode using the TypeCodeFactory class – see, for example, the **TypeCodeFactory.create_struct_tc** (p. 1644) method. Alternatively, you can use a remote TypeCode that you discovered on the network (see **Built-in Topics** (p. 153)) or one generated by **rtiddsgen** (p. 290).

2. Wrap the TypeCode in a com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 887) object.

See the constructor **DynamicDataTypeSupport.DynamicDataTypeSupport** (p. 889). This object lets you connect the type definition to a **com.rti.dds.domain.DomainParticipant** (p. 629) and manage data samples (of type **com.rti.dds.dynamicdata.DynamicData** (p. 780)).

3. Register the com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 887) with one or more domain (p. 317) participants.

See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 889). 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. 1545) from the com.rti.dds.domain.DomainParticipant (p. 629).

Use the name under which you registered your data type – see **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670). This **com.rti.dds.topic.Topic** (p. 1545) is what you will use to produce and consume data.

5. Create a com.rti.dds.dynamicdata.DynamicDataWriter (p. 893) and/or com.rti.dds.dynamicdata.DynamicDataReader (p. 851).

These objects will produce and/or consume data (of type **com.rti.dds.dynamicdata.DynamicData** (p. 780)) on the **com.rti.dds.topic.Topic** (p. 1545). You can create these objects directly from the **com.rti.dds.domain.DomainParticipant** (p. 629) – see **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 661) and **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 666) – or by first creating intermediate **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.subscription.Subscriber** (p. 1478) objects – see **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) and **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659).

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. 629) is optional; all types are automatically unregistered when the **com.rti.dds.domain.DomainParticipant** (p. 629) itself is deleted.

7.4 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 **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**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- class **BAD_TYPECODE**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- class **BadKind**

  *The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.*

- class **BadMemberId**

  The specified TypeCode member ID is invalid.

- class **BadMemberName**

  The specified TypeCode 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.util.Sequence` (p. 1432) *< boolean >.*

^ class **Bounds**

A user exception thrown when a parameter is not within the legal bounds.

^ class **BuiltinTopicReaderResourceLimits_t**

*Built-in **topic** (p. 350) reader's resource limits.*

^ class **ByteSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *byte* >.

^ class **ChannelSettings_t**

Type used to configure the properties of a channel.

^ class **ChannelSettingsSeq**

Declares IDL `sequence<` ***com.rti.dds.infrastructure.ChannelSettings_t*** *(p. 441)* >.

^ class **CharSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *char* >.

^ interface **Condition**

<<**interface**>> (p. 271) *Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1695).*

^ class **ConditionSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.infrastructure.Condition*** *(p. 451)* >.

^ class **ContentFilterProperty_t**

<<**eXtension**>> (p. 270) *Type used to provide all the required information to enable content filtering.*

^ class **Cookie_t**

<<**eXtension**>> (p. 270) *Sequence of bytes identifying a written data sample, used when writing with parameters.*

^ interface **Copyable**

<<**eXtension**>> (p. 270) <<**interface**>> (p. 271) *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 **DataReaderProtocolQosPolicy**

Along with ***com.rti.dds.infrastructure.WireProtocolQosPolicy*** *(p. 1709) and* ***com.rti.dds.infrastructure.DataWriterProtocolQosPolicy*** *(p. 571), this QoS policy configures the DDS on-the-network protocol (RTPS).*

ˆ class **DataReaderResourceLimitsQosPolicy**

Various settings that configure how a ***com.rti.dds.subscription.DataReader*** *(p. 473) allocates and uses physical memory for internal resources.*

ˆ class **DataWriterProtocolQosPolicy**

Protocol that applies only to ***com.rti.dds.publication.DataWriter*** *(p. 538) instances.*

ˆ class **DataWriterResourceLimitsInstanceReplacementKind**

Sets the kinds of instances that can be replaced when instance resource limits are reached.

ˆ class **DataWriterResourceLimitsQosPolicy**

Various settings that configure how a ***com.rti.dds.publication.DataWriter*** *(p. 538) allocates and uses physical memory for internal resources.*

ˆ class **DeadlineQosPolicy**

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

ˆ class **DestinationOrderQosPolicy**

Controls how the middleware will deal with data sent by multiple ***com.rti.dds.publication.DataWriter*** *(p. 538) entities for the same instance of data (i.e., same* ***com.rti.dds.topic.Topic*** *(p. 1545) and key).*

ˆ class **DestinationOrderQosPolicyKind**

Kinds of destination order.

ˆ class **DiscoveryBuiltinReaderFragmentationResourceLimits_t**

ˆ class **DiscoveryConfigBuiltinPluginKind**

Built-in discovery plugins that can be used.

ˆ class **DiscoveryConfigQosPolicy**

Settings for discovery configuration.

ˆ class **DiscoveryPluginPromiscuityKind**

<<**eXtension**>> (p. 270) *Type used to indicate promiscuity mode of the discovery plugin.*

^ class **DiscoveryQosPolicy**

Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

^ interface **DomainEntity**

<<**interface**>> (p. 271) *Abstract base class for all DDS entities except for the* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

^ class **DomainParticipantResourceLimitsQosPolicy**

Various settings that configure how a ***com.rti.dds.domain.DomainParticipant*** *(p. 629) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

^ class **DoubleSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *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. 473) 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. 765) setting of* ***DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS*** *(p. 772) or* ***DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS*** *(p. 771).*

^ 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. 909).*

^ interface **Entity**

<<**interface**>> (p. 271) *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. 912) types that can act as factories for one or more other* ***com.rti.dds.infrastructure.Entity*** *(p. 912) types.*

^ class **EntityNameQosPolicy**

Assigns a name and a role name to a ***com.rti.dds.domain.DomainParticipant*** *(p. 629),* ***com.rti.dds.publication.DataWriter*** *(p. 538) or* ***com.rti.dds.subscription.DataReader*** *(p. 473). 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 **ExclusiveAreaQosPolicy**

Configures multi-thread concurrency and deadlock prevention capabilities.

^ class **FloatSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *float* >.

^ class **GroupDataQosPolicy**

Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 153) during discovery.*

^ class **GuardCondition**

<<**interface**>> (p. 271) *A specific* ***com.rti.dds.infrastructure.Condition*** *(p. 451) 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.

ˆ class **InetAddressSeq**

Declares IDL `sequence`*< java.net.InetAddress >.*

ˆ class **InstanceHandle_t**

Type definition for an instance handle.

ˆ class **InstanceHandleSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) *<* ***com.rti.dds.infrastructure.InstanceHandle_t*** *(p. 1080) > .*

ˆ class **IntSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) *< int >.*

ˆ 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.

ˆ class **LifespanQosPolicy**

Specifies how long the data written by the ***com.rti.dds.publication.DataWriter*** *(p. 538) is considered valid.*

ˆ interface **Listener**

<<**interface**>> (p. 271) *Abstract base class for all* ***Listener*** *(p. 1154) interfaces.*

ˆ class **LivelinessQosPolicy**

Specifies and configures the mechanism that allows ***com.rti.dds.subscription.DataReader*** *(p. 473) entities to detect when* ***com.rti.dds.publication.DataWriter*** *(p. 538) entities become disconnected or "dead.".*

ˆ class **LivelinessQosPolicyKind**

Kinds of liveliness.

ˆ class **Locator_t**

<<**eXtension**>> (p. 270) *Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*

ˆ 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 builtin.PublicationBuiltinTopicData.

^ class **LocatorFilterSeq**

Declares IDL `sequence<` ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1178)* `>`.

^ class **LocatorSeq**

Declares IDL `sequence <` ***com.rti.dds.infrastructure.Locator_t*** *(p. 1174)* `>`.

^ class **LoggingQosPolicy**

Configures the RTI Connext logging facility.

^ class **LongDoubleSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) `<` *com.rti.dds.infrastructure.LongDouble* `>`.

^ class **LongSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) `<` *long* `>`.

^ class **MultiChannelQosPolicy**

Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

^ class **ObjectHolder**

<<**eXtension**>> (p. 270) *Holder of object instance*

^ class **OwnershipQosPolicy**

Specifies whether it is allowed for multiple ***com.rti.dds.publication.DataWriter*** *(p. 538) (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. 538) objects that attempt to modify the same instance of a data type (identified by* ***com.rti.dds.topic.Topic*** *(p. 1545) + key).*

- class **PartitionQosPolicy**

  Set of strings that introduces a logical partition among the topics visible by a ***com.rti.dds.publication.Publisher*** *(p. 1277) and a* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- 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. 270) *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 which facilitate management of the properties in the input policy.

- class **PropertySeq**

  Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.Property_t*** *(p. 1250) >.*

- class **ProtocolVersion_t**

  <<**eXtension**>> (p. 270) *Type used to represent the version of the RTPS protocol.*

- 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. 1318*).*

^ class **QosPolicyCountSeq**

Declares IDL `sequence` *<* ***com.rti.dds.infrastructure.QosPolicyCount*** *(* p. 1315 *) >.*

^ class **QosPolicyId_t**

Type to identify `QosPolicies`*.*

^ 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 **RefilterQosPolicyKind**

<<**eXtension**>> (p. 270) *Kinds of Refiltering*

^ class **ReliabilityQosPolicy**

Indicates the level of reliability offered/requested by RTI Connext.

^ 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.

ˆ class **RETCODE_ALREADY_DELETED**

The object target of this operation has already been deleted.

ˆ class **RETCODE_BAD_PARAMETER**

Illegal parameter value.

ˆ class **RETCODE_ERROR**

Generic, unspecified error.

ˆ class **RETCODE_ILLEGAL_OPERATION**

The operation was called under improper circumstances.

ˆ class **RETCODE_IMMUTABLE_POLICY**

Application attempted to modify an immutable QoS policy.

ˆ class **RETCODE_INCONSISTENT_POLICY**

Application specified a set of QoS policies that are not consistent with each other.

ˆ class **RETCODE_NO_DATA**

Indicates a transient situation where the operation did not return any data but there is no inherent error.

ˆ class **RETCODE_NOT_ENABLED**

*Operation invoked on a **com.rti.dds.infrastructure.Entity*** (p. 912) *that is not yet enabled.*

ˆ class **RETCODE_OUT_OF_RESOURCES**

RTI Connext ran out of the resources needed to complete the operation.

ˆ class **RETCODE_PRECONDITION_NOT_MET**

A pre-condition for the operation was not met.

ˆ class **RETCODE_TIMEOUT**

The operation timed out.

ˆ class **RETCODE_UNSUPPORTED**

Unsupported operation. Can only returned by operations that are unsupported.

ˆ class **RtpsReliableReaderProtocol_t**

Qos (p. 1313) *related to reliable reader protocol defined in RTPS.*

ˆ class **RtpsReliableWriterProtocol_t**

QoS related to the reliable writer protocol defined in RTPS.

ˆ class **RtpsReservedPortKind**

RTPS reserved port kind, used to identify the types of ports that can be reserved on ***domain*** (p. 317) *participant enable.*

ˆ class **RtpsWellKnownPorts_t**

RTPS well-known port mapping configuration.

ˆ class **SampleIdentity_t**

Type definition for an Sample Identity.

ˆ class **SequenceNumber_t**

Type for sequence *number representation.*

ˆ class **ShortSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < *short* >.

ˆ interface **StatusCondition**

<<**interface**>> (p. 271) *A specific* ***com.rti.dds.infrastructure.Condition*** (p. 451) *that is associated with each* ***com.rti.dds.infrastructure.Entity*** (p. 912).

ˆ class **StatusKind**

Type for status *kinds.*

ˆ class **StringSeq**

Declares IDL `sequence` < *String* > .

ˆ class **SystemException**

System exception.

ˆ class **SystemResourceLimitsQosPolicy**

Configures ***com.rti.dds.domain.DomainParticipant*** (p. 629)*-independent resources used by RTI Connext. Mainly used to change the maximum number of* ***com.rti.dds.domain.DomainParticipant*** (p. 629) *entities that can be created within a single process (address space).*

^ class **ThreadSettings_t**

The properties of a thread of execution.

^ class **ThreadSettingsCpuRotationKind**

Determines how ***com.rti.dds.infrastructure.ThreadSettings_t.cpu_list*** *(p. 1532) 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. 473) 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. 153) during discovery.*

^ class **TransportBuiltinKind**

Built-in transport kind.

^ class **TransportBuiltinQosPolicy**

Specifies which built-in transports are used.

^ 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. 473) 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. 629) level) transports with which to receive the multicast data.*

^ class **TransportMulticastMappingSeq**

Declares IDL sequence< ***com.rti.dds.infrastructure.TransportMulticastSettings_t*** *(p. 1594)* >.

ˆ class **TransportMulticastQosPolicy**

Specifies the multicast address on which a ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 629) level) transports with which to receive the multicast data.*

ˆ class **TransportMulticastQosPolicyKind**

Transport Multicast Policy Kind.

ˆ class **TransportMulticastSettings_t**

Type representing a list of multicast locators.

ˆ class **TransportMulticastSettingsSeq**

Declares IDL sequence< ***com.rti.dds.infrastructure.TransportMulticastSettings_t*** *(p. 1594)* >.

ˆ class **TransportPriorityQosPolicy**

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

ˆ class **TransportSelectionQosPolicy**

Specifies the physical transports a ***com.rti.dds.publication.DataWriter*** *(p. 538) or* ***com.rti.dds.subscription.DataReader*** *(p. 473) may use to send or receive data.*

ˆ class **TransportUnicastQosPolicy**

Specifies a subset of transports and a port number that can be used by an ***Entity*** *(p. 912) to receive data.*

ˆ class **TransportUnicastSettings_t**

Type representing a list of unicast locators.

ˆ class **TransportUnicastSettingsSeq**

Declares IDL sequence< ***com.rti.dds.infrastructure.TransportUnicastSettings_t*** *(p. 1608)* >.

ˆ class **TypeSupportQosPolicy**

Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

^ class **UserDataQosPolicy**

Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(*p. 153*) during discovery.*

^ class **UserException**

User exception.

^ class **VendorId_t**

<<**eXtension**>> (p. 270) *Type used to represent the vendor of the service implementing the RTPS protocol.*

^ class **WaitSet**

<<**interface**>> (p. 271) *Allows an application to wait until one or more of the attached* ***com.rti.dds.infrastructure.Condition*** *(*p. 451*) objects has a* `trigger_value` *of true or else until the timeout expires.*

^ class **WaitSetProperty_t**

<<**eXtension**>> (p. 270) *Specifies the* ***com.rti.dds.infrastructure.WaitSet*** *(*p. 1695*) behavior for multiple trigger events.*

^ class **WcharSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) *< char >.*

^ class **WireProtocolQosPolicy**

Specifies the wire-protocol-related attributes for the ***com.rti.dds.domain.DomainParticipant*** *(*p. 629*).*

^ class **WireProtocolQosPolicyAutoKind**

Kind of auto mechanism used to calculate the GUID prefix.

^ class **WriteParams_t**

<<**eXtension**>> (p. 270) *Input parameters for writing with com.rti.dds.topic.example.FooDataWriter.write_w_params, com.rti.dds.topic.example.FooDataWriter.dispose_w_params, com.rti.dds.topic.example.FooDataWriter.register_instance_w_params, com.rti.dds.topic.example.FooDataWriter.unregister_instance_w_params*

^ class **WriterDataLifecycleQosPolicy**

Controls how a ***com.rti.dds.publication.DataWriter*** *(*p. 538*) handles the lifecycle of the instances (keys) that it is registered to manage.*

^ class **WstringSeq**

Instantiates `com.rti.dds.util.Sequence` (p. 1432) *< char* ∗ *>.*

7.4.1 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.

7.5 Package com.rti.dds.publication

Contains the **com.rti.dds.publication.FlowController** (p. 942), **com.rti.dds.publication.Publisher** (p. 1277), and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the **publication** (p. 338) side.

Classes

- interface **DataWriter**

  <<**interface**>> (p. 271) *Allows an application to set the value of the data to be published under a given* ***com.rti.dds.topic.Topic*** *(p. 1545).*

- class **DataWriterAdapter**

  <<**eXtension**>> (p. 270) *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. 270) *The status of the writer's cache.*

- interface **DataWriterListener**

  <<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for writer status.*

- class **DataWriterProtocolStatus**

  <<**eXtension**>> (p. 270) *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. 538) entity.*

- interface **FlowController**

  <<**interface**>> (p. 271) *A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***com.rti.dds.publication.DataWriter*** *(p. 538) instances are allowed to write data.*

- class **FlowControllerProperty_t**

Determines the flow control characteristics of the ***com.rti.dds.publication.FlowController*** *(p. 942).*

ˆ class **FlowControllerSchedulingPolicy**

Kinds of flow controller scheduling policy.

ˆ class **FlowControllerTokenBucketProperty_t**

com.rti.dds.publication.FlowController *(p. 942) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.*

ˆ class **LivelinessLostStatus**

StatusKind.LIVELINESS_LOST_STATUS.

ˆ class **OfferedDeadlineMissedStatus**

StatusKind.OFFERED_DEADLINE_MISSED_STATUS.

ˆ class **OfferedIncompatibleQosStatus**

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS.

ˆ class **PublicationMatchedStatus**

StatusKind.PUBLICATION_MATCHED_STATUS.

ˆ interface **Publisher**

<<**interface**>> (p. 271) *A publisher is the object responsible for the actual dissemination of publications.*

ˆ class **PublisherAdapter**

<<**eXtension**>> (p. 270) *A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*

ˆ interface **PublisherListener**

<<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for* ***com.rti.dds.publication.Publisher*** *(p. 1277) status.*

ˆ class **PublisherQos**

QoS policies supported by a ***com.rti.dds.publication.Publisher*** *(p. 1277) entity.*

ˆ class **PublisherSeq**

Declares IDL `sequence` < ***com.rti.dds.publication.Publisher*** *(p. 1277)* > .

- class **ReliableReaderActivityChangedStatus**

  <<**eXtension**>> (p. 270) *Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*

- class **ReliableWriterCacheChangedStatus**

  <<**eXtension**>> (p. 270) *A summary of the state of a data writer's cache of unacknowledged samples written.*

- class **ReliableWriterCacheEventCount**

  <<**eXtension**>> (p. 270) *The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.*

Packages

- package **builtin**

  *Builtin **topic** (p. 350) for accessing information about the Publications discovered by RTI Connext.*

- package **example**

7.5.1 Detailed Description

Contains the **com.rti.dds.publication.FlowController** (p. 942), **com.rti.dds.publication.Publisher** (p. 1277), and **com.rti.dds.publication.DataWriter** (p. 538) classes as well as the **com.rti.dds.publication.PublisherListener** (p. 1302) and **com.rti.dds.publication.DataWriterListener** (p. 566) interfaces, and more generally, all that is needed on the **publication** (p. 338) side.

7.6 Package com.rti.dds.publication.builtin

Builtin **topic** (p. 350) for accessing information about the Publications discovered by RTI Connext.

Classes

- ˆ class **PublicationBuiltinTopicData**

  *Entry created when a **com.rti.dds.publication.DataWriter** (p. 538) is discovered in association with its **Publisher** (p. 1277).*

- ˆ class **PublicationBuiltinTopicDataDataReader**

  Instantiates `DataReader` < ***builtin.PublicationBuiltinTopicData*** *(p. 1264)* > .

- ˆ class **PublicationBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***builtin.PublicationBuiltinTopicData*** *(p. 1264)* > .

- ˆ class **PublicationBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` < ***builtin.PublicationBuiltinTopicData*** *(p. 1264)* > .

7.6.1 Detailed Description

Builtin **topic** (p. 350) for accessing information about the Publications discovered by RTI Connext.

7.7 Package com.rti.dds.publication.example

Classes

- interface **FooDataWriter**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific data writer.*

7.7.1 Detailed Description

Describes **FooDataWriter** (p. 1040), where Foo represents a user-defined data-type intended to be distributed using DDS.

7.8 Package com.rti.dds.subscription

Contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the **subscription** (p. 343) side.

Classes

- interface **DataReader**

  <<**interface**>> (p. 271) *Allows the application to: (1) declare the data it wishes to receive (i.e. make a* ***subscription*** *(p. 343)) and (2) access the data received by the attached* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- class **DataReaderAdapter**

  <<**eXtension**>> (p. 270) *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. 270) *The status of the reader's cache.*

- interface **DataReaderListener**

  <<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for reader status.*

- class **DataReaderProtocolStatus**

  <<**eXtension**>> (p. 270) *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. 473) entity.*

- class **DataReaderSeq**

  Declares IDL `sequence` < ***com.rti.dds.subscription.DataReader*** *(p. 473)* > .

- class **InstanceStateKind**

Indicates is the samples are from a live ***com.rti.dds.publication.DataWriter*** *(p. 538) or not.*

ˆ class **LivelinessChangedStatus**

StatusKind.LIVELINESS_CHANGED_STATUS.

ˆ interface **QueryCondition**

<<**interface**>> (p. 271) *These are specialised* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326) objects that allow the application to also specify a filter on the locally available data.*

ˆ interface **ReadCondition**

<<**interface**>> (p. 271) *Conditions specifically dedicated to read operations and attached to one* ***com.rti.dds.subscription.DataReader*** *(p. 473).*

ˆ class **RequestedDeadlineMissedStatus**

StatusKind.REQUESTED_DEADLINE_MISSED_STATUS.

ˆ class **RequestedIncompatibleQosStatus**

StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS.

ˆ class **SampleInfo**

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

ˆ class **SampleInfoSeq**

Declares IDL `sequence` *<* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404) > .*

ˆ class **SampleLostStatus**

StatusKind.SAMPLE_LOST_STATUS_STATUS.

ˆ class **SampleLostStatusKind**

Kinds of reasons why a sample was lost.

ˆ class **SampleRejectedStatus**

StatusKind.SAMPLE_REJECTED_STATUS.

ˆ class **SampleRejectedStatusKind**

Kinds of reasons for rejecting a sample.

ˆ class **SampleStateKind**

Indicates whether or not a sample has ever been read.

- interface **Subscriber**

  <<**interface**>> (p. 271) *A subscriber is the object responsible for actually receiving data from a* ***subscription*** *(p. 343).*

- 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. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for status about a subscriber.*

- class **SubscriberQos**

  QoS policies supported by a ***com.rti.dds.subscription.Subscriber*** *(p. 1478) entity.*

- class **SubscriberSeq**

  Declares IDL `sequence <` ***com.rti.dds.subscription.Subscriber*** *(p. 1478)* `>` .

- class **SubscriptionMatchedStatus**

  StatusKind.SUBSCRIPTION_MATCHED_STATUS.

- class **ViewStateKind**

  Indicates whether or not an instance is new.

Packages

- package **builtin**

  Builtin ***topic*** *(p. 350) for accessing information about the Subscriptions discovered by RTI Connext.*

- package **example**

7.8.1 Detailed Description

Contains the **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.subscription.ReadCondition** (p. 1326), and **com.rti.dds.subscription.QueryCondition** (p. 1324) classes, as well as the **com.rti.dds.subscription.SubscriberListener** (p. 1504) and **com.rti.dds.subscription.DataReaderListener** (p. 501) interfaces, and more generally, all that is needed on the **subscription** (p. 343) side.

7.8.2 Access to data samples

Data is made available to the application by the following operations on **com.rti.dds.subscription.DataReader** (p. 473) objects: com.rti.dds.topic.example.FooDataReader.read, com.rti.dds.topic.example.FooDataReader.read_w_condition, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.take_w_condition, 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. 1404) 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. 75) QoS allow for it.

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.dds.topic.example.FooDataReader.return_loan.

See also:

Interpretation of the SampleInfo (p. 1405)

7.8.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **com.rti.dds.subscription.DataReader** (p. 473). These operations return an ordered collection of `DataSamples` consisting of a **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 473) and **com.rti.dds.subscription.Subscriber** (p. 1478), 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. 1430))

- the desired view states (any combination of **com.rti.dds.subscription.ViewStateKind** (p. 1689))
- the desired instance states (any combination of **com.rti.dds.subscription.InstanceStateKind** (p. 1086))

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. 1326) 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 true. These operations, in conjunction with **com.rti.dds.subscription.ReadCondition** (p. 1326) objects and a **com.rti.dds.infrastructure.WaitSet** (p. 1695), 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. 86) QoS must be set properly.

7.9 Package com.rti.dds.subscription.builtin

Builtin **topic** (p. 350) for accessing information about the Subscriptions discovered by RTI Connext.

Classes

- class **SubscriptionBuiltinTopicData**

  *Entry created when a **com.rti.dds.subscription.DataReader** (p. 473) is discovered in association with its **Subscriber** (p. 1478).*

- class **SubscriptionBuiltinTopicDataDataReader**

  Instantiates `DataReader` (p. 473) < ***builtin.SubscriptionBuiltinTopicData*** *(p. 1510)* > .

- class **SubscriptionBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***builtin.SubscriptionBuiltinTopicData*** *(p. 1510)* > .

- class **SubscriptionBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` < ***builtin.SubscriptionBuiltinTopicData*** *(p. 1510)* > .

7.9.1 Detailed Description

Builtin **topic** (p. 350) for accessing information about the Subscriptions discovered by RTI Connext.

7.10 Package com.rti.dds.subscription.example

Classes

- interface **FooDataReader**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type-specific data reader.*

7.10.1 Detailed Description

Describes **FooDataReader** (p. 988), where Foo represents a user-defined data-type intended to be distributed using DDS.

7.11 Package com.rti.dds.topic

Contains the **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) classes, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.

Classes

- ˆ class **BuiltinTopicKey_t**

  *The key type of the built-in **topic** (p. 350) types.*

- ˆ interface **ContentFilter**

  <<**interface**>> (p. 271) *Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458)*

- ˆ interface **ContentFilteredTopic**

  <<**interface**>> (p. 271) *Specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) that allows for content-based subscriptions.*

- ˆ class **InconsistentTopicStatus**

  StatusKind.INCONSISTENT_TOPIC_STATUS.

- ˆ interface **MultiTopic**

  *[**Not supported (optional)**]* <<**interface**>> (p. 271) *A specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) that allows subscriptions that combine/filter/rearrange data coming from several topics.*

- ˆ interface **Topic**

  <<**interface**>> (p. 271) *The most basic description of the data to be published and subscribed.*

- ˆ class **TopicAdapter**

  <<**eXtension**>> (p. 270) *A listener adapter in the spirit of the Java AWT listener adapters. (The Adapter provides empty implementations for the listener methods)*

- ˆ interface **TopicDescription**

  ***com.rti.dds.topic.Topic** (p. 1545) entity and associated elements*

- ˆ interface **TopicListener**

<<**interface**>> (p. 271) ***com.rti.dds.infrastructure.Listener*** *(p. 1154) for* ***com.rti.dds.topic.Topic*** *(p. 1545) entities.*

- class **TopicQos**

  QoS policies supported by a ***com.rti.dds.topic.Topic*** *(p. 1545) entity.*

- interface **TypeSupport**

  <<**interface**>> (p. 271) *An abstract* marker *interface that has to be specialized for each concrete user data type that will be used by the application.*

Packages

- package **builtin**

  Builtin ***topic*** *(p. 350) for accessing information about the Topics discovered by RTI Connext.*

- package **example**

  Descriptions of ***Foo*** *(p. 955),* ***FooSeq*** *(p. 1056), and* ***FooTypeSupport*** *(p. 1060), where* ***Foo*** *(p. 955) represents a user-defined data-type intended to be distributed using DDS.*

7.11.1 Detailed Description

Contains the **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) classes, the **com.rti.dds.topic.TopicListener** (p. 1564) interface, and more generally, all that is needed by an application to define **com.rti.dds.topic.Topic** (p. 1545) objects and attach QoS policies to them.

7.12 Package com.rti.dds.topic.builtin

Builtin **topic** (p. 350) for accessing information about the Topics discovered by RTI Connext.

Classes

- ˆ class **AbstractBuiltinTopicDataTypeSupport**
- ˆ class **TopicBuiltinTopicData**

  Entry created when a ***Topic*** *(p. 1545) object discovered.*

- ˆ class **TopicBuiltinTopicDataDataReader**

  Instantiates `DataReader` < ***builtin.TopicBuiltinTopicData*** *(p. 1552)* > .

- ˆ class **TopicBuiltinTopicDataSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***builtin.TopicBuiltinTopicData*** *(p. 1552)* > .

- ˆ class **TopicBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` (p. 1651) < ***builtin.TopicBuiltinTopicData*** *(p. 1552)* > .

7.12.1 Detailed Description

Builtin **topic** (p. 350) for accessing information about the Topics discovered by RTI Connext.

7.13 Package com.rti.dds.topic.example

Descriptions of **Foo** (p. 955), **FooSeq** (p. 1056), and **FooTypeSupport** (p. 1060), where **Foo** (p. 955) represents a user-defined data-type intended to be distributed using DDS.

Classes

- ^ class **Foo**

  A representative user-defined data type.

- ^ class **FooSeq**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 955).

- ^ class **FooTypeSupport**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific interface.*

7.13.1 Detailed Description

Descriptions of **Foo** (p. 955), **FooSeq** (p. 1056), and **FooTypeSupport** (p. 1060), where **Foo** (p. 955) represents a user-defined data-type intended to be distributed using DDS.

7.14 Package com.rti.dds.type.builtin

<<*eXtension*>> *(p. 270)* RTI Connext provides a set of very simple data types for you to use with the topics in your application.

Classes

- class **Bytes**

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

- class **BytesDataReader**

  <<**interface**>> (p. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* >.

- class **BytesDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* >.

- class **BytesSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.Bytes*** *(p. 417)* > .

- class **BytesTypeSupport**

  <<**interface**>> (p. 271) ***com.rti.dds.type.builtin.Bytes*** *(p. 417) 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. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- class **KeyedBytesDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- class **KeyedBytesSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095)* >.

- class **KeyedBytesTypeSupport**

<<**interface**>> (p. 271) ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) type support.*

- class **KeyedString**

  Keyed string built-in type.

- class **KeyedStringDataReader**

  <<**interface**>> (p. 271) *Instantiates* `DataReader` < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* >.

- class **KeyedStringDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* >.

- class **KeyedStringSeq**

  Instantiates `com.rti.dds.util.Sequence` (p. 1432) < ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123)* > .

- class **KeyedStringTypeSupport**

  <<**interface**>> (p. 271) *Keyed string type support.*

- class **StringDataReader**

  <<**interface**>> (p. 271) *Instantiates* `DataReader` < *String* >.

- class **StringDataWriter**

  <<**interface**>> (p. 271) *Instantiates* `DataWriter` < *String* >.

- class **StringTypeSupport**

  <<**interface**>> (p. 271) *String type support.*

7.14.1 Detailed Description

<<***eXtension***>> *(p. 270)* 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.KeyedString** (p. 1123): 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.Bytes** (p. 417): A payload consisting of an opaque variable-length array of bytes. This type has no key.
- **com.rti.dds.type.builtin.KeyedBytes** (p. 1095): 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.KeyedString** (p. 1123) types are appropriate for simple text-based applications. The **com.rti.dds.type.builtin.Bytes** (p. 417) and **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) 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. 458)).

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** (p. 290) utility
- At runtime, by using the **Dynamic Data** (p. 170) API

7.14.2 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** (p. 354) types, the maximum size of the buffers/samples and depends on the nature of the application using the **builtin** (p. 354) type.

You can configure the maximum size of the **builtin** (p. 354) types on a per-DataWriter and per-DataReader basis using the **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) in DataWriters, DataReaders or Participants.

The following table lists the supported **builtin** (p. 354) 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 DomainParticipant unless they are overwrittem in the DataWriters and DataReaders.

The previous properties must be set consistently with respect to the corresponding ∗.max_size properties that set the maximum size of the **builtin** (p. 354) types in the **typecode** (p. 360).

7.14.3 Typecodes for Builtin Types

The typecodes associated with the **builtin** (p. 354) 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.

For more information about the built-in types, including how to control memory usage and maximum lengths, please see chapter 3, *Data Types and Data Samples*, in the RTI Connext User's Manual.

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the **com.rti.dds.type.builtin.StringDataWriter** (p. 1468) or received the **com.rti.dds.type.builtin.StringDataReader** (p. 1465) (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.KeyedStringDataWriter** (p. 1133) or **com.rti.dds.type.builtin.KeyedStringDataReader** (p. 1125) (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.KeyedStringDataWriter** (p. 1133) or received by the **com.rti.dds.type.builtin.KeyedStringDataReader** (p. 1125) (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.BytesDataWriter** (p. 424) or received by the **com.rti.dds.type.builtin.BytesDataReader** (p. 420). Default: dds.builtin_type.octets.max_size if defined. Otherwise, 2048.
dds.builtin_type.keyed_octets.alloc_-key_size	Max-imum size of the key published by the **com.rti.dds.type.builtin.KeyedBytesDataWriter** (p. 1106) or received by the **com.rti.dds.type.builtin.KeyedBytesDataReader** (p. 1098) (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.KeyedBytesDataWriter** (p. 1106) or received by a **com.rti.dds.type.builtin.KeyedBytesDataReader** (p. 1098).

Property	Description
dds.builtin_type.string.max_size	Maximum size of the strings published by the StringDataWriters and received by the StringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_key_size	Maximum size of the keys used by the KeyedStringDataWriters and KeyedStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_string.max_size	Maximum size of the strings published by the KeyedStringDataWriters and received by the KeyedStringDataReaders belonging to a DomainParticipant using the **builtin** (p. 354) type (includes the NULL-terminated character). Default: 1024
dds.builtin_type.octets.max_size	Maximum size of the octet sequences published by the OctetsDataWriters and received by the OctetsDataReader belonging to a DomainParticipant. Default: 2048
dds.builtin_type.keyed_octets.max_key_size	Maximum size of the keys used by the KeyedOctetsStringDataWriters and KeyedOctetsStringDataReaders belonging to a DomainParticipant (includes the NULL-terminated character). Default: 1024.
dds.builtin_type.keyed_octets.max_size	Maximum size of the octet sequences published by the KeyedOctetsDataWriters and received by the KeyedOctetsDataReaders belonging to a DomainParticipant. Default: 2048

Table 7.2: *Properties for Allocating Size of Builtin Types, per DomainParticipant*

7.15 Package com.rti.dds.typecode

*<<**eXtension**>>* *(p. 270)* A ***TypeCode*** *(p. 1611)* 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. 170) 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 **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 ***TypeCode*** *(p. 1611) 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*** *(p. 290) 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.*

7.15.1 Detailed Description

<<***eXtension***>> *(p. 270)* A ***TypeCode*** *(p. 1611)* 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. 170) 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 **TypeCode** (p. 1611) class, which has an analogue in CORBA.

A **TypeCode** (p. 1611) value consists of a type code *kind* (represented by the **TCKind** (p. 1526) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **TypeCode** (p. 1611), 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 **TypeCode.equals** (p. 1616) method is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

7.15.2 Accessing a Local TypeCode

When generating types with **rtiddsgen** (p. 290), type codes are enabled by default. (The *-notypecode* option can be used to disable generation of **TypeCode** (p. 1611) information.) For these types, a **TypeCode** (p. 1611) may be accessed via the **FooTypeCode.VALUE** member.

This API also includes support for dynamic creation of **TypeCode** (p. 1611) values, typically for use with the **Dynamic Data** (p. 170) API. You can create a **TypeCode** (p. 1611) using the **TypeCodeFactory** (p. 1641) class. You will construct the **TypeCode** (p. 1611) 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:

- **TypeCodeFactory.get_primitive_tc** (p. 1650), which provides the **TypeCode** (p. 1611) instances corresponding to the primitive types (e.g. **TCKind.TK_LONG** (p. 1528), **TCKind.TK_SHORT** (p. 1528), and so on).
- **TypeCodeFactory.create_string_tc** (p. 1647) and **TypeCodeFactory.create_wstring_tc** (p. 1648) create a **TypeCode** (p. 1611) representing a text string with a certain *bound* (i.e. maximum length).
- **TypeCodeFactory.create_array_tc** (p. 1649) and **TypeCodeFactory.create_sequence_tc** (p. 1648) create a **TypeCode** (p. 1611) for a collection based on the **TypeCode** (p. 1611) for its elements.
- **TypeCodeFactory.create_struct_tc** (p. 1644), **TypeCodeFactory.create_value_tc** (p. 1644), and **TypeCodeFactory.create_sparse_tc** (p. 1645) create a **TypeCode** (p. 1611) for a structured type.

7.15.3 Accessing a Remote TypeCode

In addition to being used locally, RTI Connext can transmit **TypeCode** (p. 1611) 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. 170)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote **TypeCode** (p. 1611) information is shared during discovery over the **publication** (p. 338) and **subscription** (p. 343) built-in topics and can be accessed using the built-in readers for these topics; see **Built-in Topics** (p. 153). Discovered **TypeCode** (p. 1611) values are not cached by RTI Connext upon receipt and are therefore not available from the built-in **topic** (p. 350) data returned by **com.rti.dds.publication.DataWriter.get_matched_subscription_data** (p. 551) or **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 487).

The space available locally to deserialize a discovered remote **TypeCode** (p. 1611) is specified by the **com.rti.dds.domain.DomainParticipant** (p. 629)'s **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.type_code_max_serialized_length** (p. 755) QoS parameter. To support especially complex type codes, it may be necessary for you to increase the value of this parameter.

See also:

TypeCode (p. 1611)

Dynamic Data (p. 170)
rtiddsgen (p. 290)
builtin.SubscriptionBuiltinTopicData
builtin.PublicationBuiltinTopicData

7.16 Package com.rti.dds.util

Utility types that support the DDS API.

Classes

- ˆ class **AbstractPrimitiveSequence**
- ˆ class **AbstractSequence**

  Abstract sequence.

- ˆ class **Enum**

  A superclass for all type-safe enumerated types.

- ˆ class **LoanableSequence**

  A sequence capable of storing its elements directly or taking out a loan on them from an internal middleware store.

- ˆ interface **Sequence**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo`*.*

- ˆ class **Union**

7.16.1 Detailed Description

Utility types that support the DDS API.

7.17 Package com.rti.ndds.config

Utility API's independent of the DDS standard.

Classes

- class **LibraryVersion_t**

  The version of a single library shipped as part of an RTI Connext distribution.

- class **LogCategory**

  Categories of logged messages.

- class **Logger**

  <<**interface**>> (p. 271) *The singleton type used to configure RTI Connext logging.*

- class **LogPrintFormat**

  The format used to output RTI Connext diagnostic information.

- class **LogVerbosity**

  The verbosities at which RTI Connext diagnostic information is logged.

- class **Version**

  <<**interface**>> (p. 271) *The version of an RTI Connext distribution.*

7.17.1 Detailed Description

Utility API's independent of the DDS standard.

7.18 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. 271) <<**generic**>> (p. 271) *User data type-specific data reader.*

- class **FooDataWriter**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *User data type specific data writer.*

- class **FooSeq**

  <<**interface**>> (p. 271) <<**generic**>> (p. 271) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 956).

- class **FooTypeSupport**

7.18.1 Detailed Description

Programming HowTos: Code templates for common use cases.

7.19 Package com.rti.ndds.transport

APIs related to RTI Connext pluggable transports.

Classes

- interface **ShmemTransport**

  *Built-in **transport** (p. 367) plug-in for inter-process communications using shared memory.*

- interface **Transport**

  *RTI Connext's abstract pluggable **transport** (p. 367) interface.*

- class **TransportSupport**

  <<**interface**>> (p. 271) *The utility class used to configure RTI Connext pluggable transports.*

- interface **UDPv4Transport**

  *Built-in **transport** (p. 367) plug-in using UDP/IPv4.*

- interface **UDPv6Transport**

  *Built-in **transport** (p. 367) plug-in using UDP/IPv6.*

7.19.1 Detailed Description

APIs related to RTI Connext pluggable transports.

7.19.2 Overview

RTI Connext has a pluggable transports architecture. The core of RTI Connext is **transport** (p. 367) 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 (p. 367) plugins** which implement that API.

A **transport** (p. 367) plugin implements the abstract **transport** (p. 367) API and performs the actual work of sending and receiving messages over a physical **transport** (p. 367). A collection of **builtin plugins** (see **Built-in Transport Plugins** (p. 216)) is delivered with RTI Connext for commonly used transports. New **transport** (p. 367) 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** (p. 367) plugins to be used in an application. So that RTI Connext applications work out of the box, a subset of the builtin **transport** (p. 367) plugins is *preconfigured* by default (see **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580)). You can "turn-off" some or all of the builtin **transport** (p. 367) plugins. In addition, you can configure other **transport** (p. 367) plugins for use by the application.

7.19.3 Transport Aliases

In order to use a **transport** (p. 367) plugin instance in an RTI Connext application, it must be registered with a **com.rti.dds.domain.DomainParticipant** (p. 629). When you register a **transport** (p. 367), you specify a sequence of "alias" strings to symbolically refer to the **transport** (p. 367) plugin. The same alias strings can be used to register more than one **transport** (p. 367) plugin.

You can register multiple **transport** (p. 367) plugins with a **com.rti.dds.domain.DomainParticipant** (p. 629). An **alias** symbolically refers to one or more **transport** (p. 367) plugins registered with the **com.rti.dds.domain.DomainParticipant** (p. 629). Builtin **transport** (p. 367) plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 115)).

A **transport** (p. 367) plugin's class name is automatically used as an implicit alias. It can be used to refer to all the **transport** (p. 367) plugin instances of that class.

You can use aliases to refer to **transport** (p. 367) plugins, in order to specify:

- the **transport** (p. 367) plugins to use for **discovery** (see **com.rti.dds.infrastructure.DiscoveryQosPolicy.enabled_transports** (p. 625)), and for **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) entities (see **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1600)).

- the **multicast** addresses on which to receive discovery messages (see **com.rti.dds.infrastructure.DiscoveryQosPolicy.multicast_receive_addresses** (p. 625)), and the multicast addresses and ports on which to receive user data (see **com.rti.dds.subscription.DataReaderQos.multicast** (p. 522)).

- the **unicast ports** used for user data (see **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1605)) on both **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) entities.

- the **transport** (p. 367) plugins used to parse an address string in a locator (**Locator Format** (p. 56) and **NDDS_DISCOVERY_PEERS** (p. 55)).

A **com.rti.dds.domain.DomainParticipant** (p. 629) (and contained its entities) start using a **transport** (p. 367) plugin after the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled (see **com.rti.dds.infrastructure.Entity.enable** (p. 915)). An entity will use *all* the **transport** (p. 367) plugins that match the specified **transport** (p. 367) QoS policy. All **transport** (p. 367) plugins are treated uniformly, regardless of how they were created or registered; there is no notion of some transports being more "special" that others.

7.19.4 Transport Lifecycle

A **transport** (p. 367) plugin is owned by whoever created it. Thus, if you create and register a **transport** (p. 367) plugin with a **com.rti.dds.domain.DomainParticipant** (p. 629), you are responsible for deleting it by calling its destructor. Note that builtin **transport** (p. 367) plugins (**TRANSPORT_BUILTIN** (p. 115)) and **transport** (p. 367) plugins that are loaded through the **PROPERTY** (p. 88) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 213)) are automatically managed by RTI Connext.

A user-created **transport** (p. 367) plugin must not be deleted while it is still in use by a **com.rti.dds.domain.DomainParticipant** (p. 629). This generally means that a user-created **transport** (p. 367) plugin instance can only be deleted after the **com.rti.dds.domain.DomainParticipant** (p. 629) with which it was registered is deleted (see **com.rti.dds.domain.DomainParticipantFactory.delete_participant** (p. 715)). Note that a **transport** (p. 367) plugin *cannot* be "unregistered" from a **com.rti.dds.domain.DomainParticipant** (p. 629).

A **transport** (p. 367) plugin instance cannot be registered with more than one **com.rti.dds.domain.DomainParticipant** (p. 629) at a time. This requirement is necessary to guarantee the multi-threaded safety of the **transport** (p. 367) API.

If the same physical **transport** (p. 367) resources are to be used with more than one **com.rti.dds.domain.DomainParticipant** (p. 629) in the same address space, the **transport** (p. 367) 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. 629) in the address space. Note that it is always possible to write the **transport** (p. 367) plugin so that multiple **transport** (p. 367) 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** (p. 367) plugin developer.

7.19.5 Transport Class Attributes

A **transport** (p. 367) 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** (p. 367) plugin class, and

- the *instance* attributes that can be set on a per instance basis by the **transport** (p. 367) plugin user.

Every **transport** (p. 367) plugin must specify the following class attributes.

transport (p. 367) class id (see Transport.Property_t.classid (p. 1573))
Identifies a **transport** (p. 367) plugin implementation class. It denotes a unique "class" to which the **transport** (p. 367) plugin instance belongs. The class is used to distinguish between different **transport** (p. 367) plugin implementations. Thus, a **transport** (p. 367) plugin vendor should ensure that its **transport** (p. 367) plugin implementation has a unique class.

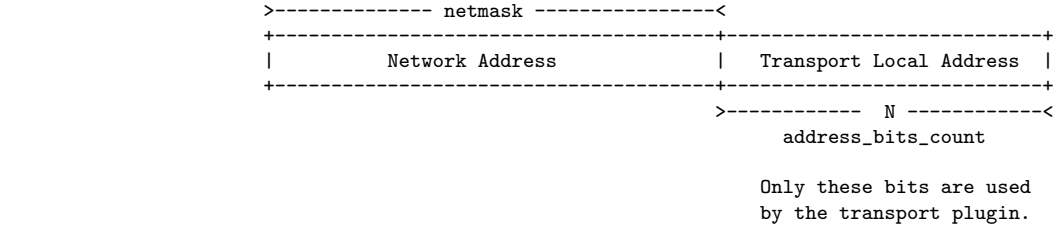
Two **transport** (p. 367) plugin instances report the same class *iff* they have compatible implementations. **Transport** (p. 1569) plugin instances with mismatching classes are not allowed (by the RTI Connext Core) to communicate with one another.

Multiple implementations (possibly from different vendors) for a physical **transport** (p. 367) mechanism can co-exist in an RTI Connext application, provided they use different **transport** (p. 367) class IDs.

The class ID can also be used to distinguish between different **transport** (p. 367) protocols over the same physical **transport** (p. 367) network (e.g., UDP vs. TCP over the IP routing infrastructure).

transport (p. 367) significant address bit count (see Transport.Property_t.address_bit_cou
RTI Connext's addressing is modeled after the IPv6 and uses 128-bit addresses (java.net.InetAddress) to route messages.

A **transport** (p. 367) plugin is expected to map the transport's internal addressing scheme to 128-bit addresses. In general, this mapping is likely to use only `N` least significant bits (LSB); these are specified by this attribute.



The remaining bits of an address using the 128-bit address representation will be considered as part of the "network address" (see **Transport Network Address** (p. 371)) and thus ignored by the **transport** (p. 367) plugin's internal addressing scheme.

For *unicast* addresses, the **transport** (p. 367) plugin is expected to ignore the higher (128 - **Transport.Property_t.address_bit_count** (p. 1573)) bits. RTI Connext is free to manipulate those bits freely in the addresses passed in/out to the **transport** (p. 367) plugin APIs.

Theoretically, the significant address bits count, `N` is related to the size of the underlying **transport** (p. 367) 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.

7.19.6 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** (p. 367) plugin writer, and can be used to:

- customize the behavior of an instance of a **transport** (p. 367) plugin, including the send and the receiver buffer sizes, the maximum message size, various **transport** (p. 367) level classes of service (CoS), and so on.

- specify the resource values, network interfaces to use, various **transport** (p. 367) level policies, and so on.

RTI Connext requires that every **transport** (p. 367) plugin instance must specify the **Transport.Property_t.message_size_max** (p. 1574) and **Transport.Property_t.gather_send_buffer_count_max** (p. 1574).

It is up to the **transport** (p. 367) plugin developer to make these available for configuration to **transport** (p. 367) 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** (p. 367) plugin class. For **example** (p. 366), if one side is configured to send messages larger than can be received by the other side, then communications via the plugin may fail.

7.19.7 Transport Network Address

The address bits not used by the **transport** (p. 367) plugin for its internal addressing constitute its network address bits.

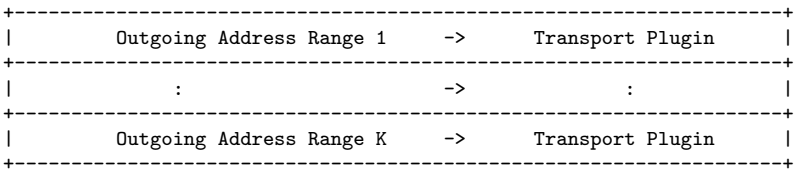
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a **transport** (p. 367) plugin via the TransportSupport.register_transport() API.

The network address for a **transport** (p. 367) 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** (p. 367) plugin are registered with a **com.rti.dds.domain.DomainParticipant** (p. 629), 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** (p. 367)).

7.19.8 Transport Send Route

By default, a **transport** (p. 367) 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 TransportSupport.add_send_route() API, so that a **transport** (p. 367) plugin will be used to send messages only to certain ranges of destination addresses. The method can be called multiple times for a **transport** (p. 367) plugin, with different address ranges.



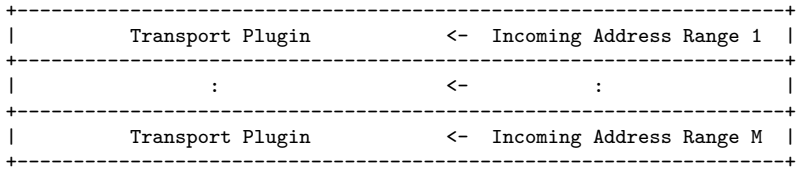
The user can set up a routing table to restrict the use of a **transport** (p. 367) plugin to send messages to selected addresses ranges.

7.19.9 Transport Receive Route

By default, a **transport** (p. 367) 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 TransportSupport.add_receive_route() API, so that a **transport** (p. 367) plugin will be used to receive messages only on certain ranges of addresses.

The method can be called multiple times for a **transport** (p. 367) plugin, with different address ranges.



The user can set up a routing table to restrict the use of a **transport** (p. 367) plugin to receive messages from selected ranges. For **example** (p. 366), the user may restrict a **transport** (p. 367) 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** (p. 367) plugin).

Chapter 8

Class Documentation

8.1 AbstractBuiltinTopicDataTypeSupport Class Reference

Inheritance diagram for AbstractBuiltinTopicDataTypeSupport::

Protected Member Functions

- ^ final void **initialize_delegateI** (DataReaderDelegate delegate)

8.1.1 Detailed Description

Abstract superclass for all *TypeSupport classes for built-in types.

Author:

rwarren

Version:

Revision

1.17

Date

2009/11/01 18:04:49

8.1.2 Member Function Documentation

8.1.2.1 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 final Class getElementType ()

Returns:

the primitive type of this sequence, not the wrapper type.

See also:

com.rti.dds.util.Sequence.getElementType() (p. 1434)

Reimplemented from **AbstractSequence** (p. 383).

8.2.2.2 abstract void add (int *index*, Object *element*) [pure virtual]

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. 383).

Implemented in **BooleanSeq** (p. 410), **ByteSeq** (p. 433), **CharSeq** (p. 450), **DoubleSeq** (p. 764), **FloatSeq** (p. 941), **IntSeq** (p. 1094), **LongSeq** (p. 1204), and **ShortSeq** (p. 1451).

8.2.2.3 void loan (Object *buffer*, int *new_length*)

Loan a contiguous buffer to this sequence.

This operation changes the `owned` flag of the sequence to 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 **Sequence.setMaximum** (p. 1433).

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.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.return_loan. See the documentation of those methods for more information.

Precondition:

Sequence.getMaximum (p. 1433) == 0; i.e. the sequence has no memory allocated to it.
AbstractPrimitiveSequence.hasOwnership (p. 380) == true; i.e. the sequence does not already have an outstanding loan

Postcondition:

The sequence will store its elements in the buffer provided.

AbstractPrimitiveSequence.hasOwnership (p. 380) == false
Sequence.size() == new_length
Sequence.getMaximum (p. 1433) == 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.

Returns:

true if `buffer` is successfully loaned to this sequence or false otherwise. Failure only occurs due to failing to meet the pre-conditions. Upon failure the sequence remains unmodified.

See also:

com.rti.dds.util.Sequence.unloan, com.rti.dds.util.Sequence.loan_discontiguous

8.2.2.4 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 == false

Postcondition:

owned == true
maximum == 0

Returns:

true if the preconditions were met. Otherwise false. The function only fails if the pre-conditions are not met, in which case it leaves the sequence unmodified.

See also:

AbstractPrimitiveSequence.loan(Object, int) (p. 378), com.rti.dds.util.Sequence.loan_discontiguous, **Sequence.setMaximum** (p. 1433)

8.2.2.5 final boolean hasOwnership ()

Return the value of the owned flag.

Returns:

true if sequence owns the underlying buffer, or false if it has an outstanding loan.

8.2.2.6 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()

8.2.2.7 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. 1433)

8.2.2.8 final int size ()

The logical size of this sequence.

8.2.2.9 final Object copy_from (Object *src*)

Implementation of the `Copyable` interface.

Parameters:

src An `AbstractPrimitiveSequence` (p. 377) which contains the data to be copied.

Returns:

`this`

Exceptions:

NullPointerException If `src` is null.

ClassCastException If `src` is not a `Sequence` (p. 1432) OR if one of the objects contained in the `Sequence` (p. 1432) is not of the expected type.

Implements **Copyable** (p. 466).

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 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. 383), the sequence's size is increased implicitly.

Postcondition:

length == MINIMUM(original length, new_max)

Parameters:

new_max Must be >= 0.

Returns:

true on success, false if the preconditions are not met. In that case the sequence is not modified.

Implements **Sequence** (p. 1433).

Reimplemented in **LoanableSequence** (p. 1172).

8.3.2.2 Class getElementType ()

Returns:

a common supertype for all elements in this sequence.

Implements **Sequence** (p. 1434).

Reimplemented in **AbstractPrimitiveSequence** (p. 377).

8.3.2.3 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. 410), **ByteSeq** (p. 433), **CharSeq** (p. 450), **DoubleSeq** (p. 764), **FloatSeq** (p. 941), **IntSeq** (p. 1094), **LongSeq** (p. 1204), **ShortSeq** (p. 1451), and **AbstractPrimitiveSequence** (p. 378).

8.3.2.4 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 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 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.

8.4.1 Detailed Description

Resource allocation settings.

QoS:

com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy (p. 741)

8.4.2 Constructor & Destructor Documentation

8.4.2.1 AllocationSettings_t (int *initial_count*, int *max_count*, int *incremental_count*)

Constructor with the given initial, maximum and incremental values.

8.4.3 Member Data Documentation

8.4.3.1 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)

8.4.3.2 int max_count

The maximum count of resources.

The maximum resources to be allocated.

[default] Depends on the case.

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), > initial_count (or = initial_count only if increment_count == 0)

8.4.3.3 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)

8.5 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.

8.5.1 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. 1277) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **com.rti.dds.publication.DataWriter** (p. 538) entities belonging to this **com.rti.dds.publication.Publisher** (p. 1277). If so, the publisher will spawn up to two threads, one for asynchronous publishing and one for asynchronous batch flushing.

See also:

com.rti.dds.infrastructure.BatchQosPolicy (p. 401).
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308).

Entity:

com.rti.dds.publication.Publisher (p. 1277)

Properties:

RxO (p. 97) = N/A

Changeable (p. 98) = **NO** (p. 98)

8.5.2 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. 1308) and a **com.rti.dds.publication.FlowController** (p. 942), 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 **PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1312) **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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. 538) entities, the instances must belong to different **com.rti.dds.publication.Publisher** (p. 1277) instances.

A **com.rti.dds.publication.Publisher** (p. 1277) must have asynchronous publishing enabled for its **com.rti.dds.publication.DataWriter** (p. 538) instances to write asynchronously.

A **com.rti.dds.publication.Publisher** (p. 1277) must have asynchronous batch flushing enabled in order to flush the batches of its **com.rti.dds.publication.DataWriter** (p. 538) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **com.rti.dds.publication.DataWriter** (p. 538) instance with batching enabled is created from this **com.rti.dds.publication.Publisher** (p. 1277).

8.5.3 Member Data Documentation

8.5.3.1 boolean disable_asynchronous_write

Disable asynchronous publishing.

If set to true, any **com.rti.dds.publication.DataWriter** (p. 538) created with **PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1312) will fail with **RETCODE_INCONSISTENT_POLICY** (p. 1367).

[**default**] false

8.5.3.2 final ThreadSettings_t thread

Settings of the publishing thread.

There is only one asynchronous publishing thread per **com.rti.dds.publication.Publisher** (p. 1277).

[**default**] priority below normal.

The actual value depends on your architecture:

For Windows: -2

For Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For Integrity: 80

For VxWorks: 110

For all others: OS default priority.

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For Integrity: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[**default**] mask = **ThreadSettingsKind.THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 112)

8.5.3.3 boolean disable_asynchronous_batch

Disable asynchronous batch flushing.

If set to true, any **com.rti.dds.publication.DataWriter** (p. 538) created with batching enabled will fail with **RETCODE_INCONSISTENT_POLICY** (p. 1367).

If **com.rti.dds.infrastructure.BatchQosPolicy.max_flush_delay** (p. 403) is different than com.rti.dds.infrastructure.Duration_t.INFINITE, **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.disable_asynchronous_batch** (p. 390) must be set false.

[default] false

8.5.3.4 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. 1277).

[default] priority below normal.

The actual value depends on your architecture:

For Windows: -2

For Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For Integrity: 80

For VxWorks: 110

For all others: OS default priority.

[default] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For Integrity: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[**default**] mask = **ThreadSettingsKind.THREAD_SETTINGS_KIND_MASK_DEFAULT** (p. 112)

8.6 AvailabilityQosPolicy Class Reference

Configures the availability of data.

Inheritance diagram for AvailabilityQosPolicy::

Public Attributes

- 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.

8.6.1 Detailed Description

Configures the availability of data.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = NO

8.6.2 Usage

This QoS policy is used in the context of the Collaborative DataWriters feature.

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 with RTI Persistence Service:** When a late-joining DataReader configured with **com.rti.dds.infrastructure.DurabilityQosPolicy** (p. 765) set to **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772) or **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771) joins a DDS **domain** (p. 317), 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 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. 1237) access_scope set to **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243). 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 **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243).

Each sample published in a DDS **domain** (p. 317) 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. 1241) is set to **PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS** (p. 1243) or **PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS** (p. 1243), the virtual HB contains information about the samples contained in the **com.rti.dds.publication.DataWriter** (p. 538) history.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is set to **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243), the virtual HB contains information about all DataWriters in the **com.rti.dds.publication.Publisher** (p. 1277).

The frequency at which virtual HBs are sent is controlled by the protocol parameters **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.virtual_heartbeat_period** (p. 1383) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.samples_per_virtual_heartbeat** (p. 1384).

8.6.3 Member Data Documentation

8.6.3.1 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.

A sample identified by (GUIDn, SNm) 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 GUIDn have announced potential availability of (GUIDn, SNm-1).
- The DataWriters for all the endpoint groups specified in **required_matched_endpoint_groups** (p. 395) have been discovered or **max_endpoint_availability_waiting_time** (p. 394) has expired.

[default] com.rti.dds.infrastructure.Duration_t.AUTO (com.rti.dds.infrastructure.Duration_t.INFINITE for **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243). Otherwise, 0 seconds)

[range] [0, com.rti.dds.infrastructure.Duration_t.INFINITE], com.rti.dds.infrastructure.Duration_t.AUTO

8.6.3.2 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).

The set of endpoint groups that are required to provide samples for a data source can be configured using **required_matched_endpoint_groups** (p. 395).

A non-consecutive sample identified by (GUIDn, SNm) cannot be delivered to the application unless DataWriters for all the endpoint groups in **required_matched_endpoint_groups** (p. 395) are discovered or this timeout expires.

[**default**] com.rti.dds.infrastructure.Duration_t.AUTO (com.rti.dds.infrastructure.Duration_t.INFINITE for **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243). Otherwise, 0 seconds)

[**range**] [0, com.rti.dds.infrastructure.Duration_t.INFINITE], com.rti.dds.infrastructure.Duration_t.AUTO

8.6.3.3 final EndpointGroupSeq required_matched_endpoint_groups

A sequence of endpoint groups.

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. 593).

[**default**] Empty sequence

8.7 BAD_PARAM Class Reference

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

Inheritance diagram for BAD_PARAM::

8.7.1 Detailed Description

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

8.8 BAD_TYPECODE Class Reference

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

Inheritance diagram for BAD_TYPECODE::

8.8.1 Detailed Description

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

8.9 BadKind Class Reference

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

Inheritance diagram for BadKind::

8.9.1 Detailed Description

The exception **BadKind** (p. 398) is thrown when an inappropriate operation is invoked on a TypeCode object.

8.10 BadMemberId Class Reference

The specified TypeCode member ID is invalid.

Inheritance diagram for BadMemberId::

8.10.1 Detailed Description

The specified TypeCode 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.BadMemberName (p. 400)

8.11 BadMemberName Class Reference

The specified TypeCode member name is invalid.

Inheritance diagram for BadMemberName::

8.11.1 Detailed Description

The specified TypeCode 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.BadMemberId (p. 399)

8.12 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.

8.12.1 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. 538)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.12.2 Member Data Documentation

8.12.2.1 boolean enable

Specifies whether or not batching is enabled.

[**default**] false

8.12.2.2 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,**ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)]

8.12.3 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_data_bytes** (p. 402) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.max_samples** (p. 402). For these two values to be consistent, they cannot be both **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

8.12.3.1 int max_samples

The maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[**default**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[**range**] [1,**ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)]

8.12.4 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_samples** (p. 402) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.max_data_bytes** (p. 402). For these two values to be consistent, they cannot be both **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

8.12.4.1 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.INFINITE

[**range**] [0,com.rti.dds.infrastructure.Duration_t.INFINITE]

8.12.5 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.max_flush_delay** (p. 403) must be consistent with **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.disable_asynchronous_batch** (p. 390) and **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404). If the delay is different than com.rti.dds.infrastructure.Duration_t.INFINITE, **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.disable_asynchronous_batch** (p. 390) must be set to false and **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404) must be set to true.

8.12.5.1 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.INFINITE, 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.INFINITE.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE

[range] [0,com.rti.dds.infrastructure.Duration_t.INFINITE]

8.12.6 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 403) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404). If **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404) is set to false, **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 403) must be set to com.rti.dds.infrastructure.Duration_t.INFINITE.

8.12.6.1 boolean thread_safe_write

Determines whether or not the write operation is thread safe.

If this parameter is set to true, multiple threads can call write on the **com.rti.dds.publication.DataWriter** (p. 538) concurrently.

[default] true

8.12.7 Consistency

The setting of **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404) must be consistent with **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 403). If **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404) is set to false, **com.rti.dds.infrastructure.BatchQosPolicy.source_timestamp_resolution** (p. 403) must be set to com.rti.dds.infrastructure.Duration_t.INFINITE.

8.13 BooleanSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 407*) that returns a java.lang.Boolean.*

- Object **set** (int index, Object element)

  A wrapper for ***setBoolean()*** *(*p. 408*).*

- void **add** (int index, Object element)

  A wrapper for addBoolean(int, int).

8.13.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < boolean >.

Instantiates:

<<***generic***>> *(*p. 271*)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

boolean
com.rti.dds.util.Sequence (p. 1432)

8.13.2 Constructor & Destructor Documentation

8.13.2.1 BooleanSeq ()

Constructs an empty sequence of booleans with an initial maximum of zero.

8.13.2.2 BooleanSeq (int *initialMaximum*)

Constructs an empty sequence of booleans with the given initial maximum.

8.13.2.3 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

8.13.3 Member Function Documentation

8.13.3.1 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.

8.13.3.2 boolean addAllBoolean (boolean[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.13.3.3 void addBoolean (boolean *element*)

Append the element to the end of the sequence.

8.13.3.4 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.

8.13.3.5 boolean getBoolean (int *index*)

Returns the boolean at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.13.3.6 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.

8.13.3.7 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.

8.13.3.8 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.

8.13.3.9 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. 410), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.13.3.10 Object get (int *index*) [virtual]

A wrapper for **getBoolean(int)** (p. 407) that returns a java.lang.Boolean.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.13.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setBoolean()** (p. 408).

Exceptions:

ClassCastException if the element is not of type Boolean.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.13.3.12 void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.14 Bounds Class Reference

A user exception thrown when a parameter is not within the legal bounds.

Inheritance diagram for Bounds::

8.14.1 Detailed Description

A user exception thrown when a parameter is not within the legal bounds.

8.15 BuiltinTopicKey_t Class Reference

The key type of the built-in **topic** (p. 350) types.

Inheritance diagram for BuiltinTopicKey_t::

Public Member Functions

- void **copy_from** (**BuiltinTopicKey_t** other)
- Object **copy_from** (Object other)

  Copy value of a data type from source.

Public Attributes

- final int[] **value**

  *An array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity** (p. 912).*

8.15.1 Detailed Description

The key type of the built-in **topic** (p. 350) types.

Each remote **com.rti.dds.infrastructure.Entity** (p. 912) to be discovered is can be uniquely identified by this key. This is the key of all the built-in **topic** (p. 350) data types.

See also:

builtin.ParticipantBuiltinTopicData
builtin.TopicBuiltinTopicData (p. 1552)
builtin.PublicationBuiltinTopicData
builtin.SubscriptionBuiltinTopicData

8.15.2 Member Function Documentation

8.15.2.1 void copy_from (BuiltinTopicKey_t *other*)

Exceptions:

NullPointerException if 'other' is null.

8.15.2.2 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. 271)* 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. 466).

8.15.3 Member Data Documentation

8.15.3.1 final int [] value

An array of four integers that uniquely represents a remote **com.rti.dds.infrastructure.Entity** (p. 912).

8.16 BuiltinTopicReaderResourceLimits_t Class Reference

Built-in **topic** (p. 350) 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)

  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.

8.16.1 Detailed Description

Built-in **topic** (p. 350) reader's resource limits.

Defines the resources that can be used for a built-in-topic data reader.

A built-in **topic** (p. 350) data reader subscribes reliably to built-in topics containing declarations of new entities or updates to existing entities in the **domain** (p. 317). Keys are used to differentiate among entities of the same type. RTI Connext assigns a unique key to each entity in a **domain** (p. 317).

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

QoS:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615)

8.16.2 Constructor & Destructor Documentation

8.16.2.1 BuiltinTopicReaderResourceLimits_t ()

Constructor with default initial and maximum values.

8.16.2.2 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*)

Constructor with the given initial and maximum values.

8.16.3 Member Data Documentation

8.16.3.1 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

8.16.3.2 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] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_samples

8.16.3.3 int initial_infos

Initial number of sample infos.

The initial number of info units that a built-in **topic** (p. 350) **com.rti.dds.subscription.DataReader** (p. 473) can have. Info units are used to store **com.rti.dds.subscription.SampleInfo** (p. 1404).

[default] 64

[range] [1, 1 million] <= max_infos

8.16.3.4 int max_infos

Maximum number of sample infos.

The maximum number of info units that a built-in **topic** (p. 350) **com.rti.dds.subscription.DataReader** (p. 473) can use to store **com.rti.dds.subscription.SampleInfo** (p. 1404).

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_infos

8.17 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 size)

  Constructor that specifies the size of the allocated bytes array.

- 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.Bytes *(*p. 417*) array value.*

8.17.1 Detailed Description

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

8.17.2 Constructor & Destructor Documentation

8.17.2.1 Bytes ()

Default Constructor.

The default constructor initializes the newly created object with null value, zero length, and zero offset.

8.17.2.2 Bytes (Bytes *src*)

Copy constructor.

Parameters:

src *<<in>>* *(p. 271)* Object to copy from.

Exceptions:

NullPointerException if src is null.

8.17.2.3 Bytes (int *size*)

Constructor that specifies the size of the allocated bytes array.

After this method is called, length and offset are set to zero.

Parameters:

size *<<in>>* *(p. 271)* Size of the allocated bytes array.

Exceptions:

IllegalArgumentException if size is negative

8.17.3 Member Function Documentation

8.17.3.1 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. 271)* Object to copy from.

Returns:

this if success. Otherwise, null.

Exceptions:

NullPointerException if `src` is null.

Implements **Copyable** (p. 466).

8.17.4 Member Data Documentation

8.17.4.1 int length

Number of bytes to serialize.

8.17.4.2 int offset

Offset from which to start serializing bytes .

The first position of the bytes array has offset 0.

8.17.4.3 byte [] value

com.rti.dds.type.builtin.Bytes (p. 417) array value.

8.18 BytesDataReader Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataReader` < **com.rti.dds.type.builtin.Bytes** (p. 417) >.

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. 473).*

- 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. 473).*

- void **read_w_condition** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via **com.rti.dds.type.builtin.BytesDataReader.read** (p. 421) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **take_w_condition** (**BytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Analogous to **com.rti.dds.type.builtin.BytesDataReader.read_w_condition** (p. 422) except it accesses samples via the **com.rti.dds.type.builtin.BytesDataReader.take** (p. 421) 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. 473).*

- 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. 473).*

- void **return_loan** (**BytesSeq** received_data, **SampleInfoSeq** info_seq)

Indicates to the ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 473).*

8.18.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataReader` < **com.rti.dds.type.builtin.Bytes** (p. 417) >.

See also:

com.rti.dds.topic.example.FooDataReader
com.rti.dds.subscription.DataReader (p. 473)

8.18.2 Member Function Documentation

8.18.2.1 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read

8.18.2.2 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take

8.18.2.3 void read_w_condition (BytesSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via **com.rti.dds.type.builtin.BytesDataReader.read** (p. 421) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition

8.18.2.4 void take_w_condition (BytesSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to **com.rti.dds.type.builtin.BytesDataReader.read_w_condition** (p. 422) except it accesses samples via the **com.rti.dds.type.builtin.BytesDataReader.take** (p. 421) operation.

See also:

com.rti.dds.topic.example.FooDataReader.take_w_condition

8.18.2.5 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_sample

8.18.2.6 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_sample

8.18.2.7 void return_loan (BytesSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.return_loan

8.19 BytesDataWriter Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.Bytes** (p. 417) >.

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.Bytes*** *(p. 417) data instance.*

- void **write** (byte[] octets, int offset, int length, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Modifies the value of a* ***com.rti.dds.type.builtin.Bytes*** *(p. 417) data instance.*

- void **write** (**ByteSeq** octets, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Modifies the value of a* ***com.rti.dds.type.builtin.Bytes*** *(p. 417) 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.BytesDataWriter.write*** *(p. 425) 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. 270) *Performs the same function as* ***BytesDataWriter.write*** *(p. 425) 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. 270) *Performs the same function as* ***BytesDataWriter.write*** *(p. 425) except that it also provides the value for the* `source_timestamp`.

8.19.1 Detailed Description

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.Bytes** (p. 417) >.

See also:

com.rti.dds.topic.example.FooDataWriter
com.rti.dds.publication.DataWriter (p. 538)

8.19.2 Member Function Documentation

8.19.2.1 void write (Bytes *instance_data*, InstanceHandle_t *handle*)

Modifies the value of a **com.rti.dds.type.builtin.Bytes** (p. 417) data instance.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.19.2.2 void write (byte[] *octets*, int *offset*, int *length*, InstanceHandle_t *handle*)

<<*eXtension*>> *(p. 270)* Modifies the value of a **com.rti.dds.type.builtin.Bytes** (p. 417) data instance.

Parameters:

octets <<*in*>> *(p. 271)* Array of bytes to be published.

offset <<*in*>> *(p. 271)* Offset from which to start publishing.

length <<*in*>> *(p. 271)* Number of bytes to be published.

handle <<*in*>> *(p. 271)* The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) should be used always.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.19.2.3 void write (ByteSeq *octets*, InstanceHandle_t *handle*)

<<*eXtension*>> *(p. 270)* Modifies the value of a **com.rti.dds.type.builtin.Bytes** (p. 417) data instance.

Parameters:

octets *<<in>>* *(p. 271)* Sequence of bytes to be published.

handle *<<in>>* *(p. 271)* The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) should be used always.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.19.2.4 void write_w_timestamp (Bytes *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as **com.rti.dds.type.builtin.BytesDataWriter.write** (p. 425) except that it also provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.19.2.5 void write_w_timestamp (byte[] *octets*, int *offset*, int *length*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

<<eXtension>> *(p. 270)* Performs the same function as **Bytes-DataWriter.write** (p. 425) except that it also provides the value for the `source_timestamp`.

Parameters:

octets *<<in>>* *(p. 271)* Array of bytes to be published.

offset *<<in>>* *(p. 271)* Offset from which to start publishing.

length *<<in>>* *(p. 271)* Number of bytes to be published.

handle *<<in>>* *(p. 271)* The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) should be used always.

source_timestamp *<<in>>* *(p. 271)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See com.rti.dds.topic.example.FooDataWriter.write_w_timestamp. Cannot be NULL.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.19.2.6 void write_w_timestamp (ByteSeq *octets*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

<<eXtension>> *(p. 270)* Performs the same function as **BytesDataWriter.write** (p. 425) except that it also provides the value for the `source_timestamp`.

Parameters:

octets *<<in>>* *(p. 271)* Sequence of bytes to be published.

handle *<<in>>* *(p. 271)* The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) should be used always.

source_timestamp *<<in>>* *(p. 271)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See com.rti.dds.topic.example.FooDataWriter.write_w_timestamp. Cannot be NULL.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.20 ByteSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 430) that returns a java.lang.Byte.*

- Object **set** (int index, Object element)

  *A wrapper for **setByte()** (p. 431).*

- void **add** (int index, Object element)

  A wrapper for addByte(int, int).

8.20.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < byte >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

byte
com.rti.dds.util.Sequence (p. 1432)

8.20.2 Constructor & Destructor Documentation

8.20.2.1 ByteSeq ()

Constructs an empty sequence of bytes with an initial maximum of zero.

8.20.2.2 ByteSeq (int *initialMaximum*)

Constructs an empty sequence of bytes with the given initial maximum.

8.20.2.3 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

8.20.3 Member Function Documentation

8.20.3.1 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.

8.20.3.2 boolean addAllByte (byte[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.20.3.3 void addByte (byte *element*)

Append the element to the end of the sequence.

8.20.3.4 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.

8.20.3.5 byte getByte (int *index*)

Returns the byte at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.20.3.6 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.

8.20.3.7 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.

8.20.3.8 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.

8.20.3.9 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. 433), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.20.3.10 Object get (int *index*) [virtual]

A wrapper for **getByte(int)** (p. 430) that returns a java.lang.Byte.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.20.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setByte()** (p. 431).

Exceptions:

ClassCastException if the element is not of type Byte.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.20.3.12 void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.21 BytesSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.Bytes** (p. 417) > .

Inheritance diagram for BytesSeq::

Public Member Functions

- **BytesSeq** ()

  Constructs an empty sequence of ***com.rti.dds.type.builtin.Bytes*** *(p. 417) objects with an initial maximum of zero.*

- **BytesSeq** (int initialMaximum)

  Constructs an empty sequence of ***com.rti.dds.type.builtin.Bytes*** *(p. 417) objects with the given initial maximum.*

- **BytesSeq** (Collection elements)

  Constructs a new sequence containing the given ***com.rti.dds.type.builtin.Bytes*** *(p. 417) objects.*

- Object **copy_from** (Object src)

Package Attributes

- transient **Sequence** **_loanedInfoSequence** = null

8.21.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.Bytes** (p. 417) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.type.builtin.Bytes (p. 417)

8.21.2 Constructor & Destructor Documentation

8.21.2.1 BytesSeq ()

Constructs an empty sequence of **com.rti.dds.type.builtin.Bytes** (p. 417) objects with an initial maximum of zero.

8.21.2.2 BytesSeq (int *initialMaximum*)

Constructs an empty sequence of **com.rti.dds.type.builtin.Bytes** (p. 417) objects with the given initial maximum.

8.21.2.3 BytesSeq (Collection *elements*)

Constructs a new sequence containing the given **com.rti.dds.type.builtin.Bytes** (p. 417) objects.

Parameters:

elements the initial contents of this sequence.

Exceptions:

NullPointerException if the input collection is null

8.21.3 Member Function Documentation

8.21.3.1 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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.21.4 Member Data Documentation

8.21.4.1 transient Sequence _loanedInfoSequence = null [package]

When a memory loan has been taken out in the lower layers of NDDS, store a pointer to the native sequence here. That way, when we call finish(), we can give the memory back.

8.22 BytesTypeSupport Class Reference

<<*interface*>> *(p. 271)* **com.rti.dds.type.builtin.Bytes** (p. 417) type support.

Inheritance diagram for BytesTypeSupport::

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.Bytes*** *(p. 417) data type.*

- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the ***com.rti.dds.type.builtin.Bytes*** *(p. 417) 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.Bytes*** *(p. 417) type.*

8.22.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.type.builtin.Bytes** (p. 417) type support.

8.22.2 Member Function Documentation

8.22.2.1 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.Bytes** (p. 417) data type.

By default, The **com.rti.dds.type.builtin.Bytes** (p. 417) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **com.rti.dds.type.builtin.BytesTypeSupport.get_type_name** (p. 439). 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.BytesTypeSupport** (p. 437) with a **com.rti.dds.domain.DomainParticipant** (p. 629) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 629) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to register the data type **com.rti.dds.type.builtin.Bytes** (p. 417) with. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type **com.rti.dds.type.builtin.Bytes** (p. 417) is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1545). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_OUT_OF_RESOURCES.

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. 670)

8.22.2.2 static void unregister_type (DomainParticipant *participant*, String *type_name*) [static]

Allows an application to unregister the **com.rti.dds.type.builtin.Bytes** (p. 417) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **com.rti.dds.type.builtin.Bytes** (p. 417) type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1545) 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. 1545) is associated with the type, the operation will fail with RETCODE_ERROR.

Postcondition:

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to unregister the data type **com.rti.dds.type.builtin.Bytes** (p. 417) from. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type **com.rti.dds.type.builtin.Bytes** (p. 417) 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. 104), RETCODE_BAD_PARAMETER or RETCODE_ERROR

MT Safety:

SAFE.

See also:

com.rti.dds.type.builtin.BytesTypeSupport.register_type (p. 437)

8.22.2.3 static String get_type_name () [static]

Get the default name for the **com.rti.dds.type.builtin.Bytes** (p. 417) type.

Can be used for calling **com.rti.dds.type.builtin.BytesTypeSupport.register_type** (p. 437) or creating **com.rti.dds.topic.Topic** (p. 1545).

Returns:

default name for the **com.rti.dds.type.builtin.Bytes** (p. 417) type.

See also:

com.rti.dds.type.builtin.BytesTypeSupport.register_type (p. 437)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.23 ChannelSettings_t Class Reference

Type used to configure the properties of a channel.

Inherits Struct.

Public Member Functions

- **ChannelSettings_t** ()

  Constructor.

- **ChannelSettings_t** (**ChannelSettings_t** src)

  Constructor.

- **ChannelSettings_t** (**TransportMulticastSettingsSeq multicast_settings**, String **filter_expression**)

  Constructor.

Public Attributes

- **TransportMulticastSettingsSeq multicast_settings**

  *A sequence of **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) 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.

8.23.1 Detailed Description

Type used to configure the properties of a channel.

QoS:

com.rti.dds.infrastructure.MultiChannelQosPolicy (p. 1205)

8.23.2 Constructor & Destructor Documentation

8.23.2.1 ChannelSettings_t ()

Constructor.

8.23.2.2 ChannelSettings_t (ChannelSettings_t *src*)

Constructor.

Parameters:

src *<<in>>* *(p. 271)* Settings used to initialized the new settings.

8.23.2.3 ChannelSettings_t (TransportMulticastSettingsSeq *multicast_settings*, String *filter_expression*)

Constructor.

Parameters:

multicast_settings *<<in>>* *(p. 271)* Multicast settings.

filter_expression *<<in>>* *(p. 271)* Filter expression.

8.23.3 Member Data Documentation

8.23.3.1 TransportMulticastSettingsSeq multicast_settings

Initial value:

```
new TransportMulticastSettingsSeq()
```

A sequence of **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) 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 four (A locator is defined by a transport alias, a multicast address and a port)

[default] Empty sequence (invalid value)

8.23.3.2 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. 1207)

The filter expression length (including NULL-terminated character) cannot be greater than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.channel_filter_expression_max_length** (p. 756).

See also:

Queries and Filters Syntax (p. 278)

[**default**] NULL (invalid value)

8.24 ChannelSettingsSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.ChannelSettings_t** (p. 441) `>`.

Inherits ArraySequence.

8.24.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.ChannelSettings_t** (p. 441) `>`.

A sequence of **com.rti.dds.infrastructure.ChannelSettings_t** (p. 441) used to configure the channels' properties. If the length of the sequence is zero, the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) has no effect.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.ChannelSettings_t (p. 441)

8.25 CharSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 447) that returns a java.lang.Character.*

- final Object **set** (int index, Object element)

  *A wrapper for **setChar()** (p. 448).*

- final void **add** (int index, Object element)

  A wrapper for addChar(int, int).

8.25.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < char >.

Instantiates:

<<***generic***>> (p. 271) **com.rti.dds.util.Sequence** (p. 1432)

See also:

char
com.rti.dds.util.Sequence (p. 1432)

8.25.2 Constructor & Destructor Documentation

8.25.2.1 CharSeq ()

Constructs an empty sequence of single-byte (serialized) characters with an initial maximum of zero.

8.25.2.2 CharSeq (int *initialMaximum*)

Constructs an empty sequence of single-byte (serialized) characters with the given initial maximum.

8.25.2.3 CharSeq (char[] *chars*)

Constructs a new sequence containing the given single-byte (serialized) characters.

Parameters:

chars the initial contents of this sequence

8.25.3 Member Function Documentation

8.25.3.1 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.

8.25.3.2 final boolean addAllChar (char[] *elements*)

Append the elements of the given array into this sequence.

Exceptions:

NullPointerException if the given array is null

8.25.3.3 final void addChar (char *element*)

Append the element to the end of the sequence.

8.25.3.4 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.

8.25.3.5 final char getChar (int *index*)

Returns the character at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.25.3.6 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.

8.25.3.7 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.

8.25.3.8 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.

8.25.3.9 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. 450), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.25.3.10 final Object get (int *index*) [virtual]

A wrapper for **getChar(int)** (p. 447) that returns a java.lang.Character.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.25.3.11 final Object set (int *index*, Object *element*) [virtual]

A wrapper for **setChar()** (p. 448).

Exceptions:

ClassCastException if the element is not of type Character.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.25.3.12 final void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.26 Condition Interface Reference

<<interface>> *(p. 271)* Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1695).

Inheritance diagram for Condition::

Package Functions

- boolean **get_trigger_value** ()

  Retrieve the `trigger_value`.

8.26.1 Detailed Description

<<interface>> *(p. 271)* Root class for all the conditions that may be attached to a **com.rti.dds.infrastructure.WaitSet** (p. 1695).

This basic class is specialised in three classes:

com.rti.dds.infrastructure.GuardCondition (p. 1066), **com.rti.dds.infrastructure.StatusCondition** (p. 1452), and **com.rti.dds.subscription.ReadCondition** (p. 1326).

A **com.rti.dds.infrastructure.Condition** (p. 451) has a `trigger_value` that can be true or false and is set automatically by RTI Connext.

See also:

com.rti.dds.infrastructure.WaitSet (p. 1695)

8.26.2 Member Function Documentation

8.26.2.1 boolean get_trigger_value () [package]

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implemented in **GuardCondition** (p. 1067).

8.27 ConditionSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.infrastructure.Condition** (p. 451) >.

Inherits AbstractNativeSequence.

Public Member Functions

- **ConditionSeq** (Collection conditions)

  *Create a **Condition** (p. 451) 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. 451) Sequence with a initial maximum of zero.*

- **ConditionSeq** (int initial_maximum)

  *Create a **Condition** (p. 451) Sequence with the specified initial maximum.*

- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.27.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.infrastructure.Condition** (p. 451) >.

Instantiates:

<<***generic***>> (p. 271) **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.WaitSet (p. 1695)
com.rti.dds.util.Sequence (p. 1432)

8.27.2 Constructor & Destructor Documentation

8.27.2.1 ConditionSeq (Collection *conditions*)

Create a **Condition** (p. 451) 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

8.27.2.2 ConditionSeq ()

Create a empty **Condition** (p. 451) Sequence with a initial maximum of zero.

8.27.2.3 ConditionSeq (int *initial_maximum*)

Create a **Condition** (p. 451) Sequence with the specified initial maximum.

Parameters:

initial_maximum The initial maximum of the sequence.

8.27.3 Member Function Documentation

8.27.3.1 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. 383), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.28 ContentFilter Interface Reference

<<*interface*>> *(p. 271)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458)

Inherited by CustomContentFilterHowTo.MyContentFilter.

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)

  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.

8.28.1 Detailed Description

<<*interface*>> *(p. 271)* Interface to be used by a custom filter of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458)

Entity:

com.rti.dds.topic.ContentFilteredTopic (p. 458)

This is the interface that can be implemented by an application-provided class and then registered with the **com.rti.dds.domain.DomainParticipant** (p. 629) such that samples can be filtered for a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) 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. 458)

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 698)

8.28.2 Member Function Documentation

8.28.2.1 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.ContentFilteredTopic** (p. 458) with the matching filter name is created, or when a **com.rti.dds.subscription.DataReader** (p. 473) 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. 271)* 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. 456) and **com.rti.dds.topic.ContentFilter.finalize** (p. 457) functions for this instance of the content filter. Can be set to NULL.

expression *<<**in**>>* *(p. 271)* 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. 271)* A string sequence with the expression parameters the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) 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. 673). 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. 271)* A pointer to the type code for the related **com.rti.dds.topic.Topic** (p. 1545) of the

com.rti.dds.topic.ContentFilteredTopic (p. 458). 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. 271)* Fully qualified class name of the related **com.rti.dds.topic.Topic** (p. 1545).

old_compile_data *<<**in**>>* *(p. 271)* 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. 458), 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 resources previously allocated for this

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.28.2.2 boolean evaluate (Object *compile_data*, Object *sample*)

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. 473) associated with the filter is received, or when a sample for a discovered **com.rti.dds.subscription.DataReader** (p. 473) 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. 271)* The last return value of the **com.rti.dds.topic.ContentFilter.compile** (p. 455) function for this instance of the content filter. Can be NULL.

sample *<<**in**>>* *(p. 271)* Pointer to a deserialized sample to be filtered

Returns:

The function must return 0 if the sample should be filtered out, non zero otherwise

8.28.2.3 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.ContentFilteredTopic** (p. 458) with the matching filter name is deleted, or when a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) with a matching filter name if the filter is unregistered with **com.rti.dds.domain.DomainParticipant.unregister_contentfilter** (p. 700)

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. 271)* The last return value of the **com.rti.dds.topic.ContentFilter.compile** (p. 455) function for this instance of the content filter. Can be NULL.

8.29 ContentFilteredTopic Interface Reference

<<*interface*>> *(p. 271)* Specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) 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`.
- **Topic get_related_topic** ()
  Get the `related_topic`.
- void **append_to_expression_parameter** (int index, String val)
  <<**eXtension**>> (p. 270) *Appends a string term to the specified parameter string.*
- void **remove_from_expression_parameter** (int index, String val)
  <<**eXtension**>> (p. 270) *Removes a string term from the specified parameter string.*

8.29.1 Detailed Description

<<*interface*>> *(p. 271)* Specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) that allows for content-based subscriptions.

It describes a more sophisticated **subscription** (p. 343) that indicates a **com.rti.dds.subscription.DataReader** (p. 473) does not want to necessarily see all values of each instance published under the **com.rti.dds.topic.Topic** (p. 1545). 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. 278) describes the syntax of `filter_expression` and `expression_parameters`.

Note on Content-Based Filtering and Sparse Value Types

If you are a user of the **Dynamic Data** (p. 170) API, you may define *sparse value types*; that is, types for which every data sample need not include a value for every field defined in the type. (See TCKind.TK_SPARSE and TypeCodeFactory.create_sparse_tc.) In order for a filter expression on a field to be well defined, that field must be present in the data sample. That means that you will only be able to perform a content-based filter on fields that are marked as **TypeCode.KEY_MEMBER** (p. 1639) or **TypeCode.NONKEY_REQUIRED_MEMBER** (p. 1639).

8.29.2 Member Function Documentation

8.29.2.1 String get_filter_expression ()

Get the `filter_expression`.

Return the `filter_expression` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

Returns:

the `filter_expression`.

8.29.2.2 void get_expression_parameters (StringSeq *parameters*)

Get the `expression_parameters`.

Return the `expression_parameters` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 458). `expression_parameters` is either specified on the last successful call to **com.rti.dds.topic.ContentFilteredTopic.set_expression_parameters** (p. 460) or, if that method is never called, the parameters specified when the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) was created.

Parameters:

parameters *<<inout>> (p. 271)* the filter expression parameters. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.domain.DomainParticipant.create_-contentfilteredtopic (p. 673)
com.rti.dds.topic.ContentFilteredTopic.set_expression_-parameters (p. 460)

8.29.2.3 void set_expression_parameters (StringSeq *parameters*)

Set the `expression_parameters`.

Change the `expression_parameters` associated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

Parameters:

parameters *<<in>> (p. 271)* the filter expression parameters Cannot be NULL.. Length of sequence cannot be greater than 100.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.29.2.4 Topic get_related_topic ()

Get the `related_topic`.

Return the **com.rti.dds.topic.Topic** (p. 1545) specified when the **com.rti.dds.topic.ContentFilteredTopic** (p. 458) was created.

Returns:

The **com.rti.dds.topic.Topic** (p. 1545) assocated with the **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

8.29.2.5 void append_to_expression_parameter (int *index*, String *val*)

<<*eXtension*>> *(p. 270)* 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** (p. 353), 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. 271)* The index of the parameter string to be modified. The first index is index 0. When using the **DomainParticipant.STRINGMATCHFILTER_NAME** (p. 151) filter, `index` *must* be 0.

val <<*in*>> *(p. 271)* The string term to be appended to the parameter string.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.29.2.6 void remove_from_expression_parameter (int *index*, String *val*)

<<*eXtension*>> *(p. 270)* 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**

(p. 353), 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. 271)* The index of the parameter string to be modified. The first index is index 0. When using the **DomainParticipant.STRINGMATCHFILTER_NAME** (p. 151) filter, `index` *must* be 0.

val <<***in***>> *(p. 271)* The string term to be removed from the parameter string.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.30 ContentFilterProperty_t Class Reference

<<*eXtension*>> *(p. 270)* 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.

8.30.1 Detailed Description

<<*eXtension*>> *(p. 270)* Type used to provide all the required information to enable content filtering.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 ContentFilterProperty_t ()

Constructor.

8.30.3 Member Data Documentation

8.30.3.1 String content_filter_topic_name

Name of the Content-filtered Topic associated with the Reader.

8.30.3.2 String related_topic_name

Name of the Topic related to the Content-filtered Topic.

8.30.3.3 String filter_class_name

Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).

8.30.3.4 String filter_expression

The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

8.30.3.5 final StringSeq expression_parameters

Defines the value for each parameter in the filter expression.

8.31 Cookie_t Class Reference

<<*eXtension*>> *(*p. *270)* Sequence of bytes identifying a written data sample, used when writing with parameters.

Inherits Struct.

Public Attributes

- ^ final **ByteSeq value**

  a sequence of octets

8.31.1 Detailed Description

<<*eXtension*>> *(*p. *270)* Sequence of bytes identifying a written data sample, used when writing with parameters.

8.31.2 Member Data Documentation

8.31.2.1 final ByteSeq value

a sequence of octets

[default] Empty (zero-sized)

[range] Octet sequence of length [0,**com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.cookie_max_length** (p. 601)]

8.32 Copyable Interface Reference

*<<**eXtension**>> (p. 270) <<**interface**>> (p. 271)* 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.

8.32.1 Detailed Description

*<<**eXtension**>> (p. 270) <<**interface**>> (p. 271)* Interface for all the user-defined data type classes that support copy.

A class implements the **com.rti.dds.infrastructure.Copyable** (p. 466) 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.)

8.32.2 Member Function Documentation

8.32.2.1 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. 271)* 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 **DynamicData** (p. 796), **InstanceHandle_t** (p. 1081), **StringSeq** (p. 1471), **SampleInfo** (p. 1409), **BuiltinTopicKey_t** (p. 413), **Bytes** (p. 418), **BytesSeq** (p. 435), **KeyedBytes** (p. 1096), **KeyedBytesSeq** (p. 1117), **KeyedString** (p. 1124), **KeyedStringSeq** (p. 1142), **AbstractPrimitiveSequence** (p. 381), **Enum** (p. 926), **Foo** (p. 956), and **FooSeq** (p. 1059).

8.33 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 database thread will wake up at this rate to clean up the database.
- 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.

8.33.1 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. 629) 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. 470) and **com.rti.dds.infrastructure.DatabaseQosPolicy.shutdown_cleanup_period** (p. 470) parameters to decrease the time it takes to delete a **com.rti.dds.domain.DomainParticipant** (p. 629) when your application is shutting down.

The **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 741) 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. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) **NO** (p. 98)

8.33.2 Member Data Documentation

8.33.2.1 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 Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 10

For INTEGRITY: 60

For VxWorks: 120

For all others: OS default priority.

[default] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 16∗1024

For INTEGRITY: 20∗1024

For VxWorks: 16∗1024

For all others: OS default stack size.

[default] mask **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1536)

8.33.2.2 final Duration_t shutdown_timeout

The maximum wait time during a shutdown.

The **domain** (p. 317) 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.INFINITE]

8.33.2.3 final Duration_t cleanup_period

The database thread will wake up at this rate to clean up the database.

[default] 61 seconds

[range] [0,1 year]

8.33.2.4 final Duration_t shutdown_cleanup_period

The clean-up period used during database shut-down.

[default] 1 second

[range] [0,1 year]

8.33.2.5 int initial_records

The initial number of total records.

[default] 1024

[range] [1,10 million]

8.33.2.6 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] 14

[range] [1,31]

8.33.2.7 int initial_weak_references

The initial number of weak references.

See **com.rti.dds.infrastructure.DatabaseQosPolicy.max_weak_references** (p. 471) 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. 471) 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. 471)

8.33.2.8 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. 471)) 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 **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102). 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] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_weak_references

See also:

com.rti.dds.infrastructure.DatabaseQosPolicy.initial_weak_references (p. 471)

8.34 DataReader Interface Reference

<<*interface*>> *(p. 271)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a **subscription** (p. 343)) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1478).

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. 1326).*

- **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. 1324).*

- void **delete_readcondition** (**ReadCondition** condition)

  *Deletes a **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324) attached to the **com.rti.dds.subscription.DataReader** (p. 473).*

- void **set_qos** (**DataReaderQos** qos)

  Sets the reader QoS.

- void **set_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Change 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** ()

  Get the reader listener.

- void **call_listenerT** (int mask)

Calls the reader listener.

- void **get_sample_rejected_status** (**SampleRejectedStatus** status)

  Accesses the StatusKind.SAMPLE_REJECTED_STATUS communication status.

- void **get_liveliness_changed_status** (**LivelinessChangedStatus** status)

  Accesses the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

- void **get_requested_deadline_missed_status** (**RequestedDeadlineMissedStatus** status)

  Accesses the StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

- void **get_requested_incompatible_qos_status** (**RequestedIncompatibleQosStatus** status)

  Accesses the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

- void **get_sample_lost_status** (**SampleLostStatus** status)

  Accesses the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

- void **get_subscription_matched_status** (**SubscriptionMatchedStatus** status)

  Accesses the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

- void **get_datareader_cache_status** (**DataReaderCacheStatus** status)

  <<**eXtension**>> (p. 270) *Get the datareader cache status for this reader.*

- void **get_datareader_protocol_status** (**DataReaderProtocolStatus** status)

  <<**eXtension**>> (p. 270) *Get the datareader protocol status for this reader.*

- void **get_matched_publication_datareader_protocol_status** (**DataReaderProtocolStatus** status, **InstanceHandle_t** publication_handle)

  <<**eXtension**>> (p. 270) *Get the datareader protocol status for this reader, per matched **publication** (p. 338) identified by the publication_handle.*

^ void **get_matched_publications** (**InstanceHandleSeq** publication_handles)

*Retrieve the list of publications currently "associated" with this **com.rti.dds.subscription.DataReader** (p. 473).*

^ void **get_matched_publication_data** (**PublicationBuiltinTopicData** publication_data, **InstanceHandle_t** publication_handle)

*This operation retrieves the information on a **publication** (p. 338) that is currently "associated" with the **com.rti.dds.subscription.DataReader** (p. 473).*

^ **TopicDescription get_topicdescription** ()

*Returns the **com.rti.dds.topic.TopicDescription** (p. 1561) associated with the **com.rti.dds.subscription.DataReader** (p. 473).*

^ **Subscriber get_subscriber** ()

*Returns the **com.rti.dds.subscription.Subscriber** (p. 1478) to which the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).*

^ void **wait_for_historical_data** (**Duration_t** max_wait)

*Waits until all "historical" data is received for **com.rti.dds.subscription.DataReader** (p. 473) entities that have a non-VOLATILE Durability Qos kind.*

^ void **read_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, int sample_states, int view_states, int instance_states)

Read 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)

Take data samples, if any are available.

^ void **read_w_condition_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** read_condition)

Read data samples, if any are available.

^ void **take_w_condition_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** read_condition)

Take data samples, if any are available.

ˆ void **read_next_sample_untyped** (Object received_data, **SampleInfo** sample_info)

Read data samples, if any are available.

ˆ void **take_next_sample_untyped** (Object received_data, **SampleInfo** sample_info)

Take data samples, if any are available.

ˆ 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)

Read data samples, if any are available.

ˆ 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)

Take data samples, if any are available.

ˆ void **read_instance_w_condition_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

Read data samples, if any are available.

ˆ void **take_instance_w_condition_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

Take data samples, if any are available.

ˆ 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)

Read data samples, if any are available.

ˆ 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)

Take data samples, if any are available.

ˆ void **read_next_instance_w_condition_untyped** (List received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** a_handle, **ReadCondition** read_condition)

Read 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)

  Take data samples, if any are available.

- void **return_loan_untyped** (List received_data, **SampleInfoSeq** info_seq)

  Return loaned sample data and meta-data.

- void **get_key_value_untyped** (Object key_holder, **InstanceHandle_t** handle)

  Fill in the key fields of the given data sample.

8.34.1 Detailed Description

<<***interface***>> *(p. 271)* Allows the application to: (1) declare the data it wishes to receive (i.e. make a **subscription** (p. 343)) and (2) access the data received by the attached **com.rti.dds.subscription.Subscriber** (p. 1478).

QoS:

com.rti.dds.subscription.DataReaderQos (p. 518)

Status:

StatusKind.DATA_AVAILABLE_STATUS;
StatusKind.LIVELINESS_CHANGED_STATUS,
com.rti.dds.subscription.LivelinessChangedStatus (p. 1159);
StatusKind.REQUESTED_DEADLINE_MISSED_STATUS,
com.rti.dds.subscription.RequestedDeadlineMissedStatus (p. 1353);
StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS,
com.rti.dds.subscription.RequestedIncompatibleQosStatus (p. 1354);
StatusKind.SAMPLE_LOST_STATUS_STATUS,
com.rti.dds.subscription.SampleLostStatus (p. 1415);
StatusKind.SAMPLE_REJECTED_STATUS,
com.rti.dds.subscription.SampleRejectedStatus (p. 1422);
StatusKind.SUBSCRIPTION_MATCHED_STATUS,
com.rti.dds.subscription.SubscriptionMatchedStatus

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

A **com.rti.dds.subscription.DataReader** (p. 473) refers to exactly one **com.rti.dds.topic.TopicDescription** (p. 1561) (either a **com.rti.dds.topic.Topic** (p. 1545), a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) or a **com.rti.dds.topic.MultiTopic** (p. 1208)) that identifies the data to be read.

The **subscription** (p. 343) 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. 473) is an abstract class. It must be specialised for each particular application data-type (see **USER_DATA** (p. 126)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type `Foo` are specified in the generic type com.rti.dds.topic.example.FooDataReader.

The following operations may be called even if the **com.rti.dds.subscription.DataReader** (p. 473) is not enabled. Other operations will fail with the value RETCODE_NOT_ENABLED if called on a disabled **com.rti.dds.subscription.DataReader** (p. 473):

- The base-class operations **com.rti.dds.subscription.DataReader.set_qos** (p. 480), **com.rti.dds.subscription.DataReader.get_qos** (p. 482), **com.rti.dds.subscription.DataReader.set_listener** (p. 482), **com.rti.dds.subscription.DataReader.get_listener** (p. 483), **com.rti.dds.infrastructure.Entity.enable** (p. 915), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 917) and **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917)
- **com.rti.dds.subscription.DataReader.get_liveliness_changed_status** (p. 484) **com.rti.dds.subscription.DataReader.get_requested_deadline_missed_status** (p. 484) **com.rti.dds.subscription.DataReader.get_requested_incompatible_qos_status** (p. 484) **com.rti.dds.subscription.DataReader.get_sample_lost_status** (p. 485) **com.rti.dds.subscription.DataReader.get_sample_rejected_status** (p. 483) **com.rti.dds.subscription.DataReader.get_subscription_matched_status** (p. 485)

All sample-accessing operations, namely: com.rti.dds.topic.example.FooDataReader.read, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.read_w_condition, and com.rti.dds.topic.example.FooDataReader.take_w_condition may fail with the error RETCODE_PRECONDITION_NOT_MET as described in **com.rti.dds.subscription.Subscriber.begin_access** (p. 1499).

See also:

Operations Allowed in Listener Callbacks (p. 1156)

8.34.2 Member Function Documentation

8.34.2.1 ReadCondition create_readcondition (int *sample_states*, int *view_states*, int *instance_states*)

Creates a **com.rti.dds.subscription.ReadCondition** (p. 1326).

The returned **com.rti.dds.subscription.ReadCondition** (p. 1326) will be attached and belong to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

sample_states *<<in>> (p. 271)* sample state of the data samples that are of interest

view_states *<<in>> (p. 271)* view state of the data samples that are of interest

instance_states *<<in>> (p. 271)* instance state of the data samples that are of interest

Returns:

return **com.rti.dds.subscription.ReadCondition** (p. 1326) created. Returns NULL in case of failure.

8.34.2.2 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. 1324).

The returned **com.rti.dds.subscription.QueryCondition** (p. 1324) will be attached and belong to the **com.rti.dds.subscription.DataReader** (p. 473).

Queries and Filters Syntax (p. 278) describes the syntax of `query_expression` and `query_parameters`.

Parameters:

sample_states *<<in>> (p. 271)* sample state of the data samples that are of interest

view_states *<<in>> (p. 271)* view state of the data samples that are of interest

instance_states *<<in>> (p. 271)* instance state of the data samples that are of interest

query_expression *<<in>> (p. 271)* Expression for the query. Cannot be NULL.

query_parameters *<<in>>* *(p. 271)* Parameters for the query expression. Cannot be NULL.

Returns:

return **com.rti.dds.subscription.QueryCondition** (p. 1324) created. Returns NULL in case of failure.

8.34.2.3 void delete_readcondition (ReadCondition *condition*)

Deletes a **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324) attached to the **com.rti.dds.subscription.DataReader** (p. 473).

Since **com.rti.dds.subscription.QueryCondition** (p. 1324) specializes **com.rti.dds.subscription.ReadCondition** (p. 1326), it can also be used to delete a **com.rti.dds.subscription.QueryCondition** (p. 1324).

Precondition:

The **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473), or the operation will fail with the error RETCODE_PRECONDITION_NOT_MET.

Parameters:

condition *<<in>>* *(p. 271)* Condition to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET

8.34.2.4 void set_qos (DataReaderQos *qos*)

Sets the reader QoS.

This operation modifies the QoS of the **com.rti.dds.subscription.DataReader** (p. 473).

The **com.rti.dds.subscription.DataReaderQos.user_data** (p. 521), **com.rti.dds.subscription.DataReaderQos.deadline** (p. 521), **com.rti.dds.subscription.DataReaderQos.latency_budget** (p. 521), **com.rti.dds.subscription.DataReaderQos.time_based_filter** (p. 521), **com.rti.dds.subscription.DataReaderQos.reader_data_lifecycle** (p. 522) can be changed. The other policies are immutable.

Parameters:

qos *<<in>> (p. 271)* The **com.rti.dds.subscription.DataReaderQos** (p. 518) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.subscription.DataReader** (p. 473) is enabled. The special value **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) can be used to indicate that the QoS of the **com.rti.dds.subscription.DataReader** (p. 473) should be changed to match the current default **com.rti.dds.subscription.DataReaderQos** (p. 518) set in the **com.rti.dds.subscription.Subscriber** (p. 1478). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
set_qos (abstract) (p. 913)
com.rti.dds.subscription.DataReader.set_qos (p. 480)
Operations Allowed in Listener Callbacks (p. 1156)

8.34.2.5 void set_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> (p. 270) Change the QoS of this reader using the input XML QoS profile.

This operation modifies the QoS of the **com.rti.dds.subscription.DataReader** (p. 473).

The **com.rti.dds.subscription.DataReaderQos.user_data** (p. 521), **com.rti.dds.subscription.DataReaderQos.deadline** (p. 521), **com.rti.dds.subscription.DataReaderQos.latency_budget** (p. 521), **com.rti.dds.subscription.DataReaderQos.time_based_filter** (p. 521), **com.rti.dds.subscription.DataReaderQos.reader_data_lifecycle** (p. 522) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>> (p. 271)* 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. 1495)).

profile_name *<<in>>* *(p. 271)* 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. 1496)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
com.rti.dds.subscription.DataReader.set_qos (p. 480)
Operations Allowed in Listener Callbacks (p. 1156)

8.34.2.6 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. 271)* The **com.rti.dds.subscription.DataReaderQos** (p. 518) to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.34.2.7 void set_listener (DataReaderListener *l*, int *mask*)

Sets the reader listener.

Parameters:

l *<<in>>* *(p. 271)* **com.rti.dds.subscription.DataReaderListener** (p. 501) to set to

mask *<<in>> (p. 271)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.subscription.DataReaderListener** (p. 501).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.34.2.8 DataReaderListener get_listener ()

Get the reader listener.

Returns:

com.rti.dds.subscription.DataReaderListener (p. 501) of the **com.rti.dds.subscription.DataReader** (p. 473).

See also:

get_listener (abstract) (p. 915)

8.34.2.9 void call_listenerT (int *mask*)

Calls the reader listener.

See also:

get_listener (abstract) (p. 915)

8.34.2.10 void get_sample_rejected_status (SampleRejectedStatus *status*)

Accesses the StatusKind.SAMPLE_REJECTED_STATUS communication status.

Parameters:

status *<<inout>> (p. 271)* **com.rti.dds.subscription.SampleRejectedStatus** (p. 1422) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.11 void get_liveliness_changed_status (LivelinessChangedStatus *status*)

Accesses the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1159) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.12 void get_requested_deadline_missed_status (RequestedDeadlineMissedStatus *status*)

Accesses the StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.subscription.RequestedDeadlineMissedStatus** (p. 1353) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.13 void get_requested_incompatible_qos_status (RequestedIncompatibleQosStatus *status*)

Accesses the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.subscription.RequestedIncompatibleQosStatus** (p. 1354) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.14 void get_sample_lost_status (SampleLostStatus *status*)

Accesses the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.subscription.SampleLostStatus** (p. 1415) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.15 void get_subscription_matched_status (SubscriptionMatchedStatus *status*)

Accesses the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.subscription.SubscriptionMatchedStatus** (p. 1520) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.34.2.16 void get_datareader_cache_status (DataReaderCacheStatus *status*)

<<eXtension>> *(p. 270)* Get the datareader cache status for this reader.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.subscription.DataReaderCacheStatus** (p. 500) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.34.2.17 void get_datareader_protocol_status (DataReaderProtocolStatus *status*)

<<*eXtension*>> *(p. 270)* Get the datareader protocol status for this reader.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.subscription.DataReaderProtocolStatus** (p. 508) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.34.2.18 void get_matched_publication_datareader_protocol_status (DataReaderProtocolStatus *status*, InstanceHandle_t *publication_handle*)

<<*eXtension*>> *(p. 270)* Get the datareader protocol status for this reader, per matched **publication** (p. 338) identified by the publication_handle.

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. 271)*. The information to be filled in on the associated **publication** (p. 338). Cannot be NULL.

publication_handle <<*in*>> *(p. 271)*. Handle to a specific **publication** (p. 338) associated with the **com.rti.dds.publication.DataWriter** (p. 538). Cannot be NULL.. Must correspond to a **publication** (p. 338) currently associated with the **com.rti.dds.subscription.DataReader** (p. 473).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED

8.34.2.19 void get_matched_publications (InstanceHandleSeq *publication_handles*)

Retrieve the list of publications currently "associated" with this **com.rti.dds.subscription.DataReader** (p. 473).

Matching publications are those in the same **domain** (p. 317) that have a matching **com.rti.dds.topic.Topic** (p. 1545), compatible QoS common partition that the **com.rti.dds.domain.DomainParticipant** (p. 629) has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 688) operation.

The handles returned in the `publication_handles'` list are the ones that are used by the DDS implementation to locally identify the corresponding matched **com.rti.dds.publication.DataWriter** (p. 538) entities. These handles match the ones that appear in the `instance_handle` field of the **com.rti.dds.subscription.SampleInfo** (p. 1404) when reading the PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME **builtin** (p. 348) **topic** (p. 350)

Parameters:

publication_handles *<<inout>> (p. 271).* 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 RETCODE_OUT_OF_RESOURCES.

The maximum number of matches possible is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 741). 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. 104), or RETCODE_OUT_OF_RESOURCES if the sequence is too small and the system can not resize it, or RETCODE_NOT_ENABLED

8.34.2.20 void get_matched_publication_data (PublicationBuiltinTopicData *publication_data*, InstanceHandle_t *publication_handle*)

This operation retrieves the information on a **publication** (p. 338) that is currently "associated" with the **com.rti.dds.subscription.DataReader** (p. 473).

Publication with a matching **com.rti.dds.topic.Topic** (p. 1545), compatible QoS and common partition that the application has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 688) operation.

The publication_handle must correspond to a **publication** (p. 338) currently associated with the **com.rti.dds.subscription.DataReader** (p. 473). Otherwise, the operation will fail with RETCODE_BAD_PARAMETER. Use the operation **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 486) to find the publications that are currently matched with the **com.rti.dds.subscription.DataReader** (p. 473).

Note: This operation does not retrieve the following information in builtin.PublicationBuiltinTopicData:

- builtin.PublicationBuiltinTopicData.type_code
- builtin.PublicationBuiltinTopicData.property

The above information is available through **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 503) (if a reader listener is installed on the builtin.PublicationBuiltinTopicDataDataReader).

Parameters:

publication_data *<<inout>> (p. 271).* The information to be filled in on the associated **publication** (p. 338). Cannot be NULL.

publication_handle *<<in>> (p. 271).* Handle to a specific **publication** (p. 338) associated with the **com.rti.dds.publication.DataWriter** (p. 538). Cannot be NULL.. Must correspond to a **publication** (p. 338) currently associated with the **com.rti.dds.subscription.DataReader** (p. 473).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED

8.34.2.21 TopicDescription get_topicdescription ()

Returns the **com.rti.dds.topic.TopicDescription** (p. 1561) associated with the **com.rti.dds.subscription.DataReader** (p. 473).

Returns that same **com.rti.dds.topic.TopicDescription** (p. 1561) that was used to create the **com.rti.dds.subscription.DataReader** (p. 473).

Returns:

com.rti.dds.topic.TopicDescription (p. 1561) associated with the **com.rti.dds.subscription.DataReader** (p. 473).

8.34.2.22 Subscriber get_subscriber ()

Returns the **com.rti.dds.subscription.Subscriber** (p. 1478) to which the **com.rti.dds.subscription.DataReader** (p. 473) belongs.

Returns:

com.rti.dds.subscription.Subscriber (p. 1478) to which the **com.rti.dds.subscription.DataReader** (p. 473) belongs.

8.34.2.23 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. 473).

Deletes all contained **com.rti.dds.subscription.ReadCondition** (p. 1326) and **com.rti.dds.subscription.QueryCondition** (p. 1324) objects.

The operation will fail with RETCODE_PRECONDITION_NOT_MET 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. 489) completes successfully, the application may delete the **com.rti.dds.subscription.DataReader** (p. 473), knowing that it has no contained **com.rti.dds.subscription.ReadCondition** (p. 1326) and **com.rti.dds.subscription.QueryCondition** (p. 1324) objects.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET

8.34.2.24 void wait_for_historical_data (Duration_t *max_wait*)

Waits until all "historical" data is received for **com.rti.dds.subscription.DataReader** (p. 473) entities that have a non-VOLATILE Durability Qos kind.

This operation is intended only for **com.rti.dds.subscription.DataReader** (p. 473) entities that have a non-VOLATILE Durability QoS kind.

As soon as an application enables a non-VOLATILE **com.rti.dds.subscription.DataReader** (p. 473), 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. 473) joined the **domain** (p. 317)) as well as any new data written by the **com.rti.dds.publication.DataWriter**

(p. 538) 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. 489) operations.

The operation **com.rti.dds.subscription.DataReader.wait_for_historical_data** (p. 489) blocks the calling thread until either all "historical" data is received, or else duration specified by the max_wait parameter clapses, whichever happens first. 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. 271)* Timeout value. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED.

8.34.2.25 void read_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, int *sample_states*, int *view_states*, int *instance_states*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.take_untyped (p. 490)
com.rti.dds.topic.example.FooDataReader.read

8.34.2.26 void take_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, int *sample_states*, int *view_states*, int *instance_states*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.read_untyped (p. 490)
com.rti.dds.topic.example.FooDataReader.take

8.34.2.27 void read_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *read_condition*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_w_condition method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.take_w_condition_untyped (p. 491)
com.rti.dds.topic.example.FooDataReader.read_w_condition

8.34.2.28 void take_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *read_condition*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_w_condition method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.read_w_condition_untyped (p. 491)
com.rti.dds.topic.example.FooDataReader.take_w_condition

8.34.2.29 void read_next_sample_untyped (Object *received_data*, SampleInfo *sample_info*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_next_sample method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.take_next_sample_untyped (p. 492)
com.rti.dds.topic.example.FooDataReader.read_next_sample

8.34.2.30 void take_next_sample_untyped (Object *received_data*, SampleInfo *sample_info*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_next_sample method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.read_next_sample_untyped (p. 492)
com.rti.dds.topic.example.FooDataReader.take_next_sample

8.34.2.31 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*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.take_instance_untyped (p. 493)
com.rti.dds.topic.example.FooDataReader.read_instance

8.34.2.32 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*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.read_instance_untyped (p. 492)
com.rti.dds.topic.example.FooDataReader.take_instance

8.34.2.33 void read_instance_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *a_handle*, ReadCondition *read_condition*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition 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. 495)
com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition

8.34.2.34 void take_instance_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *a_handle*, ReadCondition *read_condition*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition 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. 495)
com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition

8.34.2.35 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*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_next_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.take_next_instance_untyped (p. 494)
com.rti.dds.topic.example.FooDataReader.read_next_instance

8.34.2.36 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*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_next_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.subscription.DataReader.read_next_instance_-untyped (p. 494)
com.rti.dds.topic.example.FooDataReader.take_next_instance

8.34.2.37 void read_next_instance_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *a_handle*, ReadCondition *read_condition*)

Read data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition 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. 495)
com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition

8.34.2.38 void take_next_instance_w_condition_untyped (List *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *a_handle*, ReadCondition *read_condition*)

Take data samples, if any are available.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition 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. 495)
com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition

8.34.2.39 void return_loan_untyped (List *received_data*, SampleInfoSeq *info_seq*)

Return loaned sample data and meta-data.

This method allows type-independent code to work with a variety of concrete com.rti.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.return_loan method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.topic.example.FooDataReader.return_loan

8.34.2.40 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.dds.topic.example.FooDataReader classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataReader.get_key_value method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.35 DataReaderAdapter Class Reference

<<***eXtension***>> *(p. 270)* 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 StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

- void **on_requested_incompatible_qos** (**DataReader** reader, **RequestedIncompatibleQosStatus** status)

  Handles the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

- void **on_sample_rejected** (**DataReader** reader, **SampleRejectedStatus** status)

  Handles the StatusKind.SAMPLE_REJECTED_STATUS communication status.

- void **on_liveliness_changed** (**DataReader** reader, **LivelinessChangedStatus** status)

  Handles the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

- void **on_data_available** (**DataReader** reader)

  Handle the StatusKind.DATA_AVAILABLE_STATUS communication status.

- void **on_sample_lost** (**DataReader** reader, **SampleLostStatus** status)

  Handles the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

- void **on_subscription_matched** (**DataReader** reader, **SubscriptionMatchedStatus** status)

  Handles the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

8.35.1 Detailed Description

<<*eXtension*>> *(p. 270)* 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.

8.35.2 Member Function Documentation

8.35.2.1 void on_requested_deadline_missed (DataReader *reader*, RequestedDeadlineMissedStatus *status*)

Handles the StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

Implements **DataReaderListener** (p. 502).

8.35.2.2 void on_requested_incompatible_qos (DataReader *reader*, RequestedIncompatibleQosStatus *status*)

Handles the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Implements **DataReaderListener** (p. 502).

8.35.2.3 void on_sample_rejected (DataReader *reader*, SampleRejectedStatus *status*)

Handles the StatusKind.SAMPLE_REJECTED_STATUS communication status.

Implements **DataReaderListener** (p. 503).

8.35.2.4 void on_liveliness_changed (DataReader *reader*, LivelinessChangedStatus *status*)

Handles the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

Implements **DataReaderListener** (p. 503).

8.35.2.5 void on_data_available (DataReader *reader*)

Handle the StatusKind.DATA_AVAILABLE_STATUS communication status.

Implements **DataReaderListener** (p. 503).

8.35.2.6 void on_sample_lost (DataReader *reader*, SampleLostStatus *status*)

Handles the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

Implements **DataReaderListener** (p. 503).

8.35.2.7 void on_subscription_matched (DataReader *reader*, SubscriptionMatchedStatus *status*)

Handles the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Implements **DataReaderListener** (p. 503).

8.36 DataReaderCacheStatus Class Reference

*<<**eXtension**>> (p. 270)* 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.

8.36.1 Detailed Description

*<<**eXtension**>> (p. 270)* The status of the reader's cache.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

8.36.2 Member Data Documentation

8.36.2.1 long sample_count_peak

The highest number of samples in the reader's queue over the lifetime of the reader.

8.36.2.2 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. 86)

8.37 DataReaderListener Interface Reference

<<interface>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for reader status.

Inheritance diagram for DataReaderListener::

Public Member Functions

- void **on_requested_deadline_missed** (**DataReader** reader, **RequestedDeadlineMissedStatus** status)

  Handles the StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

- void **on_requested_incompatible_qos** (**DataReader** reader, **RequestedIncompatibleQosStatus** status)

  Handles the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

- void **on_sample_rejected** (**DataReader** reader, **SampleRejectedStatus** status)

  Handles the StatusKind.SAMPLE_REJECTED_STATUS communication status.

- void **on_liveliness_changed** (**DataReader** reader, **LivelinessChangedStatus** status)

  Handles the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

- void **on_data_available** (**DataReader** reader)

  Handle the StatusKind.DATA_AVAILABLE_STATUS communication status.

- void **on_sample_lost** (**DataReader** reader, **SampleLostStatus** status)

  Handles the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

- void **on_subscription_matched** (**DataReader** reader, **SubscriptionMatchedStatus** status)

  Handles the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

8.37.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for reader status.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Status:

StatusKind.DATA_AVAILABLE_STATUS;
StatusKind.LIVELINESS_CHANGED_STATUS,
com.rti.dds.subscription.LivelinessChangedStatus (p. 1159);
StatusKind.REQUESTED_DEADLINE_MISSED_STATUS,
com.rti.dds.subscription.RequestedDeadlineMissedStatus (p. 1353);
StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS,
com.rti.dds.subscription.RequestedIncompatibleQosStatus (p. 1354);
StatusKind.SAMPLE_LOST_STATUS_STATUS,
com.rti.dds.subscription.SampleLostStatus (p. 1415);
StatusKind.SAMPLE_REJECTED_STATUS,
com.rti.dds.subscription.SampleRejectedStatus (p. 1422);
StatusKind.SUBSCRIPTION_MATCHED_STATUS,
com.rti.dds.subscription.SubscriptionMatchedStatus (p. 1520);

See also:

Status Kinds (p. 106)
Operations Allowed in Listener Callbacks (p. 1156)

8.37.2 Member Function Documentation

8.37.2.1 void on_requested_deadline_missed (DataReader *reader*, RequestedDeadlineMissedStatus *status*)

Handles the StatusKind.REQUESTED_DEADLINE_MISSED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 498).

8.37.2.2 void on_requested_incompatible_qos (DataReader *reader*, RequestedIncompatibleQosStatus *status*)

Handles the StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 498).

8.37.2.3 void on_sample_rejected (DataReader *reader*, SampleRejectedStatus *status*)

Handles the StatusKind.SAMPLE_REJECTED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 498).

8.37.2.4 void on_liveliness_changed (DataReader *reader*, LivelinessChangedStatus *status*)

Handles the StatusKind.LIVELINESS_CHANGED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 498).

8.37.2.5 void on_data_available (DataReader *reader*)

Handle the StatusKind.DATA_AVAILABLE_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 499).

8.37.2.6 void on_sample_lost (DataReader *reader*, SampleLostStatus *status*)

Handles the StatusKind.SAMPLE_LOST_STATUS_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 499).

8.37.2.7 void on_subscription_matched (DataReader *reader*, SubscriptionMatchedStatus *status*)

Handles the StatusKind.SUBSCRIPTION_MATCHED_STATUS communication status.

Implemented in **DataReaderAdapter** (p. 499).

8.38 DataReaderProtocolQosPolicy Class Reference

Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709) and **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 571), 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 InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE state without being in the InstanceStateKind.ALIVE_INTANCE_STATE state.

- final **RtpsReliableReaderProtocol_t rtps_reliable_reader**

  The reliable protocol defined in RTPS.

8.38.1 Detailed Description

Along with **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709) and **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 571), 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. 504) 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. 538) and **com.rti.dds.subscription.DataReader** (p. 473) 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. 1336) for more information on the per-DataReader/DataWriter reliability configuration. **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356) 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. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.38.2 Member Data Documentation

8.38.2.1 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. 473).

The association between a **com.rti.dds.subscription.DataReader** (p. 473) and its persisted state is done using the virtual GUID.

[**default**] com.rti.dds.infrastructure.GUID_t.AUTO

8.38.2.2 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. 1714)

[range] [0,0x00ffffff]

8.38.2.3 boolean expects_inline_qos

Specifies whether this DataReader expects inline QoS with every sample.

In RTI Connext, a **com.rti.dds.subscription.DataReader** (p. 473) nominally relies on Discovery to propagate QoS on a matched **com.rti.dds.publication.DataWriter** (p. 538). Alternatively, a **com.rti.dds.subscription.DataReader** (p. 473) may get information on a matched **com.rti.dds.publication.DataWriter** (p. 538) through QoS sent inline with a sample.

Asserting **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.expects_inline_qos** (p. 506) indicates to a matching **com.rti.dds.publication.DataWriter** (p. 538) that this **com.rti.dds.subscription.DataReader** (p. 473) expects to receive inline QoS with every sample. The complete set of inline QoS that a **com.rti.dds.publication.DataWriter** (p. 538) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Connext **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) cache Discovery information, inline QoS are largely redundant and thus unecessary. Only for other stateless implementations whose **com.rti.dds.subscription.DataReader** (p. 473) 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] false

8.38.2.4 boolean disable_positive_acks

Whether the reader sends positive acknowledgements to writers.

If set to true, the reader does not send positive acknowledgments (ACKs) in response to Heartbeat messages. The reader will send negative acknowledgements (NACKs) when a Heartbeat advertises samples that it has not received.

Otherwise, if set to false (the default), the reader will send ACKs to writers that expect ACKs (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_positive_acks** (p. 573) = false) and it will not send ACKs to writers that disable ACKs (**com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_positive_acks** (p. 573) = true)

[**default**] false

8.38.2.5 boolean propagate_dispose_of_unregistered_instances

Indicates whether or not an instance can move to the InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE state without being in the InstanceStateKind.ALIVE_INTANCE_STATE state.

This field only applies to keyed readers.

When the field is set to true, the DataReader will receive dispose notifications even if the instance is not alive.

To guarantee the key availability through the usage of the API com.rti.dds.topic.example.FooDataReader.get_key_value, this option should be used in combination **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.serialize_key_with_dispose** (p. 574) on the DataWriter that should be set to true.

[**default**] false

8.38.2.6 final RtpsReliableReaderProtocol_t rtps_reliable_reader

Initial value:

```
new RtpsReliableReaderProtocol_t()
```

The reliable protocol defined in RTPS.

[**default**] min_heartbeat_response_delay 0 seconds; max_heartbeat_response_delay 0.5 seconds

8.39 DataReaderProtocolStatus Class Reference

<<*eXtension*>> *(*p. 270*)* 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 user samples from a remote DataWriter received for the first time by a local ***DataReader*** *(*p. 473*).*

- long **received_sample_count_change**

  The incremental change in the number of user samples from a remote DataWriter received for the first time by a local ***DataReader*** *(*p. 473*) since the last time the status was read.*

- long **received_sample_bytes**

  The number of bytes of user samples from a remote DataWriter received for the first time by a local ***DataReader*** *(*p. 473*).*

- long **received_sample_bytes_change**

  The incremental change in the number of bytes of user samples from a remote DataWriter received for the first time by a local ***DataReader*** *(*p. 473*) 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. 473*).*

- long **duplicate_sample_count_change**

  The incremental change in the number of samples from a remote DataWriter received, not for the first time, by a local ***DataReader*** *(*p. 473*) 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. 473*).*

- long **duplicate_sample_bytes_change**

The incremental change in the number of bytes of samples from a remote DataWriter received, not for the first time, by a local ***DataReader*** *(p. 473) since the last time the status was read.*

^ long **filtered_sample_count**

The number of user samples filtered by the local ***DataReader*** *(p. 473) due to Content-Filtered Topics or Time-Based Filter.*

^ long **filtered_sample_count_change**

The incremental change in the number of user samples filtered by the local ***DataReader*** *(p. 473) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.*

^ long **filtered_sample_bytes**

The number of bytes of user samples filtered by the local ***DataReader*** *(p. 473) due to Content-Filtered Topics or Time-Based Filter.*

^ long **filtered_sample_bytes_change**

The incremental change in the number of bytes of user samples filtered by the local ***DataReader*** *(p. 473) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.*

^ long **received_heartbeat_count**

The number of Heartbeats from a remote DataWriter received by a local ***DataReader*** *(p. 473).*

^ long **received_heartbeat_count_change**

The incremental change in the number of Heartbeats from a remote DataWriter received by a local ***DataReader*** *(p. 473) 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. 473).*

^ long **received_heartbeat_bytes_change**

The incremental change in the number of bytes of Heartbeats from a remote DataWriter received by a local ***DataReader*** *(p. 473) since the last time the status was read.*

^ long **sent_ack_count**

The number of ACKs sent from a local ***DataReader*** *(p. 473) to a matching remote DataWriter.*

^ long **sent_ack_count_change**

*The incremental change in the number of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.*

^ long **sent_ack_bytes**

*The number of bytes of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.*

^ long **sent_ack_bytes_change**

*The incremental change in the number of bytes of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.*

^ long **sent_nack_count**

*The number of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.*

^ long **sent_nack_count_change**

*The incremental change in the number of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.*

^ long **sent_nack_bytes**

*The number of bytes of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.*

^ long **sent_nack_bytes_change**

*The incremental change in the number of bytes of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.*

^ long **received_gap_count**

*The number of GAPs received from remote DataWriter to this **DataReader** (p. 473).*

^ long **received_gap_count_change**

*The incremental change in the number of GAPs received from remote DataWriter to this **DataReader** (p. 473) 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. 473).*

^ long **received_gap_bytes_change**

The incremental change in the number of bytes of GAPs received from remote DataWriter to this ***DataReader*** *(p. 473) since the last time the status was read.*

- long **rejected_sample_count**

  The number of times a sample is rejected due to exceptions in the receive path.

- long **rejected_sample_count_change**

  The incremental change in the number of times a sample is rejected due to exceptions in the receive path 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'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.

8.39.1 Detailed Description

<<***eXtension***>> *(p. 270)* 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. 473)

8.39.2 Member Data Documentation

8.39.2.1 long received_sample_count

The number of user samples from a remote DataWriter received for the first time by a local **DataReader** (p. 473).

8.39.2.2 long received_sample_count_change

The incremental change in the number of user samples from a remote DataWriter received for the first time by a local **DataReader** (p. 473) since the last time the status was read.

8.39.2.3 long received_sample_bytes

The number of bytes of user samples from a remote DataWriter received for the first time by a local **DataReader** (p. 473).

8.39.2.4 long received_sample_bytes_change

The incremental change in the number of bytes of user samples from a remote DataWriter received for the first time by a local **DataReader** (p. 473) since the last time the status was read.

8.39.2.5 long duplicate_sample_count

The number of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 473).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.39.2.6 long duplicate_sample_count_change

The incremental change in the number of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 473) 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.

8.39.2.7 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. 473).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

8.39.2.8 long duplicate_sample_bytes_change

The incremental change in the number of bytes of samples from a remote DataWriter received, not for the first time, by a local **DataReader** (p. 473) 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.

8.39.2.9 long filtered_sample_count

The number of user samples filtered by the local **DataReader** (p. 473) due to Content-Filtered Topics or Time-Based Filter.

8.39.2.10 long filtered_sample_count_change

The incremental change in the number of user samples filtered by the local **DataReader** (p. 473) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.

8.39.2.11 long filtered_sample_bytes

The number of bytes of user samples filtered by the local **DataReader** (p. 473) due to Content-Filtered Topics or Time-Based Filter.

8.39.2.12 long filtered_sample_bytes_change

The incremental change in the number of bytes of user samples filtered by the local **DataReader** (p. 473) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.

8.39.2.13 long received_heartbeat_count

The number of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 473).

8.39.2.14 long received_heartbeat_count_change

The incremental change in the number of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 473) since the last time the status was read.

8.39.2.15 long received_heartbeat_bytes

The number of bytes of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 473).

8.39.2.16 long received_heartbeat_bytes_change

The incremental change in the number of bytes of Heartbeats from a remote DataWriter received by a local **DataReader** (p. 473) since the last time the status was read.

8.39.2.17 long sent_ack_count

The number of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.

8.39.2.18 long sent_ack_count_change

The incremental change in the number of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.

8.39.2.19 long sent_ack_bytes

The number of bytes of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.

8.39.2.20 long sent_ack_bytes_change

The incremental change in the number of bytes of ACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.

8.39.2.21 long sent_nack_count

The number of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.

8.39.2.22 long sent_nack_count_change

The incremental change in the number of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.

8.39.2.23 long sent_nack_bytes

The number of bytes of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter.

8.39.2.24 long sent_nack_bytes_change

The incremental change in the number of bytes of NACKs sent from a local **DataReader** (p. 473) to a matching remote DataWriter since the last time the status was read.

8.39.2.25 long received_gap_count

The number of GAPs received from remote DataWriter to this **DataReader** (p. 473).

8.39.2.26 long received_gap_count_change

The incremental change in the number of GAPs received from remote DataWriter to this **DataReader** (p. 473) since the last time the status was read.

8.39.2.27 long received_gap_bytes

The number of bytes of GAPs received from remote DataWriter to this **DataReader** (p. 473).

8.39.2.28 long received_gap_bytes_change

The incremental change in the number of bytes of GAPs received from remote DataWriter to this **DataReader** (p. 473) since the last time the status was read.

8.39.2.29 long rejected_sample_count

The number of times a sample is rejected due to exceptions in the receive path.

8.39.2.30 long rejected_sample_count_change

The incremental change in the number of times a sample is rejected due to exceptions in the receive path since the last time the status was read.

8.39.2.31 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.

8.39.2.32 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.

8.39.2.33 SequenceNumber_t last_committed_sample_sequence_number

Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.

Applicable only when retrieving matched DataWriter statuses.

For best-effort DataReaders, this is the sequence number of the latest sample received.

For reliable DataReaders, this is the sequence number of the latest sample that is available to be read or taken from the DataReader's queue.

8.39.2.34 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.

8.40 DataReaderQos Class Reference

QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 473) entity.

Inheritance diagram for DataReaderQos::

Public Attributes

- final **DurabilityQosPolicy durability**
  Durability policy, ***DURABILITY*** *(p. 65).*
- final **DeadlineQosPolicy deadline**
  Deadline policy, ***DEADLINE*** *(p. 50).*
- final **LatencyBudgetQosPolicy latency_budget**
  Latency budget policy, ***LATENCY_BUDGET*** *(p. 76).*
- final **LivelinessQosPolicy liveliness**
  Liveliness policy, ***LIVELINESS*** *(p. 78).*
- final **ReliabilityQosPolicy reliability**
  Reliability policy, ***RELIABILITY*** *(p. 101).*
- final **DestinationOrderQosPolicy destination_order**
  Destination order policy, ***DESTINATION_ORDER*** *(p. 51).*
- final **HistoryQosPolicy history**
  History policy, ***HISTORY*** *(p. 75).*
- final **ResourceLimitsQosPolicy resource_limits**
  Resource limits policy, ***RESOURCE_LIMITS*** *(p. 102).*
- final **UserDataQosPolicy user_data**
  User data policy, ***USER_DATA*** *(p. 126).*
- final **OwnershipQosPolicy ownership**
  Ownership policy, ***OWNERSHIP*** *(p. 83).*
- final **TimeBasedFilterQosPolicy time_based_filter**
  Time-based filter policy, ***TIME_BASED_FILTER*** *(p. 113).*

^ final **ReaderDataLifecycleQosPolicy reader_data_lifecycle**

Reader data lifecycle policy, ***READER_DATA_LIFECYCLE*** *(p. 99).*

^ final **DataReaderResourceLimitsQosPolicy reader_resource_limits**

<<**eXtension**>> (p. 270) ***com.rti.dds.subscription.DataReader*** *(p. 473) resource limits policy,* ***DATA_READER_RESOURCE_LIMITS*** *(p. 46). This policy is an extension to the DDS standard.*

^ final **DataReaderProtocolQosPolicy protocol**

<<**eXtension**>> (p. 270) ***com.rti.dds.subscription.DataReader*** *(p. 473) protocol policy,* ***DATA_READER_PROTOCOL*** *(p. 45)*

^ final **TransportSelectionQosPolicy transport_selection**

<<**eXtension**>> (p. 270) *Transport selection policy,* ***TRANSPORT_SELECTION*** *(p. 122).*

^ final **TransportUnicastQosPolicy unicast**

<<**eXtension**>> (p. 270) *Unicast transport policy,* ***TRANSPORT_UNICAST*** *(p. 123).*

^ final **TransportMulticastQosPolicy multicast**

<<**eXtension**>> (p. 270) *Multicast transport policy,* ***TRANSPORT_MULTICAST*** *(p. 118).*

^ final **PropertyQosPolicy property**

<<**eXtension**>> (p. 270) *Property policy,* ***PROPERTY*** *(p. 88).*

^ final **AvailabilityQosPolicy availability**

<<**eXtension**>> (p. 270) *Availability policy,* ***AVAILABILITY*** *(p. 41).*

^ final **EntityNameQosPolicy subscription_name**

<<**eXtension**>> (p. 270) *EntityName policy,* ***ENTITY_NAME*** *(p. 70).*

^ final **TypeSupportQosPolicy type_support**

<<**eXtension**>> (p. 270) *type support data,* ***TYPESUPPORT*** *(p. 124).*

8.40.1 Detailed Description

QoS policies supported by a **com.rti.dds.subscription.DataReader** (p. 473) entity.

You must set certain members in a consistent manner:

com.rti.dds.subscription.DataReaderQos.deadline.period >= com.rti.dds.subscription.DataReaderQos.time_based_filter.minimum_separation

com.rti.dds.subscription.DataReaderQos.history.depth <= com.rti.dds.subscription.DataReaderQos.resource_limits.max_samples_per_instance

com.rti.dds.subscription.DataReaderQos.resource_limits.max_samples_per_instance <= com.rti.dds.subscription.DataReaderQos.resource_limits.max_samples com.rti.dds.subscription.DataReaderQos.resource_limits.initial_samples <= com.rti.dds.subscription.DataReaderQos.resource_limits.max_samples

com.rti.dds.subscription.DataReaderQos.resource_limits.initial_instances <= com.rti.dds.subscription.DataReaderQos.resource_limits.max_instances

com.rti.dds.subscription.DataReaderQos.reader_resource_limits.initial_remote_writers_per_instance <= com.rti.dds.subscription.DataReaderQos.reader_resource_limits.max_remote_writers_per_instance

com.rti.dds.subscription.DataReaderQos.reader_resource_limits.initial_infos <= com.rti.dds.subscription.DataReaderQos.reader_resource_limits.max_infos

com.rti.dds.subscription.DataReaderQos.reader_resource_limits.max_remote_writers_per_instance <= com.rti.dds.subscription.DataReaderQos.reader_resource_limits.max_remote_writers

com.rti.dds.subscription.DataReaderQos.reader_resource_limits.max_samples_per_remote_writer <= com.rti.dds.subscription.DataReaderQos.resource_limits.max_samples

length of com.rti.dds.subscription.DataReaderQos.user_data.value <= com.rti.dds.domain.DomainParticipantQos.resource_limits.reader_user_data_max_length

If any of the above are not true, **com.rti.dds.subscription.DataReader.set_qos** (p. 480) and **com.rti.dds.subscription.DataReader.set_qos_with_profile** (p. 481) will fail with RETCODE_INCONSISTENT_POLICY

8.40.2 Member Data Documentation

8.40.2.1 final DurabilityQosPolicy durability

Durability policy, **DURABILITY** (p. 65).

8.40.2.2 final DeadlineQosPolicy deadline

Deadline policy, **DEADLINE** (p. 50).

8.40.2.3 final LatencyBudgetQosPolicy latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 76).

8.40.2.4 final LivelinessQosPolicy liveliness

Liveliness policy, **LIVELINESS** (p. 78).

8.40.2.5 final ReliabilityQosPolicy reliability

Reliability policy, **RELIABILITY** (p. 101).

8.40.2.6 final DestinationOrderQosPolicy destination_order

Destination order policy, **DESTINATION_ORDER** (p. 51).

8.40.2.7 final HistoryQosPolicy history

History policy, **HISTORY** (p. 75).

8.40.2.8 final ResourceLimitsQosPolicy resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 102).

8.40.2.9 final UserDataQosPolicy user_data

User data policy, **USER_DATA** (p. 126).

8.40.2.10 final OwnershipQosPolicy ownership

Ownership policy, **OWNERSHIP** (p. 83).

8.40.2.11 final TimeBasedFilterQosPolicy time_based_filter

Time-based filter policy, **TIME_BASED_FILTER** (p. 113).

8.40.2.12 final ReaderDataLifecycleQosPolicy reader_data_lifecycle

Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 99).

8.40.2.13 final DataReaderResourceLimitsQosPolicy reader_resource_limits

<<*eXtension*>> *(p. 270)* **com.rti.dds.subscription.DataReader** (p. 473) resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 46). This policy is an extension to the DDS standard.

8.40.2.14 final DataReaderProtocolQosPolicy protocol

<<*eXtension*>> *(p. 270)* **com.rti.dds.subscription.DataReader** (p. 473) protocol policy, **DATA_READER_PROTOCOL** (p. 45)

8.40.2.15 final TransportSelectionQosPolicy transport_selection

<<*eXtension*>> *(p. 270)* Transport selection policy, **TRANSPORT_SELECTION** (p. 122).

Specifies the transports available for use by the **com.rti.dds.subscription.DataReader** (p. 473).

8.40.2.16 final TransportUnicastQosPolicy unicast

<<*eXtension*>> *(p. 270)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 123).

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. 538) entities in the **domain** (p. 317).

8.40.2.17 final TransportMulticastQosPolicy multicast

<<*eXtension*>> *(p. 270)* Multicast transport policy, **TRANSPORT_MULTICAST** (p. 118).

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. 538) entities in the **domain** (p. 317).

8.40.2.18 final PropertyQosPolicy property

<<*eXtension*>> *(p. 270)* Property policy, **PROPERTY** (p. 88).

8.40.2.19 final AvailabilityQosPolicy availability

<<*eXtension*>> *(p. 270)* Availability policy, **AVAILABILITY** (p. 41).

8.40.2.20 final EntityNameQosPolicy subscription_name

<<*eXtension*>> *(p. 270)* EntityName policy, **ENTITY_NAME** (p. 70).

8.40.2.21 final TypeSupportQosPolicy type_support

<<*eXtension*>> *(p. 270)* type support data, **TYPESUPPORT** (p. 124).

Optional value that is passed to a type plugin's on_endpoint_attached and deserialization functions.

8.41 DataReaderResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473) 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. 538) that a **com.rti.dds.subscription.DataReader** (p. 473) may store when maintaining a reliable connection to the **com.rti.dds.publication.DataWriter** (p. 538).*

- int **max_infos**

  *The maximum number of info units that a **com.rti.dds.subscription.DataReader** (p. 473) can use to store **com.rti.dds.subscription.SampleInfo** (p. 1404).*

- int **initial_remote_writers**

  *The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) may read a single instance.*

- int **initial_infos**

*The initial number of info units that a **com.rti.dds.subscription.DataReader** (p. 473) can have, which are used to store **com.rti.dds.subscription.SampleInfo** (p. 1404).*

^ 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. 473) for which memory has not been returned by calling com.rti.dds.topic.example.FooDataReader.return_loan.*

^ 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. 473) for which memory has not been returned by calling com.rti.dds.topic.example.FooDataReader.return_loan.*

^ 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.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take. 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. 473) can receive fragmented samples.*

^ int **max_fragmented_samples**

*The maximum number of samples for which the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473) 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. 473) 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. 473) 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. 473) 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_query_condition_filters**

  The maximum number of query condition filters a reader is allowed.

Static Public Attributes

- static final int **AUTO_MAX_TOTAL_INSTANCES**

  <<**eXtension**>> (p. 270) *This value is used to make **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_total_instances** (p. 533) equal to **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360).*

8.41.1 Detailed Description

Various settings that configure how a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473). 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. 473).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.41.2 Member Data Documentation

8.41.2.1 int max_remote_writers

The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read, including all instances.

[**default**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[**range**] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= 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 **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

Note: For efficiency, set `max_remote_writers` >= **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_writers_per_instance** (p. 527).

8.41.2.2 int max_remote_writers_per_instance

The maximum number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read a single in-

stance.

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

[range] [1, 1024] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), <= max_remote_writers or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_remote_writers_per_instance

For unkeyed types, this value has to be equal to max_remote_writers if it is not **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

Note: For efficiency, set `max_remote_writers_per_instance` <= **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_remote_writers** (p. 527)

8.41.2.3 int max_samples_per_remote_writer

The maximum number of out-of-order samples from a given remote **com.rti.dds.publication.DataWriter** (p. 538) that a **com.rti.dds.subscription.DataReader** (p. 473) may store when maintaining a reliable connection to the **com.rti.dds.publication.DataWriter** (p. 538).

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

[range] [1, 100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359)

8.41.2.4 int max_infos

The maximum number of info units that a **com.rti.dds.subscription.DataReader** (p. 473) can use to store **com.rti.dds.subscription.SampleInfo** (p. 1404).

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] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_infos

8.41.2.5 int initial_remote_writers

The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) 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.DataReaderResourceLimitsQosPolicy.initial_remote_writers_per_instance** (p. 529).

8.41.2.6 int initial_remote_writers_per_instance

The initial number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) 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.DataReaderResourceLimitsQosPolicy.initial_remote_writers** (p. 529).

8.41.2.7 int initial_infos

The initial number of info units that a **com.rti.dds.subscription.DataReader** (p. 473) can have, which are used to store **com.rti.dds.subscription.SampleInfo** (p. 1404).

[default] 32

[range] [1,1 million], <= max_infos

8.41.2.8 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. 473) for which memory has not been returned by calling com.rti.dds.topic.example.FooDataReader.return_loan.

[default] 2

[range] [1, 65536], <= max_outstanding_reads

8.41.2.9 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. 473) for which memory has not been returned by calling com.rti.dds.topic.example.FooDataReader.return_loan.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 65536] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_outstanding_reads

8.41.2.10 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.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take. 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. 401) for more information about this feature.) When polling for data or using a **com.rti.dds.infrastructure.WaitSet** (p. 1695), 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]

8.41.2.11 boolean disable_fragmentation_support

Determines whether the **com.rti.dds.subscription.DataReader** (p. 473) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] false

8.41.2.12 int max_fragmented_samples

The maximum number of samples for which the **com.rti.dds.subscription.DataReader** (p. 473) may store fragments at a given point in time.

At any given time, a **com.rti.dds.subscription.DataReader** (p. 473) 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. 538) 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. 1359).

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. 530) is false.

[default] 1024

[range] [1, 1 million]

8.41.2.13 int initial_fragmented_samples

The initial number of samples for which a **com.rti.dds.subscription.DataReader** (p. 473) may store fragments.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 530) is false.

[default] 4

[range] [1,1024], <= max_fragmented_samples

8.41.2.14 int max_fragmented_samples_per_remote_writer

The maximum number of samples per remote writer for which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 530) is false.

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

8.41.2.15 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. 530) is false.

[default] 512

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.41.2.16 boolean dynamically_allocate_fragmented_samples

Determines whether the **com.rti.dds.subscription.DataReader** (p. 473) pre-allocates storage for storing fragmented samples.

By default, the middleware will allocate memory upfront for storing fragments for up to **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.initial_fragmented_samples** (p. 531) samples. This memory may grow up to **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_fragmented_samples** (p. 531) if needed.

If `dynamically_allocate_fragmented_samples` is set to true, 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 may be 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 run-time.

Only applies if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.disable_fragmentation_support** (p. 530) is false.

[default] false

8.41.2.17 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. 1360).

These instances have associated DataWriters or samples in the DataReader's queue and are visible to the user through operations such as com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.read, and com.rti.dds.topic.example.FooDataReader.get_key_value.

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.ResourceLimitsQosPolicy.max_instances** (p. 1360). If the limit is exceeded, RTI Connext will drop the sample and report it as lost and rejected. 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.

[default] **DataReaderResourceLimitsQosPolicy.AUTO_MAX_TOTAL_INSTANCES** (p. 46)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) or **DataReaderResourceLimitsQosPolicy.AUTO_MAX_TOTAL_INSTANCES** (p. 46), >= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360)

8.41.2.18 int max_remote_virtual_writers

The maximum number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read, including all instances.

When **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope**

(p. 1241) is set to **PresentationQosPolicyAccessScopeKind.GROUP_-PRESENTATION_QOS** (p. 1243), this value determines the maximum number of DataWriter groups that can be managed by the **com.rti.dds.subscription.Subscriber** (p. 1478) containing this **com.rti.dds.subscription.DataReader** (p. 473).

Since the **com.rti.dds.subscription.Subscriber** (p. 1478) may contain more than one **com.rti.dds.subscription.DataReader** (p. 473), only the setting of the first applies.

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_-UNLIMITED** (p. 102), >= initial_remote_virtual_writers, >= max_remote_-virtual_writers_per_instance

8.41.2.19 int initial_remote_virtual_writers

The initial number of remote virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read, including all instances.

[default] 2

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_-UNLIMITED** (p. 102), <= max_remote_virtual_writers

8.41.2.20 int max_remote_virtual_writers_per_instance

The maximum number of virtual remote writers that can be associated with an instance.

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

[range] [1, 1024] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= 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.

8.41.2.21 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.

8.41.2.22 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.

8.42 DataReaderSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.DataReader** (p. 473) > .

Inherits AbstractNativeSequence.

Public Member Functions

- ^ **DataReaderSeq** (Collection readers)
- ^ int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.42.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.DataReader** (p. 473) > .

See also:

com.rti.dds.util.Sequence (p. 1432)

8.42.2 Constructor & Destructor Documentation

8.42.2.1 DataReaderSeq (Collection *readers*)

Exceptions:

NullPointerException if the given collection is null

8.42.3 Member Function Documentation

8.42.3.1 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. 383), or explicitly by calling **Sequence.setMaximum** (p. 1433).

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.43 DataWriter Interface Reference

<<*interface*>> *(p. 271)* Allows an application to set the value of the data to be published under a given **com.rti.dds.topic.Topic** (p. 1545).

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. 270) *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 StatusKind.LIVELINESS_LOST_STATUS communication status.

- void **get_offered_deadline_missed_status** (**OfferedDeadlineMissedStatus** status)

  Accesses the StatusKind.OFFERED_DEADLINE_MISSED_STATUS communication status.

- void **get_offered_incompatible_qos_status** (**OfferedIncompatibleQosStatus** status)

  Accesses the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS communication status.

- void **get_publication_matched_status** (**PublicationMatchedStatus** status)

Accesses the StatusKind.PUBLICATION_MATCHED_STATUS communication status.

^ void **get_reliable_writer_cache_changed_status** (**ReliableWriterCacheChangedStatus** status)

<<**eXtension**>> (p. 270) *Get the reliable cache status for this writer.*

^ void **get_reliable_reader_activity_changed_status** (**ReliableReaderActivityChangedStatus** status)

<<**eXtension**>> (p. 270) *Get the reliable reader activity changed status for this writer.*

^ void **get_datawriter_cache_status** (**DataWriterCacheStatus** status)

<<**eXtension**>> (p. 270) *Get the datawriter cache status for this writer.*

^ void **get_datawriter_protocol_status** (**DataWriterProtocolStatus** status)

<<**eXtension**>> (p. 270) *Get the datawriter protocol status for this writer.*

^ void **get_matched_subscription_datawriter_protocol_status** (**DataWriterProtocolStatus** status, **InstanceHandle_t** subscription_handle)

<<**eXtension**>> (p. 270) *Get the datawriter protocol status for this writer, per matched **subscription** (p. 343) identified by the subscription_handle.*

^ void **get_matched_subscription_datawriter_protocol_status_by_locator** (**DataWriterProtocolStatus** status, **Locator_t** locator)

<<**eXtension**>> (p. 270) *Get the datawriter protocol status for this writer, per matched **subscription** (p. 343) identified by the locator.*

^ void **get_matched_subscription_locators** (**LocatorSeq** locators)

<<**eXtension**>> (p. 270) *Retrieve the list of locators for subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 538).*

^ void **get_matched_subscriptions** (**InstanceHandleSeq** subscription_handles)

*Retrieve the list of subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 538).*

^ void **get_matched_subscription_data** (**SubscriptionBuiltinTopicData** subscription_data, **InstanceHandle_t** subscription_handle)

*This operation retrieves the information on a **subscription** (p. 343) that is currently "associated" with the **com.rti.dds.publication.DataWriter** (p. 538).*

^ **Topic get_topic** ()

*This operation returns the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.publication.DataWriter** (p. 538).*

^ **Publisher get_publisher** ()

*This operation returns the **com.rti.dds.publication.Publisher** (p. 1277) to which the **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) entity is acknowledged, or until timeout expires.*

^ void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

<<**eXtension**>> (p. 270) *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. 538).*

^ void **flush** ()

<<**eXtension**>> (p. 270) *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.

8.43.1 Detailed Description

<<***interface***>> *(p. 271)* Allows an application to set the value of the data to be published under a given **com.rti.dds.topic.Topic** (p. 1545).

QoS:

com.rti.dds.publication.DataWriterQos (p. 588)

Status:

StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.LivelinessLostStatus** (p. 1162);
StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1212);
StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1214);

StatusKind.PUBLICATION_MATCHED_STATUS,
com.rti.dds.publication.PublicationMatchedStatus (p. 1274);
StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS,
com.rti.dds.publication.ReliableReaderActivityChangedStatus (p. 1342);
StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS,
com.rti.dds.publication.ReliableWriterCacheChangedStatus

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

A **com.rti.dds.publication.DataWriter** (p. 538) is attached to exactly one **com.rti.dds.publication.Publisher** (p. 1277), that acts as a factory for it.

A **com.rti.dds.publication.DataWriter** (p. 538) is bound to exactly one **com.rti.dds.topic.Topic** (p. 1545) and therefore to exactly one data type. The **com.rti.dds.topic.Topic** (p. 1545) must exist prior to the **com.rti.dds.publication.DataWriter** (p. 538)'s creation.

com.rti.dds.publication.DataWriter (p. 538) is an abstract class. It must be specialized for each particular application data-type (see **USER_DATA** (p. 126)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type `Foo` are specified in the **example** (p. 342) type **com.rti.dds.publication.DataWriter** (p. 538).

The following operations may be called even if the **com.rti.dds.publication.DataWriter** (p. 538) is not enabled. Other operations will fail with RETCODE_NOT_ENABLED if called on a disabled **com.rti.dds.publication.DataWriter** (p. 538):

- The base-class operations **com.rti.dds.publication.DataWriter.set_qos** (p. 543), **com.rti.dds.publication.DataWriter.get_qos** (p. 544), **com.rti.dds.publication.DataWriter.set_listener** (p. 545), **com.rti.dds.publication.DataWriter.get_listener** (p. 545), **com.rti.dds.infrastructure.Entity.enable** (p. 915), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 917) and **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917)
- **com.rti.dds.publication.DataWriter.get_liveliness_lost_status** (p. 546) **com.rti.dds.publication.DataWriter.get_offered_deadline_missed_status** (p. 546) **com.rti.dds.publication.DataWriter.get_offered_incompatible_qos_status** (p. 546) **com.rti.dds.publication.DataWriter.get_publication_matched_status** (p. 547) **com.rti.dds.publication.DataWriter.get_reliable_writer_cache_changed_status** (p. 547) **com.rti.dds.publication.DataWriter.get_reliable_reader_activity_changed_status** (p. 547)

Several **com.rti.dds.publication.DataWriter** (p. 538) may operate in different threads. If they share the same **com.rti.dds.publication.Publisher** (p. 1277), the middleware guarantees that its operations are thread-safe.

See also:

com.rti.dds.topic.example.FooDataWriter
Operations Allowed in Listener Callbacks (p. 1156)

8.43.2 Member Function Documentation

8.43.2.1 void set_qos (DataWriterQos *qos*)

Sets the writer QoS.

This operation modifies the QoS of the **com.rti.dds.publication.DataWriter** (p. 538).

The **com.rti.dds.publication.DataWriterQos.user_data** (p. 592), **com.rti.dds.publication.DataWriterQos.deadline** (p. 591), **com.rti.dds.publication.DataWriterQos.latency_budget** (p. 591), **com.rti.dds.publication.DataWriterQos.ownership_strength** (p. 592), **com.rti.dds.publication.DataWriterQos.transport_priority** (p. 592), **com.rti.dds.publication.DataWriterQos.lifespan** (p. 592) and **com.rti.dds.publication.DataWriterQos.writer_data_lifecycle** (p. 592) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 271)* The **com.rti.dds.publication.DataWriterQos** (p. 588) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.publication.DataWriter** (p. 538) is enabled. The special value **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) can be used to indicate that the QoS of the **com.rti.dds.publication.DataWriter** (p. 538) should be changed to match the current default **com.rti.dds.publication.DataWriterQos** (p. 588) set in the **com.rti.dds.publication.Publisher** (p. 1277). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY or RETCODE_INCONSISTENT_POLICY

See also:

com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS

set_qos (abstract) (p. 913)
Operations Allowed in Listener Callbacks (p. 1156)

8.43.2.2 void set_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* 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. 538).

The **com.rti.dds.publication.DataWriterQos.user_data** (p. 592), **com.rti.dds.publication.DataWriterQos.deadline** (p. 591), **com.rti.dds.publication.DataWriterQos.latency_budget** (p. 591), **com.rti.dds.publication.DataWriterQos.ownership_strength** (p. 592), **com.rti.dds.publication.DataWriterQos.transport_priority** (p. 592), **com.rti.dds.publication.DataWriterQos.lifespan** (p. 592) and **com.rti.dds.publication.DataWriterQos.writer_data_lifecycle** (p. 592) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>>* *(p. 271)* 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. 1291)).

profile_name *<<in>>* *(p. 271)* 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. 1292)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY or RETCODE_INCONSISTENT_POLICY

See also:

com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1156)

8.43.2.3 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. 271)* The **com.rti.dds.publication.DataWriterQos** (p. 588) to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.43.2.4 void set_listener (DataWriterListener *l*, int *mask*)

Sets the writer listener.

Parameters:

l *<<in>>* *(p. 271)* **com.rti.dds.publication.DataWriterListener** (p. 566) to set to

mask *<<in>>* *(p. 271)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.publication.DataWriterListener** (p. 566).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.43.2.5 DataWriterListener get_listener ()

Get the writer listener.

Returns:

com.rti.dds.publication.DataWriterListener (p. 566) of the **com.rti.dds.publication.DataWriter** (p. 538).

See also:

get_listener (abstract) (p. 915)

8.43.2.6 void get_liveliness_lost_status (LivelinessLostStatus *status*)

Accesses the StatusKind.LIVELINESS_LOST_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.LivelinessLostStatus** (p. 1162) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.7 void get_offered_deadline_missed_status (OfferedDeadlineMissedStatus *status*)

Accesses the StatusKind.OFFERED_DEADLINE_MISSED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1212) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.8 void get_offered_incompatible_qos_status (OfferedIncompatibleQosStatus *status*)

Accesses the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1214) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.9 void get_publication_matched_status (PublicationMatchedStatus *status*)

Accesses the StatusKind.PUBLICATION_MATCHED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.PublicationMatchedStatus** (p. 1274) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.10 void get_reliable_writer_cache_changed_status (ReliableWriterCacheChangedStatus *status*)

*<<**eXtension**>>* *(p. 270)* Get the reliable cache status for this writer.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1345) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.11 void get_reliable_reader_activity_changed_status (ReliableReaderActivityChangedStatus *status*)

*<<**eXtension**>>* *(p. 270)* Get the reliable reader activity changed status for this writer.

Parameters:

status *<<inout>>* *(p. 271)* **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1342) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.43.2.12 void get_datawriter_cache_status (DataWriterCacheStatus *status*)

<<*eXtension*>> *(p. 270)* Get the datawriter cache status for this writer.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.publication.DataWriterCacheStatus** (p. 565) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.43.2.13 void get_datawriter_protocol_status (DataWriterProtocolStatus *status*)

<<*eXtension*>> *(p. 270)* Get the datawriter protocol status for this writer.

Parameters:

status <<*inout*>> *(p. 271)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 576) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.43.2.14 void get_matched_subscription_datawriter_protocol_status (DataWriterProtocolStatus *status*, InstanceHandle_t *subscription_handle*)

<<*eXtension*>> *(p. 270)* Get the datawriter protocol status for this writer, per matched **subscription** (p. 343) identified by the subscription_handle.

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. 271)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 576) to be filled in. Cannot be NULL.

subscription_handle *<<in>>* *(p. 271)* Handle to a specific **subscription** (p. 343) associated with the **com.rti.dds.subscription.DataReader** (p. 473). Cannot be NULL.Must correspond to a **subscription** (p. 343) currently associated with the **com.rti.dds.publication.DataWriter** (p. 538).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.43.2.15 void get_matched_subscription_datawriter_protocol_status_by_locator (DataWriterProtocolStatus *status*, Locator_t *locator*)

*<<**eXtension**>>* *(p. 270)* Get the datawriter protocol status for this writer, per matched **subscription** (p. 343) identified by the locator.

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. 271)* **com.rti.dds.publication.DataWriterProtocolStatus** (p. 576) to be filled in Cannot be NULL.

locator *<<in>>* *(p. 271)* Locator to a specific locator associated with the **com.rti.dds.subscription.DataReader** (p. 473). Cannot be NULL.Must correspond to a locator of one or more subscriptions currently associated with the **com.rti.dds.publication.DataWriter** (p. 538).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.43.2.16 void get_matched_subscription_locators (LocatorSeq *locators*)

*<<**eXtension**>>* *(p. 270)* Retrieve the list of locators for subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 538).

Matched **subscription** (p. 343) locators include locators for all those subscriptions in the same **domain** (p. 317) that have a matching **com.rti.dds.topic.Topic** (p. 1545), compatible QoS and common partition that the **com.rti.dds.domain.DomainParticipant**

(p. 629) has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 689) operation.

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. 473) entities.

Parameters:

locators *<<inout>> (p. 271).* Handles of all the matched **subscription** (p. 343) 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 RETCODE_OUT_OF_RESOURCES. Cannot be NULL..

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_OUT_OF_RESOURCES if the sequence is too small and the system can not resize it, or RETCODE_NOT_ENABLED

8.43.2.17 void get_matched_subscriptions (InstanceHandleSeq *subscription_handles*)

Retrieve the list of subscriptions currently "associated" with this **com.rti.dds.publication.DataWriter** (p. 538).

Matched subscriptions include all those in the same **domain** (p. 317) that have a matching **com.rti.dds.topic.Topic** (p. 1545), compatible QoS and common partition that the **com.rti.dds.domain.DomainParticipant** (p. 629) has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 689) operation.

The handles returned in the `subscription_handles` list are the ones that are used by the DDS implementation to locally identify the corresponding matched **com.rti.dds.subscription.DataReader** (p. 473) entities. These handles match the ones that appear in the **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1410) field of the **com.rti.dds.subscription.SampleInfo** (p. 1404) when reading the SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME **builtin** (p. 341) **topic** (p. 350).

Parameters:

subscription_handles *<<**inout**>>* *(p. 271).* 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 the matches and the system can not resize the sequence, this method will fail with RETCODE_OUT_OF_RESOURCES.

The maximum number of matches possible is configured with **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy** (p. 741). 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. 104), or RETCODE_OUT_OF_RESOURCES if the sequence is too small and the system can not resize it, or RETCODE_NOT_ENABLED

8.43.2.18 void get_matched_subscription_data (SubscriptionBuiltinTopicData *subscription_data*, InstanceHandle_t *subscription_handle*)

This operation retrieves the information on a **subscription** (p. 343) that is currently "associated" with the **com.rti.dds.publication.DataWriter** (p. 538).

The `subscription_handle` must correspond to a **subscription** (p. 343) currently associated with the **com.rti.dds.publication.DataWriter** (p. 538). Otherwise, the operation will fail and fail with RETCODE_BAD_PARAMETER. Use **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 550) to find the subscriptions that are currently matched with the **com.rti.dds.publication.DataWriter** (p. 538).

Note: This operation does not retrieve the following information in builtin.SubscriptionBuiltinTopicData:

- builtin.SubscriptionBuiltinTopicData.type_code
- builtin.SubscriptionBuiltinTopicData.property
- builtin.SubscriptionBuiltinTopicData.content_filter_property

The above information is available through **com.rti.dds.subscription.DataReaderListener.on_data_-**

available() (p. 503) (if a reader listener is installed on the builtin.SubscriptionBuiltinTopicDataDataReader).

Parameters:

subscription_data *<<inout>>* *(p. 271)*. The information to be filled in on the associated **subscription** (p. 343). Cannot be NULL.

subscription_handle *<<in>>* *(p. 271)*. Handle to a specific **subscription** (p. 343) associated with the **com.rti.dds.subscription.DataReader** (p. 473). Cannot be NULL.. Must correspond to a **subscription** (p. 343) currently associated with the **com.rti.dds.publication.DataWriter** (p. 538).

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_NOT_ENABLED

8.43.2.19 Topic get_topic ()

This operation returns the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.publication.DataWriter** (p. 538).

This is the same **com.rti.dds.topic.Topic** (p. 1545) that was used to create the **com.rti.dds.publication.DataWriter** (p. 538).

Returns:

com.rti.dds.topic.Topic (p. 1545) that was used to create the **com.rti.dds.publication.DataWriter** (p. 538).

8.43.2.20 Publisher get_publisher ()

This operation returns the **com.rti.dds.publication.Publisher** (p. 1277) to which the **com.rti.dds.publication.DataWriter** (p. 538) belongs.

Returns:

com.rti.dds.publication.Publisher (p. 1277) to which the **com.rti.dds.publication.DataWriter** (p. 538) belongs.

8.43.2.21 void wait_for_acknowledgments (Duration_t *max_wait*)

Blocks the calling thread until all data written by reliable **com.rti.dds.publication.DataWriter** (p. 538) 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.DataWriter** (p. 538) entity is acknowledged by all matched reliable **com.rti.dds.subscription.DataReader** (p. 473) entities, or else 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; a time out indicates that max_wait elapsed before all the data was acknowledged.

If the **com.rti.dds.publication.DataWriter** (p. 538) does not have **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1336) kind set to RELIABLE the operation will complete immediately with RETCODE_OK.

Parameters:

max_wait *<<in>> (p. 271)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 776) .

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NOT_ENABLED, RETCODE_TIMEOUT

8.43.2.22 void wait_for_asynchronous_publishing (Duration_t *max_wait*)

*<<**eXtension**>> (p. 270)* 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. 538) is sent and acknowledged (if reliable) by all matched **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) is complete in addition to what **com.rti.dds.publication.DataWriter.wait_for_acknowledgments** (p. 553) provides.

If the **com.rti.dds.publication.DataWriter** (p. 538) does not have **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1308) kind set to PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS the operation will complete immediately with RETCODE_OK.

Parameters:

max_wait *<<in>> (p. 271)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 776) .

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NOT_ENABLED, RETCODE_TIMEOUT

8.43.2.23 void assert_liveliness ()

This operation manually asserts the liveliness of this **com.rti.dds.publication.DataWriter** (p. 538).

This is used in combination with the **LIVELINESS** (p. 78) policy to indicate to RTI Connext that the **com.rti.dds.publication.DataWriter** (p. 538) remains active.

You only need to use this operation if the **LIVELINESS** (p. 78) setting is either LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS or LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS. Otherwise, it has no effect.

Note: writing data via the com.rti.dds.topic.example.FooDataWriter.write or com.rti.dds.topic.example.FooDataWriter.write_w_timestamp operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 538) itself, and its **com.rti.dds.domain.DomainParticipant** (p. 629). Consequently the use of **assert_liveliness()** (p. 554) is only needed if the application is not writing data regularly.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED

See also:

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1164)

8.43.2.24 void flush ()

<<*eXtension*>> *(p. 270)* 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. 538) does not have **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1308) kind set to 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. 538) does have **com.rti.dds.infrastructure.PublishModeQosPolicy** (p. 1308) kind set to 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 than com.rti.dds.topic.example.FooDataWriter.write.

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. 555) is only thread-safe with batching if **com.rti.dds.infrastructure.BatchQosPolicy.thread_safe_write** (p. 404) is TRUE.

Exceptions:

> ***One*** of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

8.43.2.25 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.register_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.unregister_instance_untyped (p. 556)
com.rti.dds.topic.example.FooDataWriter.register_instance

8.43.2.26 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.unregister_instance_w_-timestamp_untyped (p. 557)
com.rti.dds.topic.example.FooDataWriter.register_instance

8.43.2.27 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.unregister_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.register_instance_untyped (p. 555)
com.rti.dds.topic.example.FooDataWriter.unregister_instance

8.43.2.28 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate 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. 556)
FooDataWriter.unregister_instance_w_timestamp

8.43.2.29 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.write method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.write_w_timestamp_untyped (p. 557)
com.rti.dds.topic.example.FooDataWriter.write

8.43.2.30 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.write_w_timestamp method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.write_untyped (p. 557)
com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.43.2.31 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.dispose method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.dispose_w_timestamp_-untyped (p. 558)
com.rti.dds.topic.example.FooDataWriter.dispose

8.43.2.32 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.publication.DataWriter.dispose_untyped (p. 558)
com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp

8.43.2.33 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.get_key_value method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.43.2.34 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.dds.topic.example.FooDataWriter classes in a consistent way.

Statically type-safe code should use the appropriate com.rti.dds.topic.example.FooDataWriter.lookup_instance method instead of this one. See that method for detailed documentation.

See also:

com.rti.dds.topic.example.FooDataWriter.lookup_instance

8.44 DataWriterAdapter Class Reference

*<<**eXtension**>> (p. 270)* 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 StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

- void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

- void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the StatusKind.LIVELINESS_LOST_STATUS status.

- void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

- void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  <<**eXtension**>> (p. 270) *A change has occurred in the writer's cache of unacknowledged samples.*

- void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  <<**eXtension**>> (p. 270) *A matched reliable reader has become active or become inactive.*

- void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

  *Notifies when an instance is replaced in **DataWriter** (p. 538) queue.*

8.44.1 Detailed Description

<<*eXtension*>> *(p. 270)* 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.

8.44.2 Member Function Documentation

8.44.2.1 void on_offered_deadline_missed (DataWriter *writer*, OfferedDeadlineMissedStatus *status*)

Handles the StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **DEADLINE** (p. 50) 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. 538) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 567).

8.44.2.2 void on_offered_incompatible_qos (DataWriter *writer*, OfferedIncompatibleQosStatus *status*)

Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 588) of the **com.rti.dds.publication.DataWriter** (p. 538) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 473). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 538) has discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 568).

8.44.2.3 void on_liveliness_lost (DataWriter *writer*, LivelinessLostStatus *status*)

Handles the StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **LIVELINESS** (p. 78) qos policy was not respected; this **com.rti.dds.subscription.DataReader** (p. 473) entities will consider the **com.rti.dds.publication.DataWriter** (p. 538) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 538) simply renames not alive for another liveliness period.

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 568).

8.44.2.4 void on_publication_matched (DataWriter *writer*, PublicationMatchedStatus *status*)

Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 538) has found a **com.rti.dds.subscription.DataReader** (p. 473) that matches the **com.rti.dds.topic.Topic** (p. 1545), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 473) that was previously considered to be matched.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current **publication** (p. 338) match status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 569).

8.44.2.5 void on_reliable_writer_cache_changed (DataWriter *writer*, ReliableWriterCacheChangedStatus *status*)

<<*eXtension*>> *(p. 270)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 569).

8.44.2.6 void on_reliable_reader_activity_changed (DataWriter *writer*, ReliableReaderActivityChangedStatus *status*)

<<*eXtension*>> *(p. 270)* A matched reliable reader has become active or become inactive.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 570).

8.44.2.7 void on_instance_replaced (DataWriter *writer*, InstanceHandle_t *handle*)

Notifies when an instance is replaced in **DataWriter** (p. 538) queue.

This callback is called when an instance is replaced by the **com.rti.dds.publication.DataWriter** (p. 538) 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.

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

handle <<***out***>> *(p. 271)* Handle of the replaced instance

Implements **DataWriterListener** (p. 570).

8.45 DataWriterCacheStatus Class Reference

<<*eXtension*>> *(p. 270)* The status of the writer's cache.

Inherits Status.

Public Attributes

- long **sample_count_peak**

  Highest number of samples in the writer's queue over the lifetime of the writer.

- long **sample_count**

  Number of samples in the writer's queue.

8.45.1 Detailed Description

<<*eXtension*>> *(p. 270)* The status of the writer's cache.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

8.45.2 Member Data Documentation

8.45.2.1 long sample_count_peak

Highest number of samples in the writer's queue over the lifetime of the writer.

8.45.2.2 long sample_count

Number of samples in the writer's queue.

8.46 DataWriterListener Interface Reference

<<*interface*>> *(*p. *271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for writer status.

Inheritance diagram for DataWriterListener::

Public Member Functions

- void **on_offered_deadline_missed** (**DataWriter** writer, **OfferedDeadlineMissedStatus** status)

  Handles the StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

- void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

- void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the StatusKind.LIVELINESS_LOST_STATUS status.

- void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

- void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  <<**eXtension**>> (p. 270) *A change has occurred in the writer's cache of unacknowledged samples.*

- void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  <<**eXtension**>> (p. 270) *A matched reliable reader has become active or become inactive.*

- void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

  Notifies when an instance is replaced in ***DataWriter*** *(*p. *538) queue.*

8.46.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for writer status.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Status:

StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.LivelinessLostStatus** (p. 1162);
StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1212);
StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1214);
StatusKind.PUBLICATION_MATCHED_STATUS, **com.rti.dds.publication.PublicationMatchedStatus** (p. 1274);
StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1342);
StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS, **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1345);

See also:

Status Kinds (p. 106)
Operations Allowed in Listener Callbacks (p. 1156)

8.46.2 Member Function Documentation

8.46.2.1 void on_offered_deadline_missed (DataWriter *writer*, OfferedDeadlineMissedStatus *status*)

Handles the StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **DEADLINE** (p. 50) 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. 538) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 271)* Locally created

com.rti.dds.publication.DataWriter (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 704), and **DataWriterAdapter** (p. 561).

8.46.2.2 void on_offered_incompatible_qos (DataWriter *writer*, OfferedIncompatibleQosStatus *status*)

Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 588) of the **com.rti.dds.publication.DataWriter** (p. 538) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 473). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 538) has discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 705), and **DataWriterAdapter** (p. 561).

8.46.2.3 void on_liveliness_lost (DataWriter *writer*, LivelinessLostStatus *status*)

Handles the StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **LIVELINESS** (p. 78) qos policy was not respected; this **com.rti.dds.subscription.DataReader** (p. 473) entities will consider the **com.rti.dds.publication.DataWriter** (p. 538) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 538) simply renames not alive for another liveliness period.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 705), and **DataWriterAdapter** (p. 562).

8.46.2.4 void on_publication_matched (DataWriter *writer*, PublicationMatchedStatus *status*)

Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 538) has found a **com.rti.dds.subscription.DataReader** (p. 473) that matches the **com.rti.dds.topic.Topic** (p. 1545), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 473) that was previously considered to be matched.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current **publication** (p. 338) match status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 706), and **DataWriterAdapter** (p. 562).

8.46.2.5 void on_reliable_writer_cache_changed (DataWriter *writer*, ReliableWriterCacheChangedStatus *status*)

<<*eXtension*>> *(p. 270)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status *<<**out**>>* *(p. 271)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 707), and **DataWriterAdapter** (p. 563).

8.46.2.6 void on_reliable_reader_activity_changed (DataWriter *writer*, ReliableReaderActivityChangedStatus *status*)

*<<**eXtension**>>* *(p. 270)* A matched reliable reader has become active or become inactive.

Parameters:

writer *<<**out**>>* *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status *<<**out**>>* *(p. 271)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implemented in **DomainParticipantAdapter** (p. 706), and **DataWriterAdapter** (p. 563).

8.46.2.7 void on_instance_replaced (DataWriter *writer*, InstanceHandle_t *handle*)

Notifies when an instance is replaced in **DataWriter** (p. 538) queue.

This callback is called when an instance is replaced by the **com.rti.dds.publication.DataWriter** (p. 538) 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.

Parameters:

writer *<<**out**>>* *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

handle *<<**out**>>* *(p. 271)* Handle of the replaced instance

Implemented in **DomainParticipantAdapter** (p. 707), and **DataWriterAdapter** (p. 564).

8.47 DataWriterProtocolQosPolicy Class Reference

Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 538) 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.

- final **RtpsReliableWriterProtocol_t rtps_reliable_writer**

  The reliable protocol defined in RTPS.

8.47.1 Detailed Description

Protocol that applies only to **com.rti.dds.publication.DataWriter** (p. 538) 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. 571) 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. 538) and **com.rti.dds.subscription.DataReader** (p. 473) 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. 1336) for more information on the per-DataReader/DataWriter reliability configuration. **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356) 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. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.47.2 Member Data Documentation

8.47.2.1 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. 538).

RTI Connext uses the virtual GUID to associate a persisted writer history to a specific **com.rti.dds.publication.DataWriter** (p. 538).

The RTI Connext Persistence Service uses the virtual GUID to send samples on behalf of the original **com.rti.dds.publication.DataWriter** (p. 538).

[**default**] com.rti.dds.infrastructure.GUID_t.AUTO

8.47.2.2 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. 1714)

[**range**] [0,0x00ffffff]

8.47.2.3 boolean push_on_write

Whether to push sample out when write is called.

If set to true (the default), the writer will send a sample every time write is called. Otherwise, the sample is put into the queue waiting for a NACK from remote reader(s) to be sent out.

Note: push_on_write must be TRUE for Asynchronous DataWriters (those with **PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1312)). Otherwise, samples will never be sent.

[**default**] true

8.47.2.4 boolean disable_positive_acks

Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to true, the writer does not expect readers to send send positive acknowledgments to the writer. Consequently, instead of keeping a sample queued until all readers have positively acknowledged it, the writer will keep a sample for at least **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_min_sample_keep_duration** (p. 1387), after which the sample is logically considered as positively acknowledged.

If set to false (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 507) = false) and it applies the keep-duration to its non-acknowledging readers (**com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 507) = true).

A writer with both acknowledging and non-acknowledging readers keeps a sample queued until acknowledgements have been received from all acknowledging readers and the keep-duration has elapsed for non-acknowledging readers.

[default] false

8.47.2.5 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 false, the keyhash is sent on the wire with every data instance.

When this field is true, the keyhash is not sent on the wire and the readers must compute the value using the received data.

If the *reader* is CPU bound, sending the keyhash on the wire may increase performance, because the reader does not have to get the keyhash from the data.

If the *writer* is CPU bound, sending the keyhash on the wire may decrease performance, because it requires more bandwidth (16 more bytes per sample).

Note: Setting disable_inline_keyhash to true is not compatible with using RTI Real-Time Connect or RTI Recorder.

[default] false

8.47.2.6 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 true if there are DataReaders where **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.propagate_dispose_of_unregistered_instances** (p. 507) is also true.

Important: When this field is true, batching will not be compatible with RTI Connext 4.3e, 4.4b, or 4.4c. The **com.rti.dds.subscription.DataReader** (p. 473) entities will receive incorrect data and/or encounter deserialization errors.

[**default**] false

8.47.2.7 final RtpsReliableWriterProtocol_t rtps_reliable_writer

Initial value:

```
new RtpsReliableWriterProtocol_t()
```

The reliable protocol defined in RTPS.

[**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.AUTO;

samples_per_virtual_heartbeat **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102);

max_heartbeat_retries 10;

inactivate_nonprogressing_readers false;

heartbeats_per_max_samples 8;

min_nack_response_delay 0.0 seconds;

max_nack_response_delay 0.2 seconds;

max_bytes_per_nack_response 131072

8.48 DataWriterProtocolStatus Class Reference

<<*eXtension*>> *(*p. 270*)* 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. 538*) to a matching remote DataReader.*

- long **pushed_sample_count_change**

  The incremental change in the number of user samples pushed on write from a local ***DataWriter*** *(*p. 538*) to a matching remote DataReader 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. 538*) to a matching remote DataReader.*

- long **pushed_sample_bytes_change**

  The incremental change in the number of bytes of user samples pushed on write from a local ***DataWriter*** *(*p. 538*) to a matching remote DataReader since the last time the status was read.*

- long **filtered_sample_count**

  The number of user samples preemptively filtered by a local ***DataWriter*** *(*p. 538*) due to Content-Filtered Topics.*

- long **filtered_sample_count_change**

  The incremental change in the number of user samples preemptively filtered by a local ***DataWriter*** *(*p. 538*) due to Content-Filtered Topics since the last time the status was read.*

- long **filtered_sample_bytes**

  The number of user samples preemptively filtered by a local ***DataWriter*** *(*p. 538*) due to Content-Filtered Topics.*

- long **filtered_sample_bytes_change**

*The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 538) 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. 538) and matching remote DataReader.*

^ long **sent_heartbeat_count_change**

*The incremental change in the number of Heartbeats sent between a local **DataWriter** (p. 538) and matching remote DataReader since the last time the status was read.*

^ long **sent_heartbeat_bytes**

*The number of bytes of Heartbeats sent between a local **DataWriter** (p. 538) and matching remote DataReader.*

^ long **sent_heartbeat_bytes_change**

*The incremental change in the number of bytes of Heartbeats sent between a local **DataWriter** (p. 538) and matching remote DataReader since the last time the status was read.*

^ long **pulled_sample_count**

*The number of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders.*

^ long **pulled_sample_count_change**

*The incremental change in the number of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders since the last time the status was read.*

^ long **pulled_sample_bytes**

*The number of bytes of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders.*

^ long **pulled_sample_bytes_change**

*The incremental change in the number of bytes of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders 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. 538).*

^ long **received_ack_count_change**

*The incremental change in the number of ACKs from a remote DataReader received by a local **DataWriter** (p. 538) 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. 538).*

^ long **received_ack_bytes_change**

*The incremental change in the number of bytes of ACKs from a remote DataReader received by a local **DataWriter** (p. 538) 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. 538).*

^ long **received_nack_count_change**

*The incremental change in the number of NACKs from a remote DataReader received by a local **DataWriter** (p. 538) 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. 538).*

^ long **received_nack_bytes_change**

*The incremental change in the number of bytes of NACKs from a remote DataReader received by a local **DataWriter** (p. 538) since the last time the status was read.*

^ long **sent_gap_count**

*The number of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders.*

^ long **sent_gap_count_change**

*The incremental change in the number of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders since the last time the status was read.*

^ long **sent_gap_bytes**

*The number of bytes of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders.*

^ long **sent_gap_bytes_change**

*The incremental change in the number of bytes of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders since the last time the status was read.*

^ long **rejected_sample_count**

The number of times a sample is rejected due to exceptions in the send path.

^ long **rejected_sample_count_change**

The incremental change in the number of times a sample is rejected due to exceptions in the send path since the last time the status was read.

^ int **send_window_size**

Current maximum number of outstanding samples allowed in the DataWriter's queue.

^ **SequenceNumber_t first_available_sample_sequence_number**

*The sequence number of the first available sample currently queued in the local **DataWriter** (p. 538).*

^ **SequenceNumber_t last_available_sample_sequence_number**

*The sequence number of the last available sample currently queued in the local **DataWriter** (p. 538).*

^ **SequenceNumber_t first_unacknowledged_sample_sequence_number**

*The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 538).*

^ **SequenceNumber_t first_available_sample_virtual_sequence_number**

*The virtual sequence number of the first available sample currently queued in the local **DataWriter** (p. 538).*

^ **SequenceNumber_t last_available_sample_virtual_sequence_number**

*The virtual sequence number of the last available sample currently queued in the local **DataWriter** (p. 538).*

^ **SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number**

*The virtual sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 538).*

^ **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. 538).*

^ **SequenceNumber_t first_unelapsed_keep_duration_sample_sequence_number**

The sequence number of the first sample whose keep duration has not yet elapsed.

8.48.1 Detailed Description

<<***eXtension***>> *(p. 270)* 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. 538)

8.48.2 Member Data Documentation

8.48.2.1 long pushed_sample_count

The number of user samples pushed on write from a local **DataWriter** (p. 538) to a matching remote DataReader.

Counts protocol (RTPS) messages pushed by a **DataWriter** (p. 538) 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.

8.48.2.2 long pushed_sample_count_change

The incremental change in the number of user samples pushed on write from a local **DataWriter** (p. 538) to a matching remote DataReader since the last time the status was read.

Counts protocol (RTPS) messages pushed by a **DataWriter** (p. 538) when writing, unregistering, and disposing.

For large data, counts whole samples, not fragments.

8.48.2.3 long pushed_sample_bytes

The number of bytes of user samples pushed on write from a local **DataWriter** (p. 538) to a matching remote DataReader.

Counts bytes of protocol (RTPS) messages pushed by a **DataWriter** (p. 538) 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.

For large data, counts bytes of whole samples, not fragments.

8.48.2.4 long pushed_sample_bytes_change

The incremental change in the number of bytes of user samples pushed on write from a local **DataWriter** (p. 538) to a matching remote DataReader since the last time the status was read.

Counts bytes of protocol (RTPS) messages pushed by a **DataWriter** (p. 538) when writing, unregistering, and disposing.

For large data, counts bytes of whole samples, not fragments.

8.48.2.5 long filtered_sample_count

The number of user samples preemptively filtered by a local **DataWriter** (p. 538) due to Content-Filtered Topics.

8.48.2.6 long filtered_sample_count_change

The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 538) due to Content-Filtered Topics since the last time the status was read.

8.48.2.7 long filtered_sample_bytes

The number of user samples preemptively filtered by a local **DataWriter** (p. 538) due to Content-Filtered Topics.

8.48.2.8 long filtered_sample_bytes_change

The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 538) due to Content-Filtered Topics since the last time the status was read.

8.48.2.9 long sent_heartbeat_count

The number of Heartbeats sent between a local **DataWriter** (p. 538) 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.

8.48.2.10 long sent_heartbeat_count_change

The incremental change in the number of Heartbeats sent between a local **DataWriter** (p. 538) and matching remote DataReader since the last time the status was read.

8.48.2.11 long sent_heartbeat_bytes

The number of bytes of Heartbeats sent between a local **DataWriter** (p. 538) 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.

8.48.2.12 long sent_heartbeat_bytes_change

The incremental change in the number of bytes of Heartbeats sent between a local **DataWriter** (p. 538) and matching remote DataReader since the last time the status was read.

8.48.2.13 long pulled_sample_count

The number of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 538) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) is false.

For large data, counts whole samples, not fragments.

8.48.2.14 long pulled_sample_count_change

The incremental change in the number of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders 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. 538) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) is false.

For large data, counts whole samples, not fragments.

8.48.2.15 long pulled_sample_bytes

The number of bytes of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 538) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) is false.

For large data, counts bytes of whole samples, not fragments.

8.48.2.16 long pulled_sample_bytes_change

The incremental change in the number of bytes of user samples pulled from local **DataWriter** (p. 538) by matching DataReaders 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. 538) when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) is false.

For large data, counts bytes of whole samples, not fragments.

8.48.2.17 long received_ack_count

The number of ACKs from a remote DataReader received by a local **DataWriter** (p. 538).

8.48.2.18 long received_ack_count_change

The incremental change in the number of ACKs from a remote DataReader received by a local **DataWriter** (p. 538) since the last time the status was read.

8.48.2.19 long received_ack_bytes

The number of bytes of ACKs from a remote DataReader received by a local **DataWriter** (p. 538).

8.48.2.20 long received_ack_bytes_change

The incremental change in the number of bytes of ACKs from a remote DataReader received by a local **DataWriter** (p. 538) since the last time the status was read.

8.48.2.21 long received_nack_count

The number of NACKs from a remote DataReader received by a local **DataWriter** (p. 538).

8.48.2.22 long received_nack_count_change

The incremental change in the number of NACKs from a remote DataReader received by a local **DataWriter** (p. 538) since the last time the status was read.

8.48.2.23 long received_nack_bytes

The number of bytes of NACKs from a remote DataReader received by a local **DataWriter** (p. 538).

8.48.2.24 long received_nack_bytes_change

The incremental change in the number of bytes of NACKs from a remote DataReader received by a local **DataWriter** (p. 538) since the last time the status was read.

8.48.2.25 long sent_gap_count

The number of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders.

8.48.2.26 long sent_gap_count_change

The incremental change in the number of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders since the last time the status was read.

8.48.2.27 long sent_gap_bytes

The number of bytes of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders.

8.48.2.28 long sent_gap_bytes_change

The incremental change in the number of bytes of GAPs sent from local **DataWriter** (p. 538) to matching remote DataReaders since the last time the status was read.

8.48.2.29 long rejected_sample_count

The number of times a sample is rejected due to exceptions in the send path.

8.48.2.30 long rejected_sample_count_change

The incremental change in the number of times a sample is rejected due to exceptions in the send path since the last time the status was read.

8.48.2.31 int send_window_size

Current maximum number of outstanding samples allowed in the DataWriter's queue.

Spans the range from **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_-t.min_send_window_size** (p. 1389) to **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_-t.max_send_window_size** (p. 1390).

8.48.2.32 SequenceNumber_t first_available_sample_sequence_number

The sequence number of the first available sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.33 SequenceNumber_t last_available_sample_sequence_number

The sequence number of the last available sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.34 SequenceNumber_t first_unacknowledged_sample_sequence_number

The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.35 SequenceNumber_t first_available_sample_virtual_sequence_number

The virtual sequence number of the first available sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.36 SequenceNumber_t last_available_sample_virtual_sequence_number

The virtual sequence number of the last available sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.37 SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number

The virtual sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.38 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. 538).

Applies only for local **DataWriter** (p. 538) status.

8.48.2.39 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. 573) is set.

Sequence number of the first sample kept in the DataWriter's queue whose keep_duration (applied when **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.disable_-positive_acks** (p. 573) is set) has not yet elapsed.

Applies only for local **DataWriter** (p. 538) status.

8.49 DataWriterQos Class Reference

QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 538) entity.

Inheritance diagram for DataWriterQos::

Public Attributes

- final **DurabilityQosPolicy durability**
  *Durability policy, **DURABILITY** (p. 65).*
- final **DurabilityServiceQosPolicy durability_service**
  *DurabilityService policy, **DURABILITY_SERVICE** (p. 66).*
- final **DeadlineQosPolicy deadline**
  *Deadline policy, **DEADLINE** (p. 50).*
- final **LatencyBudgetQosPolicy latency_budget**
  *Latency budget policy, **LATENCY_BUDGET** (p. 76).*
- final **LivelinessQosPolicy liveliness**
  *Liveliness policy, **LIVELINESS** (p. 78).*
- final **ReliabilityQosPolicy reliability**
  *Reliability policy, **RELIABILITY** (p. 101).*
- final **DestinationOrderQosPolicy destination_order**
  *Destination order policy, **DESTINATION_ORDER** (p. 51).*
- final **HistoryQosPolicy history**
  *History policy, **HISTORY** (p. 75).*
- final **ResourceLimitsQosPolicy resource_limits**
  *Resource limits policy, **RESOURCE_LIMITS** (p. 102).*
- final **TransportPriorityQosPolicy transport_priority**
  *Transport priority policy, **TRANSPORT_PRIORITY** (p. 121).*
- final **LifespanQosPolicy lifespan**
  *Lifespan policy, **LIFESPAN** (p. 77).*

^ final **UserDataQosPolicy user_data**

User data policy, ***USER_DATA*** *(*p. 126*).*

^ final **OwnershipQosPolicy ownership**

Ownership policy, ***OWNERSHIP*** *(*p. 83*).*

^ final **OwnershipStrengthQosPolicy ownership_strength**

Ownership strength policy, ***OWNERSHIP_STRENGTH*** *(*p. 84*).*

^ final **WriterDataLifecycleQosPolicy writer_data_lifecycle**

Writer data lifecycle policy, ***WRITER_DATA_LIFECYCLE*** *(*p. 134*).*

^ final **DataWriterResourceLimitsQosPolicy writer_resource_limits**

<<**eXtension**>> (p. 270) *Writer resource limits policy,* ***DATA_WRITER_RESOURCE_LIMITS*** *(*p. 49*).*

^ final **DataWriterProtocolQosPolicy protocol**

<<**eXtension**>> (p. 270) ***com.rti.dds.publication.DataWriter*** *(*p. 538*) protocol policy,* ***DATA_WRITER_PROTOCOL*** *(*p. 48*)*

^ final **TransportSelectionQosPolicy transport_selection**

<<**eXtension**>> (p. 270) *Transport plugin selection policy,* ***TRANSPORT_SELECTION*** *(*p. 122*).*

^ final **TransportUnicastQosPolicy unicast**

<<**eXtension**>> (p. 270) *Unicast transport policy,* ***TRANSPORT_UNICAST*** *(*p. 123*).*

^ final **PublishModeQosPolicy publish_mode**

<<**eXtension**>> (p. 270) *Publish mode policy,* ***PUBLISH_MODE*** *(*p. 89*).*

^ final **PropertyQosPolicy property**

<<**eXtension**>> (p. 270) *Property policy,* ***PROPERTY*** *(*p. 88*).*

^ final **BatchQosPolicy batch**

<<**eXtension**>> (p. 270) *Batch policy,* ***BATCH*** *(*p. 42*).*

^ final **MultiChannelQosPolicy multi_channel**

<<**eXtension**>> (p. 270) *Multi channel policy,* ***MULTICHANNEL*** *(*p. 81*).*

- final **EntityNameQosPolicy publication_name** = create_entity_name_policyI()
- final **TypeSupportQosPolicy type_support**

  <<**eXtension**>> (p. 270) *Type support data,* ***TYPESUPPORT*** *(p. 124).*

8.49.1 Detailed Description

QoS policies supported by a **com.rti.dds.publication.DataWriter** (p. 538) entity.

You must set certain members in a consistent manner:

- com.rti.dds.publication.DataWriterQos.history.depth <= com.rti.dds.publication.DataWriterQos.resource_limits.max_samples_per_instance

- com.rti.dds.publication.DataWriterQos.resource_limits.max_samples_per_instance <= com.rti.dds.publication.DataWriterQos.resource_limits.max_samples

- com.rti.dds.publication.DataWriterQos.resource_limits.initial_samples <= com.rti.dds.publication.DataWriterQos.resource_limits.max_samples

- com.rti.dds.publication.DataWriterQos.resource_limits.initial_instances <= com.rti.dds.publication.DataWriterQos.resource_limits.max_instances

- length of com.rti.dds.publication.DataWriterQos.user_data.value <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .writer_user_data_max_length

If any of the above are not true, **com.rti.dds.publication.DataWriter.set_qos** (p. 543) and **com.rti.dds.publication.DataWriter.set_qos_with_profile** (p. 544) and **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282) and **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1283) will fail with RETCODE_INCONSISTENT_POLICY and **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284) and **com.rti.dds.publication.Publisher.create_datawriter_with_profile** (p. 1286) and will return NULL.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

See also:

QoS Policies (p. 90) allowed ranges within each Qos.

8.49.2 Member Data Documentation

8.49.2.1 final DurabilityQosPolicy durability

Durability policy, **DURABILITY** (p. 65).

8.49.2.2 final DurabilityServiceQosPolicy durability_service

DurabilityService policy, **DURABILITY_SERVICE** (p. 66).

8.49.2.3 final DeadlineQosPolicy deadline

Deadline policy, **DEADLINE** (p. 50).

8.49.2.4 final LatencyBudgetQosPolicy latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 76).

8.49.2.5 final LivelinessQosPolicy liveliness

Liveliness policy, **LIVELINESS** (p. 78).

8.49.2.6 final ReliabilityQosPolicy reliability

Reliability policy, **RELIABILITY** (p. 101).

8.49.2.7 final DestinationOrderQosPolicy destination_order

Destination order policy, **DESTINATION_ORDER** (p. 51).

8.49.2.8 final HistoryQosPolicy history

History policy, **HISTORY** (p. 75).

8.49.2.9 final ResourceLimitsQosPolicy resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 102).

8.49.2.10 final TransportPriorityQosPolicy transport_priority

Transport priority policy, **TRANSPORT_PRIORITY** (p. 121).

8.49.2.11 final LifespanQosPolicy lifespan

Lifespan policy, **LIFESPAN** (p. 77).

8.49.2.12 final UserDataQosPolicy user_data

User data policy, **USER_DATA** (p. 126).

8.49.2.13 final OwnershipQosPolicy ownership

Ownership policy, **OWNERSHIP** (p. 83).

8.49.2.14 final OwnershipStrengthQosPolicy ownership_strength

Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 84).

8.49.2.15 final WriterDataLifecycleQosPolicy writer_data_lifecycle

Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 134).

8.49.2.16 final DataWriterResourceLimitsQosPolicy writer_resource_limits

<<*eXtension*>> *(p. 270)* Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 49).

8.49.2.17 final DataWriterProtocolQosPolicy protocol

<<*eXtension*>> *(p. 270)* **com.rti.dds.publication.DataWriter** (p. 538) protocol policy, **DATA_WRITER_PROTOCOL** (p. 48)

8.49.2.18 final TransportSelectionQosPolicy transport_selection

<<*eXtension*>> *(p. 270)* Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 122).

Specifies the transports available for use by the **com.rti.dds.publication.DataWriter** (p. 538).

8.49.2.19 final TransportUnicastQosPolicy unicast

<<eXtension>> *(p. 270)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 123).

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. 473) entities in the **domain** (p. 317).

8.49.2.20 final PublishModeQosPolicy publish_mode

<<eXtension>> *(p. 270)* Publish mode policy, **PUBLISH_MODE** (p. 89).

Determines whether the **com.rti.dds.publication.DataWriter** (p. 538) publishes data synchronously or asynchronously and how.

8.49.2.21 final PropertyQosPolicy property

<<eXtension>> *(p. 270)* Property policy, **PROPERTY** (p. 88).

8.49.2.22 final BatchQosPolicy batch

<<eXtension>> *(p. 270)* Batch policy, **BATCH** (p. 42).

8.49.2.23 final MultiChannelQosPolicy multi_channel

<<eXtension>> *(p. 270)* Multi channel policy, **MULTICHANNEL** (p. 81).

8.49.2.24 final EntityNameQosPolicy publication_name = create_entity_name_policyI()

8.49.2.25 final TypeSupportQosPolicy type_support

<<eXtension>> *(p. 270)* Type support data, **TYPESUPPORT** (p. 124).

Optional value that is passed to a type plugin's on_endpoint_attached and serialization functions.

8.50 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. 538) *to reclaim unregistered acknowledged instances.*

- static final **DataWriterResourceLimitsInstanceReplacementKind ALIVE_INSTANCE_REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. 538) *to reclaim alive acknowledged instances.*

- static final **DataWriterResourceLimitsInstanceReplacementKind DISPOSED_INSTANCE_REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. 538) *to reclaim disposed acknowledged instances.*

- static final **DataWriterResourceLimitsInstanceReplacementKind ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT**

  *Allows a **com.rti.dds.publication.DataWriter*** (p. 538) *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. 538) *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. 538) *to reclaim a either an alive acknowledged instance or a disposed acknowledged instance.*

8.50.1 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. 1360) is reached, a **com.rti.dds.publication.DataWriter** (p. 538) 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.DataWriterResourceLimitsQosPolicy.instance_replacement.

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. 538) will replace the oldest instance satisfying that kind. For example, when the kind is **DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED_INSTANCE_REPLACEMENT** (p. 596), a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) will replace the old instance with the new instance, and invoke, if available, the com.rti.dds.publication.DataWriterListener_InstanceReplacedCallback to notify the user about an instance being replaced.

A **com.rti.dds.publication.DataWriter** (p. 538) 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. 601) is true, a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) tries replacing unregistered instances. Since an unregistered instance indicates that the **com.rti.dds.publication.DataWriter** (p. 538) is done modifying it, unregistered instances are replaced before instances of any other state (alive, disposed). This is the same as the **DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED_INSTANCE_REPLACEMENT** (p. 596) kind. Then, a **com.rti.dds.publication.DataWriter** (p. 538) tries replacing what is specified by com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_replacement. 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. 598)

8.50.2 Member Data Documentation

8.50.2.1 final DataWriterResourceLimitsInstanceReplacementKind UNREGISTERED_INSTANCE_REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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 [**default**]

8.50.2.2 final DataWriterResourceLimitsInstanceReplacementKind ALIVE_INSTANCE_REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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.

8.50.2.3 final DataWriterResourceLimitsInstanceReplacementKind DISPOSED_INSTANCE_REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) to reclaim a disposed acknowledged instance. The least recently disposed instance will be reclaimed.

8.50.2.4 final DataWriterResourceLimitsInstanceReplacementKind ALIVE_THEN_DISPOSED_INSTANCE_-REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) first try reclaiming an alive acknowledged instance. If no instance is reclaimable, then it trys reclaiming a disposed acknowledged instance. The least recently used (i.e. registered, written, or disposed) instance will be reclaimed.

8.50.2.5 final DataWriterResourceLimitsInstanceReplacementKind DISPOSED_THEN_ALIVE_INSTANCE_-REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) first try reclaiming a disposed acknowledged instance. If no instance is reclaimable, then it trys reclaiming an alive acknowledged instance. The least recently used (i.e. disposed, registered, or written) instance will be reclaimed.

8.50.2.6 final DataWriterResourceLimitsInstanceReplacementKind ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT [static]

Allows a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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. 538) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

8.51 DataWriterResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538).*

- 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. 538).*

- int **max_remote_reader_filters**

  The maximum number of remote readers for which the writer will perform content-based filtering.

- int **max_batches**

  *Represents the maximum number of batches a **com.rti.dds.publication.DataWriter** (p. 538) will manage.*

- int **initial_batches**

  *Represents the initial number of batches a **com.rti.dds.publication.DataWriter** (p. 538) will manage.*

- int **cookie_max_length**

  *Represents the maximum length in bytes of a **com.rti.dds.infrastructure.Cookie_t** (p. 465).*

- 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. 538).*

^ int **max_virtual_writers**

*The maximum number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 538).*

8.51.1 Detailed Description

Various settings that configure how a **com.rti.dds.publication.DataWriter** (p. 538) 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.dds.topic.example.FooDataWriter.write on the same **com.rti.dds.publication.DataWriter** (p. 538), 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. 538). 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. 538).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.51.2 Member Data Documentation

8.51.2.1 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. 538).

This value only applies if **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) has its kind set to **HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076) and **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1339) is > 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

8.51.2.2 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. 538).

This value only applies if **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) has its kind set to **HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076) and **com.rti.dds.infrastructure.ReliabilityQosPolicy.max_blocking_time** (p. 1339) is > 0.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 10000] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_concurrent_blocking_threads

8.51.2.3 int max_remote_reader_filters

The maximum number of remote readers for which the writer will perform content-based filtering.

[default] 32

[range] [0, 32]

8.51.2.4 int max_batches

Represents the maximum number of batches a **com.rti.dds.publication.DataWriter** (p. 538) will manage.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

When batching is enabled, the maximum number of samples that a **com.rti.dds.publication.DataWriter** (p. 538) can store is limited by this value and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359).

[range] [1,100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) >= DDS_RtpsReliableWriterProtocol_t.heartbeats_

per_max_samples if batching is enabled

See also:

com.rti.dds.infrastructure.BatchQosPolicy (p. 401)

8.51.2.5 int initial_batches

Represents the initial number of batches a **com.rti.dds.publication.DataWriter** (p. 538) will manage.

[default] 8

[range] [1,100 million]

See also:

com.rti.dds.infrastructure.BatchQosPolicy (p. 401)

8.51.2.6 int cookie_max_length

Represents the maximum length in bytes of a **com.rti.dds.infrastructure.Cookie_t** (p. 465).

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

Sets the maximum allowed byte-sequence length of a **com.rti.dds.infrastructure.Cookie_t** (p. 465) used when writing with parameters

[range] [1,100 million] or **ResourceLimitsQosPolicy.LENGTH_-UNLIMITED** (p. 102)

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_params,
com.rti.dds.topic.example.FooDataWriter.dispose_w_params,
com.rti.dds.topic.example.FooDataWriter.register_instance_w_params,
com.rti.dds.topic.example.FooDataWriter.unregister_instance_w_params

8.51.2.7 boolean replace_empty_instances

Whether or not to replace empty instances during instance replacement.

When a **com.rti.dds.publication.DataWriter** (p. 538) has more active instances than allowed by

com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_-instances (p. 1360), it tries to make room by replacing an existing instance. This field configures whether empty instances (i.e. instances with no samples) may be replaced. If set true, then a **com.rti.dds.publication.DataWriter** (p. 538) will first try reclaiming empty instances, before trying to replace whatever is specified by com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.instance_-replacement.

[default] false

See also:

com.rti.dds.infrastructure.DataWriterResourceLimitsInstanceReplacementKind (p. 594)

8.51.2.8 boolean autoregister_instances

Whether or not to automatically register new instances.

[default] true

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.dds.topic.example.FooDataWriter.write

8.51.2.9 int initial_virtual_writers

The initial number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 538).

[default] 1

[range] [1, 1000000], or **ResourceLimitsQosPolicy.LENGTH_-UNLIMITED** (p. 102)

8.51.2.10 int max_virtual_writers

The maximum number of virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 538).

Sets the maximum number of unique virtual writers supported by a **com.rti.dds.publication.DataWriter** (p. 538), 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**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[**range**] [1, 1000000], or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.52 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.

8.52.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **com.rti.dds.subscription.DataReader** (p. 473) 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. 538) indicates that the application commits to write a new value (using the **com.rti.dds.publication.DataWriter** (p. 538)) for each instance managed by the **com.rti.dds.publication.DataWriter** (p. 538) 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 run time 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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_DEADLINE_MISSED_STATUS (p. 1458), **StatusKind.REQUESTED_DEADLINE_MISSED_STATUS**

(p. 1458), **StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **YES** (p. 98)

8.52.2 Usage

This policy is useful for cases where a **com.rti.dds.topic.Topic** (p. 1545) 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. 538) and a **com.rti.dds.subscription.DataReader** (p. 473) 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. 1154) or **com.rti.dds.infrastructure.Condition** (p. 451) 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. 1154) or **com.rti.dds.infrastructure.Condition** (p. 451).

8.52.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period <= requested period* holds.

8.52.4 Consistency

The setting of the **DEADLINE** (p. 50) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 113).

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 **RETCODE_INCONSISTENT_POLICY** (p. 1367) in **set_qos (abstract)**

(p. 913), or the **com.rti.dds.infrastructure.Entity** (p. 912) will not be created.

For a **com.rti.dds.subscription.DataReader** (p. 473), the **DEADLINE** (p. 50) policy and **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy** (p. 1541) may interact such that even though the **com.rti.dds.publication.DataWriter** (p. 538) is writing samples fast enough to fulfill its commitment to its own deadline, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1544). To avoid triggering the **com.rti.dds.subscription.DataReader** (p. 473)'s deadline, even though the matched **com.rti.dds.publication.DataWriter** (p. 538) 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. 1541) for more information about the interactions between deadlines and time-based filters.

See also:

com.rti.dds.infrastructure.TimeBasedFilterQosPolicy (p. 1541)

8.52.5 Member Data Documentation

8.52.5.1 final Duration_t period

Duration of the deadline period.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE

[range] [1 nanosec, 1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE, >= **com.rti.dds.infrastructure.TimeBasedFilterQosPolicy.minimum_separation** (p. 1544)

8.53 DestinationOrderQosPolicy Class Reference

Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 538) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1545) and key).

Inheritance diagram for DestinationOrderQosPolicy::

Public Attributes

- ^ **DestinationOrderQosPolicyKind kind**

  Specifies the desired kind of destination order.

- ^ final **Duration_t source_timestamp_tolerance**

  <<**eXtension**>> (p. 270) *Allowed tolerance between source timestamps of consecutive samples.*

8.53.1 Detailed Description

Controls how the middleware will deal with data sent by multiple **com.rti.dds.publication.DataWriter** (p. 538) entities for the same instance of data (i.e., same **com.rti.dds.topic.Topic** (p. 1545) and key).

Entity:

com.rti.dds.topic.Topic (p. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.53.2 Usage

When multiple DataWriters send data for the same **topic** (p. 350), 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. 538) entities (which may be associated with different **com.rti.dds.publication.Publisher** (p. 1277) entities) running on different nodes.

The default setting, **DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 610), indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 84) 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. 473) in the order in which it was *received* (which may lead to inconsistent final values).

The setting **DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 611) indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 84) allows it, within each instance) the `source_timestamp` of the change shall be used to determine the most recent information. That is, data will be delivered by a **com.rti.dds.subscription.DataReader** (p. 473) in the order in which it was *sent*. If data arrives on the network with a source timestamp that is later than the source timestamp of the last data delivered, the new data will be dropped. This 'by source timestamp' ordering therefore works best when system clocks are relatively synchronized among writing machines.

When using **DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 611), not all data sent by multiple **com.rti.dds.publication.DataWriter** (p. 538) entities may be delivered to a **com.rti.dds.subscription.DataReader** (p. 473) and not all DataReaders will see the same data sent by DataWriters. However, all DataReaders will see the same "final" data when DataWriters "stop" sending data. This is the only setting that, in the case of concurrently publishing **com.rti.dds.publication.DataWriter** (p. 538) entities updating the same instance of a shared-ownership **topic** (p. 350), ensures all subscribers will end up with the same final value for the instance.

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** (p. 350), all applications will end up having the same state.

8.53.3 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. 609) are considered ordered such that **DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 610) < **DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 611)

8.53.4 Member Data Documentation

8.53.4.1 DestinationOrderQosPolicyKind kind

Specifies the desired kind of destination order.

[**default**] **DestinationOrderQosPolicyKind.BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 610),

8.53.4.2 final Duration_t source_timestamp_tolerance

<<***eXtension***>> *(p. 270)* Allowed tolerance between source timestamps of consecutive samples.

When a **com.rti.dds.publication.DataWriter** (p. 538) sets **com.rti.dds.infrastructure.DestinationOrderQosPolicyKind** (p. 610) to **DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 611), when 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. 473) sets **com.rti.dds.infrastructure.DestinationOrderQosPolicyKind** (p. 610) to **DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 611), the **com.rti.dds.subscription.DataReader** (p. 473) will accept a sample only if the difference between its source timestamp and the reception timestamp is no greater than this tolerance. Otherwise, the sample is rejected.

[**default**] 100 milliseconds for **com.rti.dds.publication.DataWriter** (p. 538), 30 seconds for **com.rti.dds.subscription.DataReader** (p. 473)

8.54 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. 1478).

- 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).

8.54.1 Detailed Description

Kinds of destination order.

QoS:

com.rti.dds.infrastructure.DestinationOrderQosPolicy (p. 607)

8.54.2 Member Data Documentation

8.54.2.1 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. 1478).

Since each subscriber may receive the data at different times there is no guaranteed 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.

8.54.2.2 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.

See also:

Special Instructions if Using 'Timestamp' APIs and BY_SOURCE_TIMESTAMP Destination Ordering: (p. 1288)

8.55 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

8.55.1 Detailed Description

DiscoveryBuiltinReaderFragmentationResourceLimits_t (p. 612)

8.55.2 Constructor & Destructor Documentation

8.55.2.1 DiscoveryBuiltinReaderFragmentationResourceLimits_t ()

DiscoveryBuiltinReaderFragmentationResourceLimits_t_new_with_no_parameter

8.55.2.2 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

8.55.3 Member Data Documentation

8.55.3.1 boolean disable_fragmentation_support = false

DiscoveryBuiltinReaderFragmentationResourceLimits_t_disable_fragmentation_support

8.55.3.2 int max_fragmented_samples = 1024

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragmented_samples

8.55.3.3 int initial_fragmented_samples = 4

DiscoveryBuiltinReaderFragmentationResourceLimits_t_initial_fragmented_samples

8.55.3.4 int max_fragmented_samples_per_remote_writer = 256

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragmented_samples_per_remote_writer

8.55.3.5 int max_fragments_per_sample = 512

DiscoveryBuiltinReaderFragmentationResourceLimits_t_max_fragments_per_sample

8.55.3.6 boolean dynamically_allocate_fragmented_samples = false

DiscoveryBuiltinReaderFragmentationResourceLimits_t_dynamically_allocate_fragmented_samples

8.56 DiscoveryConfigBuiltinPluginKind Class Reference

Built-in discovery plugins that can be used.

Static Public Attributes

- static final int **SDP**

  Built-in discovery plugins that can be used.

- static final int **MASK_NONE** = 0

  A bit-mask (list) of built-in discovery plugins.

- static final int **MASK_ALL** = 0xefff

  A bit-mask (list) of built-in discovery plugins.

- static final int **MASK_DEFAULT** = **SDP**

8.56.1 Detailed Description

Built-in discovery plugins that can be used.

See also:

com.rti.dds.infrastructure.DiscoveryConfigBuiltinPluginKindMask

8.56.2 Member Data Documentation

8.56.2.1 final int MASK_DEFAULT = SDP [static]

DiscoveryConfigBuiltinPluginKindMask.MASK_DEFAULT

8.57 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.

- **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 or 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 a built-in **publication** (p. 338) writer.*

^ final **WriterDataLifecycleQosPolicy publication_writer_data_lifecycle**

*Writer data lifecycle settings for a built-in **publication** (p. 338) writer.*

^ final **RtpsReliableWriterProtocol_t subscription_writer**

*RTPS protocol-related configuration settings for a built-in **subscription** (p. 343) writer.*

^ final **WriterDataLifecycleQosPolicy subscription_writer_data_lifecycle**

*Writer data lifecycle settings for a built-in **subscription** (p. 343) writer.*

^ final **RtpsReliableReaderProtocol_t publication_reader**

*RTPS protocol-related configuration settings for a built-in **publication** (p. 338) reader.*

^ final **RtpsReliableReaderProtocol_t subscription_reader**

*RTPS protocol-related configuration settings for a built-in **subscription** (p. 343) reader.*

^ int **builtin_discovery_plugins**

The kind mask for built-in discovery plugins.

^ final **RtpsReliableReaderProtocol_t participant_message_reader**

RTPS protocol-related configuration settings for a built-in participant message reader.

^ final **RtpsReliableWriterProtocol_t participant_message_writer**

RTPS protocol-related configuration settings for a built-in participant message writer.

8.57.1 Detailed Description

Settings for discovery configuration.

This QoS policy is an extension to the DDS standard.

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. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.57.2 Member Data Documentation

8.57.2.1 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. 618).

[**default**] 100 seconds

[**range**] [1 nanosec,1 year], > participant_liveliness_assert_period

8.57.2.2 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. 617).

[default] 30 seconds

[range] [1 nanosec,1 year), < participant_liveliness_lease_duration

8.57.2.3 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.RemoteParticipantPurgeKind.LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE** (p. 1351). However, **com.rti.dds.infrastructure.RemoteParticipantPurgeKind.NO_REMOTE_PARTICIPANT_PURGE** (p. 1351) 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. 1168).

[default] **com.rti.dds.infrastructure.RemoteParticipantPurgeKind.LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE** (p. 1351)

8.57.2.4 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 that 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. 473) will learn of the loss of liveliness of a matched **com.rti.dds.publication.DataWriter** (p. 538) 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**] [0, 1 year]

8.57.2.5 int initial_participant_announcements

The number of initial announcements sent when a participant is first enabled or when a remote participant is newly discovered.

Also, when a new remote participant appears, the local participant can announce itself to the peers multiple times controlled by this parameter.

[**default**] 5

[**range**] [0,1 million]

8.57.2.6 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. 620) 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. 619) must be consistent with **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 620). For these two values to be consistent, they must verify that:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period (p. 619) <= **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 620).

[**default**] 1 second

[range] [1 nanosec,1 year]

8.57.2.7 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. 619) 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. 620) must be consistent with **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period** (p. 619). For these two values to be consistent, they must verify that:

com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.min_initial_participant_announcement_period (p. 619) <= **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.max_initial_participant_announcement_period** (p. 620).

[default] 1 second

[range] [1 nanosec,1 year]

8.57.2.8 final BuiltinTopicReaderResourceLimits_t participant_reader_resource_limits

Resource limits.

Resource limit of the built-in **topic** (p. 350) participant reader. For details, see **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t** (p. 414).

8.57.2.9 final BuiltinTopicReaderResourceLimits_t publication_reader_resource_limits

Resource limits.

Resource limit of the built-in **topic** (p. 350) **publication** (p. 338) reader. For details, see **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t** (p. 414).

8.57.2.10 final BuiltinTopicReaderResourceLimits_t subscription_reader_resource_limits

Resource limits.

Resource limit of the built-in **topic** (p. 350) **subscription** (p. 343) reader. For details, see **com.rti.dds.infrastructure.BuiltinTopicReaderResourceLimits_t** (p. 414).

8.57.2.11 final RtpsReliableWriterProtocol_t publication_writer

RTPS protocol-related configuration settings for a built-in **publication** (p. 338) writer.

For details, refer to the **com.rti.dds.publication.DataWriterQos** (p. 588)

8.57.2.12 final WriterDataLifecycleQosPolicy publication_writer_data_lifecycle

Writer data lifecycle settings for a built-in **publication** (p. 338) writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 1722). **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 1723) will always be forced to true.

8.57.2.13 final RtpsReliableWriterProtocol_t subscription_writer

RTPS protocol-related configuration settings for a built-in **subscription** (p. 343) writer.

For details, refer to the **com.rti.dds.publication.DataWriterQos** (p. 588)

8.57.2.14 final WriterDataLifecycleQosPolicy subscription_writer_data_lifecycle

Writer data lifecycle settings for a built-in **subscription** (p. 343) writer.

For details, refer to the **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy** (p. 1722). **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 1723) will always be forced to true.

8.57.2.15 final RtpsReliableReaderProtocol_t publication_reader

RTPS protocol-related configuration settings for a built-in **publication** (p. 338) reader.

For details, refer to the **com.rti.dds.subscription.DataReaderQos** (p. 518)

8.57.2.16 final RtpsReliableReaderProtocol_t subscription_reader

RTPS protocol-related configuration settings for a built-in **subscription** (p. 343) reader.

For details, refer to the **com.rti.dds.subscription.DataReaderQos** (p. 518)

8.57.2.17 int builtin_discovery_plugins

Initial value:

```
DiscoveryConfigBuiltinPluginKind.MASK_DEFAULT
```

The kind mask for built-in discovery plugins.

There are several built-in discovery plugin. This mask enables the different plugins. Any plugin not enabled will not be created.

[default] DiscoveryConfigBuiltinPluginKind.SDP (p. 53)

8.57.2.18 final RtpsReliableReaderProtocol_t participant_message_reader

RTPS protocol-related configuration settings for a built-in participant message reader.

For details, refer to the **com.rti.dds.subscription.DataReaderQos** (p. 518)

8.57.2.19 final RtpsReliableWriterProtocol_t participant_message_writer

RTPS protocol-related configuration settings for a built-in participant message writer.

For details, refer to the **com.rti.dds.publication.DataWriterQos** (p. 588)

8.58 DiscoveryPluginPromiscuityKind Class Reference

<<***eXtension***>> *(p. 270)* Type used to indicate promiscuity mode of the discovery plugin.

Inheritance diagram for DiscoveryPluginPromiscuityKind::

8.58.1 Detailed Description

<<***eXtension***>> *(p. 270)* Type used to indicate promiscuity mode of the discovery plugin.

8.59 DiscoveryQosPolicy Class Reference

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 DomainParticipant.*

- 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.

8.59.1 Detailed Description

Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.59.2 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. 55)
com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615)

8.59.3 Member Data Documentation

8.59.3.1 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. 473) and **com.rti.dds.publication.DataWriter** (p. 538)).

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. 115).

[**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.

8.59.3.2 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 DomainParticipant.

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. 57)).

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. 625) 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. 625), 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. 625).

[default] See **NDDS_DISCOVERY_PEERS** (p. 55)

[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. 57)

8.59.3.3 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. 538) using this transport priority.

[default] 0

8.59.3.4 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.

If there is a remote peer **com.rti.dds.domain.DomainParticipant** (p. 629) such as is described in this list, it will become aware of this participant and will engage in the Discovery protocol to exchange meta-data with this participant.

Each element of this list must be a `peer` descriptor in the proper format (see **Peer Descriptor Format** (p. 56)).

[**default**] See **NDDS_DISCOVERY_PEERS** (p. 55)

[**range**] Sequence of arbitrary length.

See also:

Peer Descriptor Format (p. 56)
com.rti.dds.domain.DomainParticipant.add_peer() (p. 692)

8.59.3.5 boolean accept_unknown_peers

Whether to accept a new participant that is not in the initial peers list.

If false, the participant will only communicate with those in the initial peers list and those added via **com.rti.dds.domain.DomainParticipant.add_peer()** (p. 692).

If true, the participant will also communicate with all discovered remote participants.

Note: If accept_unknown_peers is false and shared memory is disabled, applications on the same node will *not* communicate if only 'localhost' is specified in the peers list. If shared memory is disabled or 'shmem://' is not specified in the peers list, to communicate with other applications on the same node through the loopback interface, you must put the actual node address or hostname in **NDDS_DISCOVERY_PEERS** (p. 55).

[**default**] true

8.60 DomainEntity Interface Reference

<<*interface*>> *(p. 271)* Abstract base class for all DDS entities except for the **com.rti.dds.domain.DomainParticipant** (p. 629).

Inheritance diagram for DomainEntity::

8.60.1 Detailed Description

<<*interface*>> *(p. 271)* Abstract base class for all DDS entities except for the **com.rti.dds.domain.DomainParticipant** (p. 629).

Its sole purpose is to *conceptually* express that **com.rti.dds.domain.DomainParticipant** (p. 629) is a special kind of **com.rti.dds.infrastructure.Entity** (p. 912) that acts as a container of all other **com.rti.dds.infrastructure.Entity** (p. 912) but itself cannot contain other **com.rti.dds.domain.DomainParticipant** (p. 629).

8.61 DomainParticipant Interface Reference

<<*interface*>> (p. 271) Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 628) objects.

Inheritance diagram for DomainParticipant::

Public Member Functions

- void **get_default_flowcontroller_property** (**FlowControllerProperty_t** prop)

  <<**eXtension**>> (p. 270) *Copies the default* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 946) values for this* ***domain*** *(p. 317) participant into the given* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 946) instance.*

- void **set_default_flowcontroller_property** (**FlowControllerProperty_t** prop)

  <<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.publication.FlowControllerProperty_t*** *(p. 946) values for this* ***domain*** *(p. 317) participant.*

- void **get_default_topic_qos** (**TopicQos** qos)

  Copies the default ***com.rti.dds.topic.TopicQos*** *(p. 1566) values for this* ***domain*** *(p. 317) participant into the given* ***com.rti.dds.topic.TopicQos*** *(p. 1566) instance.*

- void **set_default_topic_qos** (**TopicQos** qos)

  Set the default ***com.rti.dds.topic.TopicQos*** *(p. 1566) values for this* ***domain*** *(p. 317) participant.*

- void **set_default_topic_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.topic.TopicQos*** *(p. 1566) values for this* ***domain*** *(p. 317) participant based on the input XML QoS profile.*

- void **get_default_publisher_qos** (**PublisherQos** qos)

  Copy the default ***com.rti.dds.publication.PublisherQos*** *(p. 1303) values into the provided* ***com.rti.dds.publication.PublisherQos*** *(p. 1303) instance.*

- void **set_default_publisher_qos** (**PublisherQos** qos)

Set the default ***com.rti.dds.publication.PublisherQos*** *(p. 1303) values for this* ***DomainParticipant*** *(p. 629).*

- void **set_default_publisher_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.publication.PublisherQos*** *(p. 1303) values for this* ***DomainParticipant*** *(p. 629) based on the input XML QoS profile.*

- void **get_default_datawriter_qos** (**DataWriterQos** qos)

  <<**eXtension**>> (p. 270) *Copy the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) values into the provided* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) instance.*

- void **set_default_datawriter_qos** (**DataWriterQos** qos)

  <<**eXtension**>> (p. 270) *Set the default DataWriterQos values for this* ***DomainParticipant*** *(p. 629).*

- void **set_default_datawriter_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) values for this* ***domain*** *(p. 317) participant based on the input XML QoS profile.*

- void **get_default_datareader_qos** (**DataReaderQos** qos)

  <<**eXtension**>> (p. 270) *Copy the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) values into the provided* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) instance.*

- void **set_default_subscriber_qos** (**SubscriberQos** qos)

  Set the default ***com.rti.dds.subscription.SubscriberQos*** *(p. 1506) values for this Domain{articipant.*

- void **set_default_subscriber_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.subscription.SubscriberQos*** *(p. 1506) values for this* ***DomainParticipant*** *(p. 629) based on the input XML QoS profile.*

- void **get_default_subscriber_qos** (**SubscriberQos** qos)

  Copy the default ***com.rti.dds.subscription.SubscriberQos*** *(p. 1506) values into the provided* ***com.rti.dds.subscription.SubscriberQos*** *(p. 1506) instance.*

- void **set_default_datareader_qos** (**DataReaderQos** qos)

<<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) values for this* ***domain*** *(p. 317) participant.*

ˆ void **set_default_datareader_qos_with_profile** (String library_name, String profile_name)

<<**eXtension**>> (p. 270) *Set the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) values for this* ***DomainParticipant*** *(p. 629) based on the input XML QoS profile.*

ˆ **FlowController create_flowcontroller** (String name, **FlowControllerProperty_t** prop)

<<**eXtension**>> (p. 270) *Creates a* ***com.rti.dds.publication.FlowController*** *(p. 942) with the desired property.*

ˆ void **delete_flowcontroller** (**FlowController** fc)

<<**eXtension**>> (p. 270) *Deletes an existing* ***com.rti.dds.publication.FlowController*** *(p. 942).*

ˆ **Publisher create_publisher** (**PublisherQos** qos, **PublisherListener** listener, int mask)

Creates a ***com.rti.dds.publication.Publisher*** *(p. 1277) with the desired QoS policies and attaches to it the specified* ***com.rti.dds.publication.PublisherListener*** *(p. 1302).*

ˆ **Publisher create_publisher_with_profile** (String library_name, String profile_name, **PublisherListener** listener, int mask)

<<**eXtension**>> (p. 270) *Creates a new* ***com.rti.dds.publication.Publisher*** *(p. 1277) object using the* ***com.rti.dds.publication.PublisherQos*** *(p. 1303) associated with the input XML QoS profile.*

ˆ void **delete_publisher** (**Publisher** p)

Deletes an existing ***com.rti.dds.publication.Publisher*** *(p. 1277).*

ˆ **Subscriber create_subscriber** (**SubscriberQos** qos, **SubscriberListener** listener, int mask)

Creates a ***com.rti.dds.subscription.Subscriber*** *(p. 1478) with the desired QoS policies and attaches to it the specified* ***com.rti.dds.subscription.SubscriberListener*** *(p. 1504).*

ˆ **Subscriber create_subscriber_with_profile** (String library_name, String profile_name, **SubscriberListener** listener, int mask)

<<**eXtension**>> (p. 270) *Creates a new* ***com.rti.dds.subscription.Subscriber*** *(p. 1478) object using the*

com.rti.dds.publication.PublisherQos *(p. 1303) associated with the input XML QoS profile.*

- void **delete_subscriber** (**Subscriber** s)

  Deletes an existing ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- **DataWriter create_datawriter** (**Topic** topic, **DataWriterQos** qos, **DataWriterListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a* ***com.rti.dds.publication.DataWriter*** *(p. 538) that will be attached and belong to the implicit* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- **DataWriter create_datawriter_with_profile** (**Topic** topic, String library_name, String profile_name, **DataWriterListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a* ***com.rti.dds.publication.DataWriter*** *(p. 538) using a XML QoS profile that will be attached and belong to the implicit* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- void **delete_datawriter** (**DataWriter** a_datawriter)

  <<**eXtension**>> (p. 270) *Deletes a* ***com.rti.dds.publication.DataWriter*** *(p. 538) that belongs to the implicit* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- **DataReader create_datareader** (**TopicDescription** topic, **DataReaderQos** qos, **DataReaderListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a* ***com.rti.dds.subscription.DataReader*** *(p. 473) that will be attached and belong to the implicit* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- **DataReader create_datareader_with_profile** (**TopicDescription** topic, String library_name, String profile_name, **DataReaderListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a* ***com.rti.dds.subscription.DataReader*** *(p. 473) using a XML QoS profile that will be attached and belong to the implicit* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- void **delete_datareader** (**DataReader** a_datareader)

  <<**eXtension**>> (p. 270) *Deletes a* ***com.rti.dds.subscription.DataReader*** *(p. 473) that belongs to the implicit* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- **Topic create_topic** (String topic_name, String type_name, **TopicQos** qos, **TopicListener** listener, int mask)

*Creates a **com.rti.dds.topic.Topic** (p. 1545) with the desired QoS policies and attaches to it the specified **com.rti.dds.topic.TopicListener** (p. 1564).*

^ **Topic create_topic_with_profile** (String topic_name, String type_name, String library_name, String profile_name, **TopicListener** listener, int mask)

<<**eXtension**>> (p. 270) *Creates a new **com.rti.dds.topic.Topic** (p. 1545) object using the **com.rti.dds.publication.PublisherQos** (p. 1303) associated with the input XML QoS profile.*

^ void **delete_topic** (**Topic** topic)

*Deletes a **com.rti.dds.topic.Topic** (p. 1545).*

^ **ContentFilteredTopic create_contentfilteredtopic** (String name, **Topic** related_topic, String filter_expression, **StringSeq** expression_parameters)

*Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 458), 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. 270) *Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) using the specified filter to do content-based subscriptions.*

^ void **delete_contentfilteredtopic** (**ContentFilteredTopic** a_contentfilteredtopic)

*Deletes a **com.rti.dds.topic.ContentFilteredTopic** (p. 458).*

^ **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. 1208).*

^ void **set_qos** (**DomainParticipantQos** qos)

*Change the QoS of this **DomainParticipant** (p. 629).*

^ void **set_qos_with_profile** (String library_name, String profile_name)

<<**eXtension**>> (p. 270) *Change the QoS of this **domain** (p. 317) participant using the input XML QoS profile.*

- void **get_qos** (**DomainParticipantQos** qos)

  Get the participant QoS.

- String **get_default_library** ()

  <<**eXtension**>> (p. 270) *Gets the default XML library associated with a* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- void **set_default_library** (String library_name)

  <<**eXtension**>> (p. 270) *Sets the default XML library for a* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- String **get_default_profile** ()

  <<**eXtension**>> (p. 270) *Gets the default XML profile associated with a* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- void **set_default_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Sets the default XML profile for a* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- String **get_default_profile_library** ()

  <<**eXtension**>> (p. 270) *Gets the library where the default XML QoS profile is contained for a* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- 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. 270) *Allows the application to access all the publishers the participant has.*

- void **get_subscribers** (**SubscriberSeq** subscribers)

  <<**eXtension**>> (p. 270) *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. 1478).*

- **FlowController lookup_flowcontroller** (String name)

  <<**eXtension**>> (p. 270) *Looks up an existing locally-created* ***com.rti.dds.publication.FlowController*** *(p. 942), 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. 1545), based on its name.*

- **TopicDescription lookup_topicdescription** (String topic_name)

  Looks up an existing, locally created ***com.rti.dds.topic.TopicDescription*** *(p. 1561), based on its name.*

- void **ignore_participant** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a remote ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- void **ignore_topic** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a ***com.rti.dds.topic.Topic*** *(p. 1545).*

- void **ignore_publication** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a ***publication*** *(p. 338).*

- void **ignore_subscription** (**InstanceHandle_t** handle)

  Instructs RTI Connext to locally ignore a ***subscription*** *(p. 343).*

- int **get_domain_id** ()

  Get the unique ***domain*** *(p. 317) identifier.*

- void **assert_liveliness** ()

  Manually asserts the liveliness of this ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- 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. 629).*

- void **add_peer** (String peer_desc_string)

  <<**eXtension**>> (p. 270) *Attempt to contact one or more additional peer participants.*

- void **remove_peer** (String peer_desc_string)

  <<**eXtension**>> (p. 270) *Remove one or more peer participants from the list of peers with which this* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) will try to communicate.*

- void **get_current_time** (**Time_t** current_time)

Returns the current value of the time.

- void **get_discovered_participants** (**InstanceHandleSeq** participant_handles)

  Returns list of discovered ***com.rti.dds.domain.DomainParticipant*** *(p. 629) s.*

- void **get_discovered_participant_data** (**ParticipantBuiltinTopicData** participant_data, **InstanceHandle_t** participant_handle)

  Returns ***builtin.ParticipantBuiltinTopicData*** *(p. 1227) for the specified* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) .*

- void **get_discovered_topics** (**InstanceHandleSeq** topic_handles)

  Returns list of discovered ***com.rti.dds.topic.Topic*** *(p. 1545) 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. 1545).*

- boolean **contains_entity** (**InstanceHandle_t** a_handle)

  Completes successfully with true if the referenced ***com.rti.dds.infrastructure.Entity*** *(p. 912) is contained by the* ***com.rti.dds.domain.DomainParticipant*** *(p. 629).*

- void **register_contentfilter** (String filter_name, **ContentFilter** contentfilter)

  <<**eXtension**>> (p. 270) *Register a content filter which can be used to create a* ***com.rti.dds.topic.ContentFilteredTopic*** *(p. 458).*

- **ContentFilter lookup_contentfilter** (String filter_name)

  <<**eXtension**>> (p. 270) *Lookup a content filter previously registered with* ***com.rti.dds.domain.DomainParticipant.register_contentfilter*** *(p. 698).*

- void **unregister_contentfilter** (String filter_name)

  <<**eXtension**>> (p. 270) *Unregister a content filter previously registered with* ***com.rti.dds.domain.DomainParticipant.register_contentfilter*** *(p. 698).*

- **Publisher get_implicit_publisher** ()

  <<**eXtension**>> (p. 270) *Returns the implicit* ***com.rti.dds.publication.Publisher*** *(p. 1277). If an implicit Publisher does not already exist, this creates one.*

- **Subscriber get_implicit_subscriber** ()

  <<**eXtension**>> (p. 270) *Returns the implicit* ***com.rti.dds.subscription.Subscriber*** *(p. 1478). If an implicit Subscriber does not already exist, this creates one.*

Static Public Attributes

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

  Special value for creating a ***com.rti.dds.topic.Topic*** *(p. 1545) with default QoS.*

- static final **PublisherQos PUBLISHER_QOS_DEFAULT** = new **PublisherQos**()

  Special value for creating a ***com.rti.dds.publication.Publisher*** *(p. 1277) with default QoS.*

- static final **SubscriberQos SUBSCRIBER_QOS_DEFAULT**

  Special value for creating a ***com.rti.dds.subscription.Subscriber*** *(p. 1478) with default QoS.*

- static final **FlowControllerProperty_t FLOW_CONTROLLER_PROPERTY_DEFAULT**

  <<**eXtension**>> (p. 270) *Special value for creating a* ***com.rti.dds.publication.FlowController*** *(p. 942) with default property.*

- static final String **SQLFILTER_NAME**

  <<**eXtension**>> (p. 270) *The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

- static final String **STRINGMATCHFILTER_NAME**

  <<**eXtension**>> (p. 270) *The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

8.61.1 Detailed Description

<<***interface***>> *(p. 271)* Container for all **com.rti.dds.infrastructure.DomainEntity** (p. 628) objects.

The **DomainParticipant** (p. 629) object plays several roles:

- It acts as a container for all other **com.rti.dds.infrastructure.Entity** (p. 912) objects.

- It acts as *factory* for the **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.topic.Topic** (p. 1545) and **com.rti.dds.topic.MultiTopic** (p. 1208) **com.rti.dds.infrastructure.Entity** (p. 912) 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** (p. 317) 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** (p. 317), offering operations that allow the application to ignore locally any information about a given participant (**ignore_participant()** (p. 686)), **publication** (p. 338) (**ignore_publication()** (p. 688)), **subscription** (p. 343) (**ignore_subscription()** (p. 689)) or **topic** (p. 350) (**ignore_topic()** (p. 687)).

The following operations may be called even if the **com.rti.dds.domain.DomainParticipant** (p. 629) is not enabled. (Operations NOT in this list will fail with the value RETCODE_NOT_ENABLED if called on a disabled **DomainParticipant** (p. 629)).

- Operations defined at the base-class level: **set_qos()** (p. 677), **set_qos_with_profile()** (p. 678), **get_qos()** (p. 679), **set_listener()** (p. 682), **get_listener()** (p. 682), **enable()** (p. 915);

- Factory operations: **create_flowcontroller()** (p. 654), **create_topic()** (p. 670), **create_topic_with_profile()** (p. 671), **create_publisher()** (p. 656), **create_publisher_with_profile()** (p. 657), **create_subscriber()** (p. 659), **create_subscriber_with_profile()** (p. 660), **delete_flowcontroller()** (p. 655), **delete_topic()** (p. 673), **delete_publisher()** (p. 658), **delete_subscriber()** (p. 661), **set_default_flowcontroller_property()** (p. 640), **get_default_flowcontroller_property()** (p. 639), **set_default_topic_qos()** (p. 642), **set_default_topic_qos_with_profile()** (p. 642), **get_default_topic_qos()** (p. 641), **set_default_publisher_qos()** (p. 644), **set_default_publisher_qos_with_profile()** (p. 645), **get_default_publisher_qos()** (p. 644), **set_default_subscriber_qos()** (p. 649), **set_default_subscriber_qos_with_profile()** (p. 650), **get_default_subscriber_qos()** (p. 652), **delete_contained_entities()** (p. 691), **set_default_datareader_qos()** (p. 652), **set_default_datareader_qos_with_profile()** (p. 653), **get_default_datareader_qos()** (p. 649), **set_default_datawriter_qos()** (p. 647), **set_default_datawriter_qos_with_profile()** (p. 648), **get_default_datawriter_qos()** (p. 647), **set_default_library()** (p. 679), **set_default_profile()** (p. 680);

- Operations for looking up topics: **lookup_topicdescription()** (p. 686);
- Operations that access status: **get_statuscondition()** (p. 917), **get_status_changes()** (p. 917).

QoS:

com.rti.dds.domain.DomainParticipantQos (p. 736)

Status:

Status Kinds (p. 106)

Listener:

com.rti.dds.domain.DomainParticipantListener (p. 734)

See also:

Operations Allowed in Listener Callbacks (p. 1156)

8.61.2 Member Function Documentation

8.61.2.1 void get_default_flowcontroller_property (FlowControllerProperty_t *prop*)

<<eXtension>> *(p. 270)* Copies the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) values for this **domain** (p. 317) participant into the given **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) 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. 640), or else, if the call was never made, the default values listed in **com.rti.dds.publication.FlowControllerProperty_t** (p. 946).

MT Safety:

UNSAFE. It is not safe to retrieve the default flow controller properties from a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_flowcontroller_property** (p. 640)

Parameters:

prop *<<in>>* *(p. 271)* Default property to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipant.FLOW_CONTROLLER_PROPERTY_-DEFAULT (p. 150)
com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 654)

8.61.2.2 void set_default_flowcontroller_property (FlowControllerProperty_t *prop*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) values for this **domain** (p. 317) participant.

This default value will be used for newly created **com.rti.dds.publication.FlowController** (p. 942) if **DomainParticipant.FLOW_CONTROLLER_PROPERTY_-DEFAULT** (p. 150) is specified as the `property` parameter when **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 654) is called.

Precondition:

The specified property values must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default flow controller properties for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_-default_flowcontroller_property** (p. 640) , **com.rti.dds.domain.DomainParticipant.get_-default_flowcontroller_property** (p. 639) or calling **com.rti.dds.domain.DomainParticipant.create_flowcontroller** (p. 654) with **DomainParticipant.FLOW_CONTROLLER_-PROPERTY_DEFAULT** (p. 150) as the `qos` parameter.

Parameters:

prop *<<in>>* *(p. 271)* Default property to be set. The special value **DomainParticipant.FLOW_CONTROLLER_PROPERTY_-DEFAULT** (p. 150) 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. 640) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 150)
com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 654)

8.61.2.3 void get_default_topic_qos (TopicQos *qos*)

Copies the default **com.rti.dds.topic.TopicQos** (p. 1566) values for this **domain** (p. 317) participant into the given **com.rti.dds.topic.TopicQos** (p. 1566) 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. 642), or else, if the call was never made, the default values listed in **com.rti.dds.topic.TopicQos** (p. 1566).

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. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642)

Parameters:

qos *<<in>> (p. 271)* Default qos to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipant.TOPIC_QOS_DEFAULT (p. 148)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.61.2.4 void set_default_topic_qos (TopicQos *qos*)

Set the default **com.rti.dds.topic.TopicQos** (p. 1566) values for this **domain** (p. 317) participant.

This default value will be used for newly created **com.rti.dds.topic.Topic** (p. 1545) if **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default **topic** (p. 350) QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642), **com.rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 641) or calling **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) with **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) as the `qos` parameter.

Parameters:

qos *<<in>>* *(p. 271)* Default qos to be set. The special value **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) 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. 642) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.TOPIC_QOS_DEFAULT (p. 148)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.61.2.5 void set_default_topic_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.topic.TopicQos**

(p. 1566) values for this **domain** (p. 317) participant based on the input XML QoS profile.

This default value will be used for newly created **com.rti.dds.topic.Topic** (p. 1545) if **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) is called.

Precondition:

The **com.rti.dds.topic.TopicQos** (p. 1566) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default **topic** (p. 350) QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642), **com.rti.dds.domain.DomainParticipant.get_default_topic_qos** (p. 641) or calling **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) with **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) as the `qos` parameter.

Parameters:

library_name *<<in>> (p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

If the input profile cannot be found the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.TOPIC_QOS_DEFAULT (p. 148)
com.rti.dds.domain.DomainParticipant.create_topic_with_profile (p. 671)

8.61.2.6 void get_default_publisher_qos (PublisherQos *qos*)

Copy the default **com.rti.dds.publication.PublisherQos** (p. 1303) values into the provided **com.rti.dds.publication.PublisherQos** (p. 1303) 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. 644), or **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos_with_profile** (p. 645), or else, if the call was never made, the default values listed in **com.rti.dds.publication.PublisherQos** (p. 1303).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) 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. 644), will be used to create the **com.rti.dds.publication.Publisher** (p. 1277).

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 644)

Parameters:

qos *<<**inout**>> (p. 271)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_publisher (p. 656)

8.61.2.7 void set_default_publisher_qos (PublisherQos *qos*)

Set the default **com.rti.dds.publication.PublisherQos** (p. 1303) values for this **DomainParticipant** (p. 629).

This set of default values will be used for a newly created **com.rti.dds.publication.Publisher** (p. 1277) if **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 644), **com.rti.dds.domain.DomainParticipant.get_default_publisher_qos** (p. 644) or calling **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) with **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) as the `qos` parameter.

Parameters:

qos *<<in>> (p. 271)* Default qos to be set. The special value **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) 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. 644) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_publisher (p. 656)

8.61.2.8 void set_default_publisher_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> (p. 270) Set the default **com.rti.dds.publication.PublisherQos** (p. 1303) values for this **DomainParticipant** (p. 629) 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. 1277) if **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) is called.

Precondition:

The **com.rti.dds.publication.PublisherQos** (p. 1303) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 644), **com.rti.dds.domain.DomainParticipant.get_default_publisher_qos** (p. 644) or calling **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) with **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) as the `qos` parameter.

Parameters:

library_name *<<in>> (p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_publisher_with_profile (p. 657)

8.61.2.9 void get_default_datawriter_qos (DataWriterQos *qos*)

<<eXtension>> *(p. 270)* Copy the default **com.rti.dds.publication.DataWriterQos** (p. 588) values into the provided **com.rti.dds.publication.DataWriterQos** (p. 588) 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. 647), or **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos_with_profile** (p. 648), or else, if the call was never made, the default values listed in **com.rti.dds.publication.DataWriterQos** (p. 588).

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 DataWriter QoS from a DomainPartipant while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 647).

Parameters:

qos *<<inout>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.61.2.10 void set_default_datawriter_qos (DataWriterQos *qos*)

<<eXtension>> *(p. 270)* Set the default DataWriterQos values for this **DomainParticipant** (p. 629).

This set of default values will be inherited for a newly created **com.rti.dds.publication.Publisher** (p. 1277).

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling

com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos (p. 647) or **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 647).

Parameters:

qos *<<in>>* *(p. 271)* Default qos to be set. The special value **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) 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. 647) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.61.2.11 void set_default_datawriter_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.publication.DataWriterQos** (p. 588) values for this **domain** (p. 317) 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. 1277).

Precondition:

The **com.rti.dds.publication.DataWriterQos** (p. 588) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datawriter_qos** (p. 647) or **com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos** (p. 647)

Parameters:

library_name *<<in>>* *(p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.61.2.12 void get_default_datareader_qos (DataReaderQos *qos*)

<<eXtension>> (p. 270) Copy the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values into the provided **com.rti.dds.subscription.DataReaderQos** (p. 518) 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. 652), or **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos_with_profile** (p. 653), or else, if the call was never made, the default values listed in **com.rti.dds.subscription.DataReaderQos** (p. 518).

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. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 652).

Parameters:

qos *<<inout>> (p. 271)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.61.2.13 void set_default_subscriber_qos (SubscriberQos *qos*)

Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1506) values for this Domain{articipant.

This set of default values will be used for a newly created **com.rti.dds.subscription.Subscriber** (p. 1478) if **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 649), **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 652) or calling **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659) with **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) as the `qos` parameter.

Parameters:

qos *<<in>>* *(p. 271)* Default qos to be set. The special value **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) 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. 649) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.61.2.14 void set_default_subscriber_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.subscription.SubscriberQos** (p. 1506) values for this **DomainParticipant** (p. 629) 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. 1478) if **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) is specified as the `qos`

parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659) is called.

Precondition:

The **com.rti.dds.subscription.SubscriberQos** (p. 1506) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 649), **com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos** (p. 652) or calling **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659) with **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) as the `qos` parameter.

Parameters:

library_name *<<in>>* *(p. 271)* 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. 679)).

profile_name *<<in>>* *(p. 271)* 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. 680)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_subscriber_with_profile (p. 660)

8.61.2.15 void get_default_subscriber_qos (SubscriberQos *qos*)

Copy the default **com.rti.dds.subscription.SubscriberQos** (p. 1506) values into the provided **com.rti.dds.subscription.SubscriberQos** (p. 1506) 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. 649), or **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos_with_profile** (p. 650), or else, if the call was never made, the default values listed in **com.rti.dds.subscription.SubscriberQos** (p. 1506).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipant.create_subscriber** (p. 659) 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. 652), will be used to create the **com.rti.dds.subscription.Subscriber** (p. 1478).

MT Safety:

UNSAFE. It is not safe to retrieve the default Subscriber QoS from a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_subscriber_qos** (p. 649).

Parameters:

qos *<<**inout**>> (p. 271)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_subscriber (p. 659)

8.61.2.16 void set_default_datareader_qos (DataReaderQos *qos*)

*<<**eXtension**>> (p. 270)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values for this **domain** (p. 317) participant.

This set of default values will be inherited for a newly created **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 652) or **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 649).

Parameters:

qos *<<in>>* *(p. 271)* Default qos to be set. The special value **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) 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. 652) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.61.2.17 void set_default_datareader_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values for this **DomainParticipant** (p. 629) 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. 1478).

Precondition:

The **com.rti.dds.subscription.DataReaderQos** (p. 518) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a **DomainParticipant** (p. 629) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_datareader_qos** (p. 652) or **com.rti.dds.domain.DomainParticipant.get_default_datareader_qos** (p. 649).

Parameters:

library_name *<<in>>* *(p. 271)* 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. 679)).

profile_name *<<in>>* *(p. 271)* 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. 680)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.61.2.18 FlowController create_flowcontroller (String *name*, FlowControllerProperty_t *prop*)

<<eXtension>> *(p. 270)* Creates a **com.rti.dds.publication.FlowController** (p. 942) with the desired property.

The created **com.rti.dds.publication.FlowController** (p. 942) is associated with a **com.rti.dds.publication.DataWriter** (p. 538) via **com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name** (p. 1310). A single **com.rti.dds.publication.FlowController** (p. 942) may service multiple **com.rti.dds.publication.DataWriter** (p. 538) instances, even if they belong to a different **com.rti.dds.publication.Publisher** (p. 1277). The `property` determines how the **com.rti.dds.publication.FlowController** (p. 942) shapes the network traffic.

Precondition:

The specified `property` must be consistent, or the operation will fail and no **com.rti.dds.publication.FlowController** (p. 942) will be created.

MT Safety:

UNSAFE. If **DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 150) 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. 640) or trying to lookup that flow controller with **com.rti.dds.domain.DomainParticipant.lookup_flowcontroller** (p. 684).

Parameters:

name *<<in>>* *(p. 271)* name of the **com.rti.dds.publication.FlowController** (p. 942) to create. A **com.rti.dds.publication.DataWriter** (p. 538) is associated with a **com.rti.dds.publication.FlowController** (p. 942) by name. Limited to 255 characters.

prop *<<in>>* *(p. 271)* property to be used for creating the new **com.rti.dds.publication.FlowController** (p. 942). The special value **DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 150) can be used to indicate that the **com.rti.dds.publication.FlowController** (p. 942) should be created with the default **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Returns:

Newly created flow controller object or NULL on failure.

See also:

com.rti.dds.publication.FlowControllerProperty_t (p. 946) for rules on consistency among property
DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 150)
com.rti.dds.domain.DomainParticipant.get_default_flowcontroller_property (p. 639)

8.61.2.19 void delete_flowcontroller (FlowController *fc*)

<<eXtension>> *(p. 270)* Deletes an existing **com.rti.dds.publication.FlowController** (p. 942).

Precondition:

The **com.rti.dds.publication.FlowController** (p. 942) must not have any attached **com.rti.dds.publication.DataWriter** (p. 538) objects. If there are any attached **com.rti.dds.publication.DataWriter** (p. 538) objects, it will fail with RETCODE_PRECONDITION_NOT_MET.
The **com.rti.dds.publication.FlowController** (p. 942) must have been created by this **com.rti.dds.domain.DomainParticipant** (p. 629), or else it will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

The **com.rti.dds.publication.FlowController** (p. 942) is deleted if this method completes successfully.

Parameters:

fc *<<in>>* *(p. 271)* The **com.rti.dds.publication.FlowController** (p. 942) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

8.61.2.20 Publisher create_publisher (PublisherQos *qos*, PublisherListener *listener*, int *mask*)

Creates a **com.rti.dds.publication.Publisher** (p. 1277) with the desired QoS policies and attaches to it the specified **com.rti.dds.publication.PublisherListener** (p. 1302).

Precondition:

The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.publication.Publisher** (p. 1277) will be created.

MT Safety:

UNSAFE. If **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) 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. 644).

Parameters:

qos *<<in>>* *(p. 271)* QoS to be used for creating the new **com.rti.dds.publication.Publisher** (p. 1277). The spe-

cial value **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) can be used to indicate that the **com.rti.dds.publication.Publisher** (p. 1277) should be created with the default **com.rti.dds.publication.PublisherQos** (p. 1303) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

listener *<<in>> (p. 271).* Listener to be attached to the newly created **com.rti.dds.publication.Publisher** (p. 1277).

mask *<<in>> (p. 271).* Changes of communication status to be invoked on the listener.

Returns:

newly created publisher object or NULL on failure.

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.PublisherQos (p. 1303) for rules on consistency among QoS
DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_publisher_with_profile (p. 657)
com.rti.dds.domain.DomainParticipant.get_default_publisher_qos (p. 644)
com.rti.dds.publication.Publisher.set_listener (p. 1294)

8.61.2.21 Publisher create_publisher_with_profile (String *library_name*, String *profile_name*, PublisherListener *listener*, int *mask*)

<<eXtension>> (p. 270) Creates a new **com.rti.dds.publication.Publisher** (p. 1277) object using the **com.rti.dds.publication.PublisherQos** (p. 1303) associated with the input XML QoS profile.

Precondition:

The **com.rti.dds.publication.PublisherQos** (p. 1303) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.publication.Publisher** (p. 1277) will be created.

Parameters:

library_name *<<in>> (p. 271)* Library name containing the XML QoS profile. If library_name is null RTI Connext will use the de-

fault library (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 679)).

profile_name <<*in*>> *(p. 271)* 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. 680)).

listener <<*in*>> *(p. 271)*. Listener to be attached to the newly created **com.rti.dds.publication.Publisher** (p. 1277).

mask <<*in*>> *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

newly created publisher object or NULL on failure.

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.PublisherQos (p. 1303) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_publisher (p. 656)
com.rti.dds.domain.DomainParticipant.get_default_publisher_qos (p. 644)
com.rti.dds.publication.Publisher.set_listener (p. 1294)

8.61.2.22 void delete_publisher (Publisher *p*)

Deletes an existing **com.rti.dds.publication.Publisher** (p. 1277).

Precondition:

The **com.rti.dds.publication.Publisher** (p. 1277) must not have any attached **com.rti.dds.publication.DataWriter** (p. 538) objects. If there are existing **com.rti.dds.publication.DataWriter** (p. 538) objects, it will fail with RETCODE_PRECONDITION_NOT_MET.
com.rti.dds.publication.Publisher (p. 1277) must have been created by this **com.rti.dds.domain.DomainParticipant** (p. 629), or else it will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.publication.Publisher** (p. 1277) will not be called after this method completes successfully.

Parameters:

p *<<in>>* *(p. 271)* **com.rti.dds.publication.Publisher** (p. 1277) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

8.61.2.23 Subscriber create_subscriber (SubscriberQos *qos*, SubscriberListener *listener*, int *mask*)

Creates a **com.rti.dds.subscription.Subscriber** (p. 1478) with the desired QoS policies and attaches to it the specified **com.rti.dds.subscription.SubscriberListener** (p. 1504).

Precondition:

The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.subscription.Subscriber** (p. 1478) will be created.

MT Safety:

UNSAFE. If **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) 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. 649).

Parameters:

qos *<<in>>* *(p. 271)* QoS to be used for creating the new **com.rti.dds.subscription.Subscriber** (p. 1478). The special value **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) can be used to indicate that the **com.rti.dds.subscription.Subscriber** (p. 1478) should be created with the default **com.rti.dds.subscription.SubscriberQos** (p. 1506) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

listener *<<in>>* *(p. 271)*. Listener to be attached to the newly created **com.rti.dds.subscription.Subscriber** (p. 1478).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

newly created subscriber object or NULL on failure.

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.SubscriberQos (p. 1506) for rules on consistency among QoS
DomainParticipant.SUBSCRIBER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_subscriber_with_profile (p. 660)
com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 652)
com.rti.dds.subscription.Subscriber.set_listener (p. 1498)

8.61.2.24 Subscriber create_subscriber_with_profile (String *library_name*, String *profile_name*, SubscriberListener *listener*, int *mask*)

<<eXtension>> *(p. 270)* Creates a new **com.rti.dds.subscription.Subscriber** (p. 1478) object using the **com.rti.dds.publication.PublisherQos** (p. 1303) associated with the input XML QoS profile.

Precondition:

The **com.rti.dds.subscription.SubscriberQos** (p. 1506) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.subscription.Subscriber** (p. 1478) will be created.

Parameters:

library_name *<<in>>* *(p. 271)* 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. 679)).

profile_name *<<in>>* *(p. 271)* 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. 680)).

listener *<<in>>* *(p. 271)*. Listener to be attached to the newly created **com.rti.dds.subscription.Subscriber** (p. 1478).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

newly created subscriber object or NULL on failure.

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.SubscriberQos (p. 1506) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_subscriber (p. 659)
com.rti.dds.domain.DomainParticipant.get_default_subscriber_qos (p. 652)
com.rti.dds.subscription.Subscriber.set_listener (p. 1498)

8.61.2.25 void delete_subscriber (Subscriber *s*)

Deletes an existing **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

The **com.rti.dds.subscription.Subscriber** (p. 1478) must not have any attached **com.rti.dds.subscription.DataReader** (p. 473) objects. If there are existing **com.rti.dds.subscription.DataReader** (p. 473) objects, it will fail with RETCODE_PRECONDITION_NOT_MET
The **com.rti.dds.subscription.Subscriber** (p. 1478) must have been created by this **com.rti.dds.domain.DomainParticipant** (p. 629), or else it will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

A Listener installed on the **com.rti.dds.subscription.Subscriber** (p. 1478) will not be called after this method completes successfully.

Parameters:

s *<<in>> (p. 271)* **com.rti.dds.subscription.Subscriber** (p. 1478) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

8.61.2.26 DataWriter create_datawriter (Topic *topic*, DataWriterQos *qos*, DataWriterListener *listener*, int *mask*)

<<eXtension>> (p. 270) Creates a **com.rti.dds.publication.DataWriter** (p. 538) that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1277).

Precondition:

The given **com.rti.dds.topic.Topic** (p. 1545) must have been created from the same **DomainParticipant** (p. 629) as the implicit Publisher. If it was created from a different **DomainParticipant** (p. 629), this method will fail.

The **com.rti.dds.publication.DataWriter** (p. 538) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) when the following methods are called: **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 661), **com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile** (p. 663), or **com.rti.dds.domain.DomainParticipant.get_implicit_publisher** (p. 701).

MT Safety:

UNSAFE. If **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) 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. 647).

Parameters:

topic *(p. **350**)* *<<in>>* *(p. 271)* The **com.rti.dds.topic.Topic** (p. 1545) that the **com.rti.dds.publication.DataWriter** (p. 538) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 271)* QoS to be used for creating the new **com.rti.dds.publication.DataWriter** (p. 538). The special value **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) should be created with the default **com.rti.dds.publication.DataWriterQos** (p. 588) set in the implicit **com.rti.dds.publication.Publisher** (p. 1277). The special value **Publisher.DATAWRITER_QOS_USE_TOPIC_QOS** (p. 178) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) should be created with the combination of the default **com.rti.dds.publication.DataWriterQos** (p. 588) set on the **com.rti.dds.publication.Publisher** (p. 1277) and the **com.rti.dds.topic.TopicQos** (p. 1566) of the **com.rti.dds.topic.Topic** (p. 1545). Cannot be NULL.

listener *<<in>>* *(p. 271)* The listener of the **com.rti.dds.publication.DataWriter** (p. 538).

mask *<<in>> (p. 271).* Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.publication.DataWriter** (p. 538) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataWriter
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS
Publisher.DATAWRITER_QOS_DEFAULT (p. 177)
Publisher.DATAWRITER_QOS_USE_TOPIC_QOS (p. 178)
com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile (p. 663)
com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos (p. 647)
com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 701)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.publication.DataWriter.set_listener (p. 545)

8.61.2.27 DataWriter create_datawriter_with_profile (Topic *topic*, String *library_name*, String *profile_name*, DataWriterListener *listener*, int *mask*)

<<eXtension>> (p. 270) Creates a **com.rti.dds.publication.DataWriter** (p. 538) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.publication.Publisher** (p. 1277).

Precondition:

The given **com.rti.dds.topic.Topic** (p. 1545) must have been created from the same **DomainParticipant** (p. 629) as the implicit Publisher. If it was created from a different **DomainParticipant** (p. 629), this method will return NULL.

The **com.rti.dds.publication.DataWriter** (p. 538) created using this method will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) when the following methods

are called: **com.rti.dds.domain.DomainParticipant.create_datawriter** (p. 661), **com.rti.dds.domain.DomainParticipant.create_datawriter_with_profile** (p. 663), or **com.rti.dds.domain.DomainParticipant.get_implicit_publisher** (p. 701)

Parameters:

topic *(p. **350**)* *<<in>>* *(p. 271)* The **com.rti.dds.topic.Topic** (p. 1545) that the **com.rti.dds.publication.DataWriter** (p. 538) will be associated with. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 679)).

profile_name *<<in>>* *(p. 271)* 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. 680)).

listener *<<in>>* *(p. 271)* The listener of the **com.rti.dds.publication.DataWriter** (p. 538).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.publication.DataWriter** (p. 538) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataWriter
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_datawriter (p. 661)
com.rti.dds.domain.DomainParticipant.get_default_datawriter_qos (p. 647)
com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 701)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.publication.DataWriter.set_listener (p. 545)

8.61.2.28 void delete_datawriter (DataWriter *a_datawriter*)

<<*eXtension*>> *(p. 270)* Deletes a **com.rti.dds.publication.DataWriter** (p. 538) that belongs to the implicit **com.rti.dds.publication.Publisher** (p. 1277).

The deletion of the **com.rti.dds.publication.DataWriter** (p. 538) will automatically unregister all instances. Depending on the settings of the **WRITER_DATA_LIFECYCLE** (p. 134) QosPolicy, the deletion of the **com.rti.dds.publication.DataWriter** (p. 538) may also dispose all instances.

8.61.3 Special Instructions if Using 'Timestamp' APIs and BY_SOURCE_TIMESTAMP Destination Ordering:

If the DataWriter's **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 609) is DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, calls to **delete_datawriter()** (p. 665) may fail if your application has previously used the 'with timestamp' APIs (write_w_timestamp(), register_instance_w_timestamp(), unregister_instance_w_timestamp(), or dispose_w_timestamp()) with a timestamp larger (later) than the time at which **delete_datawriter()** (p. 665) is called. To prevent **delete_datawriter()** (p. 665) from failing in this situation, either:

- Change the **WRITER_DATA_LIFECYCLE** (p. 134) QosPolicy so that RTI Connext will not autodispose unregistered instances (set **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_unregistered_instances** (p. 1723) to false.) or
- Explicitly call unregister_instance_w_timestamp() for all instances modified with the *_w_timestamp() APIs before calling **delete_datawriter()** (p. 665).

Precondition:

If the **com.rti.dds.publication.DataWriter** (p. 538) does not belong to the implicit **com.rti.dds.publication.Publisher** (p. 1277), the operation will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.publication.DataWriter** (p. 538) will not be called after this method completes successfully.

Parameters:

a_datawriter <<*in*>> *(p. 271)* The **com.rti.dds.publication.DataWriter** (p. 538) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET.

See also:

com.rti.dds.domain.DomainParticipant.get_implicit_publisher (p. 701)

8.61.3.1 DataReader create_datareader (TopicDescription *topic*, DataReaderQos *qos*, DataReaderListener *listener*, int *mask*)

<<***eXtension***>> *(p. 270)* Creates a **com.rti.dds.subscription.DataReader** (p. 473) that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

The given **com.rti.dds.topic.TopicDescription** (p. 1561) must have been created from the same **DomainParticipant** (p. 629) as the implicit Subscriber. If it was created from a different **DomainParticipant** (p. 629), this method will return NULL.

The **com.rti.dds.subscription.DataReader** (p. 473) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) when the following methods are called: **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 666), **com.rti.dds.domain.DomainParticipant.create_datareader_with_profile** (p. 668), or **com.rti.dds.domain.DomainParticipant.get_implicit_subscriber** (p. 701).

MT Safety:

UNSAFE. If **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) 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. 652).

Parameters:

topic *(p. **350**)* <<***in***>> *(p. 271)* The **com.rti.dds.topic.TopicDescription** (p. 1561) that the **com.rti.dds.subscription.DataReader** (p. 473) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 271)* The qos of the **com.rti.dds.subscription.DataReader** (p. 473). The special value **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 473) should be created with the default **com.rti.dds.subscription.DataReaderQos** (p. 518) set in the implicit **com.rti.dds.subscription.Subscriber** (p. 1478). If **com.rti.dds.topic.TopicDescription** (p. 1561) is of type **com.rti.dds.topic.Topic** (p. 1545) or **com.rti.dds.topic.ContentFilteredTopic** (p. 458), the special value **Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 191) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 473) should be created with the combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 518) set on the implicit **com.rti.dds.subscription.Subscriber** (p. 1478) and the **com.rti.dds.topic.TopicQos** (p. 1566) (in the case of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458), the **com.rti.dds.topic.TopicQos** (p. 1566) of the related **com.rti.dds.topic.Topic** (p. 1545)). if **Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 191) is used, `topic` (p. 350) cannot be a **com.rti.dds.topic.MultiTopic** (p. 1208). Cannot be NULL.

listener *<<in>>* *(p. 271)* The listener of the **com.rti.dds.subscription.DataReader** (p. 473).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.subscription.DataReader** (p. 473) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataReader
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_datareader_with_profile (p. 668)
com.rti.dds.domain.DomainParticipant.get_default_datareader_qos (p. 649)
com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 701)

com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.subscription.DataReader.set_listener (p. 482)

8.61.3.2 DataReader create_datareader_with_profile (TopicDescription *topic*, String *library_name*, String *profile_name*, DataReaderListener *listener*, int *mask*)

<<eXtension>> (p. 270) Creates a **com.rti.dds.subscription.DataReader** (p. 473) using a XML QoS profile that will be attached and belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

The given **com.rti.dds.topic.TopicDescription** (p. 1561) must have been created from the same **DomainParticipant** (p. 629) as the implicit subscriber. If it was created from a different **DomainParticipant** (p. 629), this method will return NULL.

The **com.rti.dds.subscription.DataReader** (p. 473) created using this method will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) when the following methods are called: **com.rti.dds.domain.DomainParticipant.create_datareader** (p. 666), **com.rti.dds.domain.DomainParticipant.create_datareader_with_profile** (p. 668), or **com.rti.dds.domain.DomainParticipant.get_implicit_subscriber** (p. 701)

Parameters:

topic *(p. **350**) <<in>> (p. 271)* The **com.rti.dds.topic.TopicDescription** (p. 1561) that the **com.rti.dds.subscription.DataReader** (p. 473) will be associated with. Cannot be NULL.

library_name *<<in>> (p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

listener *<<in>> (p. 271)* The listener of the **com.rti.dds.subscription.DataReader** (p. 473).

mask *<<in>> (p. 271).* Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.subscription.DataReader** (p. 473) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataReader
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_datareader (p. 666)
com.rti.dds.domain.DomainParticipant.get_default_datareader_qos (p. 649)
com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 701)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.subscription.DataReader.set_listener (p. 482)

8.61.3.3 void delete_datareader (DataReader *a_datareader*)

<<eXtension>> *(p. 270)* Deletes a **com.rti.dds.subscription.DataReader** (p. 473) that belongs to the implicit **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

If the **com.rti.dds.subscription.DataReader** (p. 473) does not belong to the implicit **com.rti.dds.subscription.Subscriber** (p. 1478), or if ther are any existing **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324) objects that are attached to the **com.rti.dds.subscription.DataReader** (p. 473), 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 RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.subscription.DataReader** (p. 473) will not be called after this method completes successfully.

Parameters:

a_datareader *<<in>>* *(p. 271)* The **com.rti.dds.subscription.DataReader** (p. 473) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET.

See also:

com.rti.dds.domain.DomainParticipant.get_implicit_subscriber (p. 701)

8.61.3.4 Topic create_topic (String *topic_name*, String *type_name*, TopicQos *qos*, TopicListener *listener*, int *mask*)

Creates a **com.rti.dds.topic.Topic** (p. 1545) with the desired QoS policies and attaches to it the specified **com.rti.dds.topic.TopicListener** (p. 1564).

Precondition:

The application is not allowed to create two **com.rti.dds.topic.Topic** (p. 1545) objects with the same `topic_name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 629). If the application attempts this, this method will fail and return a NULL **topic** (p. 350).
The specified QoS policies must be consistent, or the operation will fail and no **com.rti.dds.topic.Topic** (p. 1545) will be created.
Prior to creating a **com.rti.dds.topic.Topic** (p. 1545), the type must have been registered with RTI Connext. This is done using the **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) operation on a derived class of the TypeSupport interface.

MT Safety:

UNSAFE. It is not safe to create a **topic** (p. 350) while another thread is trying to lookup that **topic** (p. 350) description with **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 686).

MT Safety:

UNSAFE. If **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) is used for `qos`, it is not safe to create the **topic** (p. 350) while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642).

Parameters:

topic_name *<<in>> (p. 271)* Name for the new **topic** (p. 350), must not exceed 255 characters. Cannot be NULL.

type_name *<<in>>* *(p. 271)* The type to which the new **com.rti.dds.topic.Topic** (p. 1545) will be bound. Cannot be NULL.

qos *<<in>>* *(p. 271)* QoS to be used for creating the new **com.rti.dds.topic.Topic** (p. 1545). The special value **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) can be used to indicate that the **com.rti.dds.topic.Topic** (p. 1545) should be created with the default **com.rti.dds.topic.TopicQos** (p. 1566) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

listener *<<in>>* *(p. 271)*. Listener to be attached to the newly created **com.rti.dds.topic.Topic** (p. 1545).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

newly created **topic** (p. 350), or NULL on failure

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.topic.TopicQos (p. 1566) for rules on consistency among QoS
DomainParticipant.TOPIC_QOS_DEFAULT (p. 148)
com.rti.dds.domain.DomainParticipant.create_topic_with_profile (p. 671)
com.rti.dds.domain.DomainParticipant.get_default_topic_qos (p. 641)
com.rti.dds.topic.Topic.set_listener (p. 1549)

8.61.3.5 Topic create_topic_with_profile (String *topic_name*, String *type_name*, String *library_name*, String *profile_name*, TopicListener *listener*, int *mask*)

<<eXtension>> *(p. 270)* Creates a new **com.rti.dds.topic.Topic** (p. 1545) object using the **com.rti.dds.publication.PublisherQos** (p. 1303) associated with the input XML QoS profile.

Precondition:

The application is not allowed to create two **com.rti.dds.topic.TopicDescription** (p. 1561) objects with the same `topic_name` attached to the same

com.rti.dds.domain.DomainParticipant (p. 629). If the application attempts this, this method will fail and return a NULL **topic** (p. 350).
The **com.rti.dds.topic.TopicQos** (p. 1566) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.topic.Topic** (p. 1545) will be created.
Prior to creating a **com.rti.dds.topic.Topic** (p. 1545), the type must have been registered with RTI Connext. This is done using the **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) operation on a derived class of the TypeSupport interface.

MT Safety:

UNSAFE. It is not safe to create a **topic** (p. 350) while another thread is trying to lookup that **topic** (p. 350) description with **com.rti.dds.domain.DomainParticipant.lookup_topicdescription** (p. 686).

Parameters:

topic_name *<<in>> (p. 271)* Name for the new **topic** (p. 350), must not exceed 255 characters. Cannot be NULL.

type_name *<<in>> (p. 271)* The type to which the new **com.rti.dds.topic.Topic** (p. 1545) will be bound. Cannot be NULL.

library_name *<<in>> (p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

listener *<<in>> (p. 271).* Listener to be attached to the newly created **com.rti.dds.topic.Topic** (p. 1545).

mask *<<in>> (p. 271).* Changes of communication status to be invoked on the listener.

Returns:

newly created **topic** (p. 350), or NULL on failure

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation

com.rti.dds.topic.TopicQos (p. 1566) for rules on consistency among QoS
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)
com.rti.dds.domain.DomainParticipant.get_default_topic_qos (p. 641)
com.rti.dds.topic.Topic.set_listener (p. 1549)

8.61.3.6 void delete_topic (Topic *topic*)

Deletes a **com.rti.dds.topic.Topic** (p. 1545).

Precondition:

If the **com.rti.dds.topic.Topic** (p. 1545) does not belong to the application's **com.rti.dds.domain.DomainParticipant** (p. 629), this operation fails with RETCODE_PRECONDITION_NOT_MET.
Make sure no objects are using the **topic** (p. 350). More specifically, there must be no existing **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), or **com.rti.dds.topic.MultiTopic** (p. 1208) objects belonging to the same **com.rti.dds.domain.DomainParticipant** (p. 629) that are using the **com.rti.dds.topic.Topic** (p. 1545). If delete_topic is called on a **com.rti.dds.topic.Topic** (p. 1545) with any of these existing objects attached to it, it will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.topic.Topic** (p. 1545) will not be called after this method completes successfully.

Parameters:

topic *(p. **350**)* *<<in>>* *(p. 271)* **com.rti.dds.topic.Topic** (p. 1545) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET

8.61.3.7 ContentFilteredTopic create_contentfilteredtopic (String *name*, Topic *related_topic*, String *filter_expression*, StringSeq *expression_parameters*)

Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 458), that can be used to do content-based subscriptions.

The **com.rti.dds.topic.ContentFilteredTopic** (p. 458) only relates to samples published under that **com.rti.dds.topic.Topic** (p. 1545), 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. 278) describes the syntax of `filter_expression` and `expression_parameters`.

Precondition:

The application is not allowed to create two **com.rti.dds.topic.ContentFilteredTopic** (p. 458) objects with the same `topic_name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 629). 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. 629), this operation returns NULL.
This function will create a content filter using the **builtin** (p. 319) 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. Do not use rtiddsgen's -notypecode option if you want to use the **builtin** (p. 319) SQL filter.

Parameters:

name *<<in>>* *(p. 271)* Name for the new content filtered **topic** (p. 350), must not exceed 255 characters. Cannot be NULL.

related_topic *<<in>>* *(p. 271)* **com.rti.dds.topic.Topic** (p. 1545) to be filtered. Cannot be NULL.

filter_expression *<<in>>* *(p. 271)* Cannot be NULL

expression_parameters *<<in>>* *(p. 271)* 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. 458), or NULL on failure

8.61.3.8 ContentFilteredTopic create_contentfilteredtopic_with_filter (String *name*, Topic *related_topic*, String *filter_expression*, StringSeq *expression_parameters*, String *filter_name*)

<<eXtension>> *(p. 270)* Creates a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) using the specified filter to do content-based subscriptions.

Parameters:

name *<<in>>* *(p. 271)* Name for the new content filtered **topic** (p. 350). Cannot exceed 255 characters. Cannot be NULL.

related_topic *<<in>>* *(p. 271)* **com.rti.dds.topic.Topic** (p. 1545) to be filtered. Cannot be NULL.

filter_expression *<<in>>* *(p. 271)* Cannot be NULL.

expression_parameters *<<in>>* *(p. 271)* 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. 271)* Name of content filter to use. Must previously have been registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 698) on the same **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Builtin filter names are **DomainParticipant.SQLFILTER_NAME** (p. 151) and **DomainParticipant.STRINGMATCHFILTER_NAME** (p. 151)

Returns:

newly created **com.rti.dds.topic.ContentFilteredTopic** (p. 458), or NULL on failure

8.61.3.9 void delete_contentfilteredtopic (ContentFilteredTopic *a_contentfilteredtopic*)

Deletes a **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

Precondition:

The deletion of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458) is not allowed if there are any existing **com.rti.dds.subscription.DataReader** (p. 473) objects that are using the **com.rti.dds.topic.ContentFilteredTopic** (p. 458). If the operation is called on a **com.rti.dds.topic.ContentFilteredTopic** (p. 458)

with existing **com.rti.dds.subscription.DataReader** (p. 473) objects attached to it, it will fail with RETCODE_PRECONDITION_NOT_MET. The **com.rti.dds.topic.ContentFilteredTopic** (p. 458) must be created by this **com.rti.dds.domain.DomainParticipant** (p. 629), or else this operation will fail with RETCODE_PRECONDITION_NOT_MET.

Parameters:

a_contentfilteredtopic *<<in>> (p. 271)*

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET

8.61.3.10 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. 1545) are specified using the `subscription_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 278) describes the syntax of `subscription_expression` and `expression_parameters`.

Precondition:

The application is not allowed to create two **com.rti.dds.topic.TopicDescription** (p. 1561) objects with the same `name` attached to the same **com.rti.dds.domain.DomainParticipant** (p. 629). If the application attempts this, this method will fail and return NULL.
Prior to creating a **com.rti.dds.topic.MultiTopic** (p. 1208), the type must have been registered with RTI Connext. This is done using the **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) operation on a derived class of the TypeSupport interface. Otherwise, this method will return NULL.

Parameters:

name *<<in>> (p. 271)* Name of the newly create **com.rti.dds.topic.MultiTopic** (p. 1208). Cannot be NULL.

type_name *<<in>>* *(p. 271)* Cannot be NULL.

subscription_expression *<<in>>* *(p. 271)* Cannot be NULL.

expression_parameters *<<in>>* *(p. 271)* Cannot be NULL.

Returns:

NULL

8.61.3.11 void delete_multitopic (MultiTopic *a_multitopic*)

[**Not supported (optional)**] Deletes a **com.rti.dds.topic.MultiTopic** (p. 1208).

Precondition:

The deletion of a **com.rti.dds.topic.MultiTopic** (p. 1208) is not allowed if there are any existing **com.rti.dds.subscription.DataReader** (p. 473) objects that are using the **com.rti.dds.topic.MultiTopic** (p. 1208). If the delete_multitopic operation is called on a **com.rti.dds.topic.MultiTopic** (p. 1208) with existing **com.rti.dds.subscription.DataReader** (p. 473) objects attached to it, it will fail with RETCODE_PRECONDITION_NOT_MET.
The **com.rti.dds.topic.MultiTopic** (p. 1208) must be created by this **com.rti.dds.domain.DomainParticipant** (p. 629), or else this operation will fail with RETCODE_PRECONDITION_NOT_MET.

Parameters:

a_multitopic *<<in>>* *(p. 271)*

Exceptions:

RETCODE_UNSUPPORTED

8.61.3.12 void set_qos (DomainParticipantQos *qos*)

Change the QoS of this **DomainParticipant** (p. 629).

The **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 738) and **com.rti.dds.domain.DomainParticipantQos.entity_factory** (p. 738) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 271)* Set of policies to be applied to **com.rti.dds.domain.DomainParticipant** (p. 629). Policies

must be consistent. Immutable policies cannot be changed after **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled. The special value **DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 145) can be used to indicate that the QoS of the **com.rti.dds.domain.DomainParticipant** (p. 629) should be changed to match the current default **com.rti.dds.domain.DomainParticipantQos** (p. 736) set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 708). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY if immutable policy is changed, or RETCODE_INCONSISTENT_POLICY if policies are inconsistent

See also:

com.rti.dds.domain.DomainParticipantQos (p. 736) for rules on consistency among QoS
set_qos (abstract) (p. 913)

8.61.3.13 void set_qos_with_profile (String *library_name*, String *profile_name*)

*<<**eXtension**>> (p. 270)* Change the QoS of this **domain** (p. 317) participant using the input XML QoS profile.

The **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 738) and **com.rti.dds.domain.DomainParticipantQos.entity_factory** (p. 738) can be changed. The other policies are immutable.

Parameters:

library_name *<<**in**>> (p. 271)* 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. 721)).

profile_name *<<**in**>> (p. 271)* 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. 722)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY if immutable policy is changed, or RETCODE_INCONSISTENT_POLICY if policies are inconsistent

See also:

com.rti.dds.domain.DomainParticipantQos (p. 736) for rules on consistency among QoS

8.61.3.14 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. 271)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.61.3.15 String get_default_library ()

*<<**eXtension**>> (p. 270)* Gets the default XML library associated with a **com.rti.dds.domain.DomainParticipant** (p. 629).

Returns:

The default library or null if the default library was not set.

See also:

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

8.61.3.16 void set_default_library (String *library_name*)

*<<**eXtension**>> (p. 270)* Sets the default XML library for a **com.rti.dds.domain.DomainParticipant** (p. 629).

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'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. 629) inherits the default from the **com.rti.dds.domain.DomainParticipantFactory** (p. 708) (see **com.rti.dds.domain.DomainParticipantFactory.set_default_library** (p. 721)).

Parameters:

library_name *<<**in**>>* *(p. 271)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

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

8.61.3.17 String get_default_profile ()

*<<**eXtension**>>* *(p. 270)* Gets the default XML profile associated with a **com.rti.dds.domain.DomainParticipant** (p. 629).

Returns:

The default profile or null if the default profile was not set.

See also:

com.rti.dds.domain.DomainParticipant.set_default_profile (p. 680)

8.61.3.18 void set_default_profile (String *library_name*, String *profile_name*)

*<<**eXtension**>>* *(p. 270)* Sets the default XML profile for a **com.rti.dds.domain.DomainParticipant** (p. 629).

This method specifies the profile that will be used as the default the next time a default **DomainParticipant** (p. 629) profile is needed during a call to one of this DomainParticipant's operations. When calling a **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) inherits the default from the **com.rti.dds.domain.DomainParticipantFactory** (p. 708) (see **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 722)).

This method does not set the default QoS for entities created by the **com.rti.dds.domain.DomainParticipant** (p. 629); for this functionality, use the methods set_default_<entity>_qos_with_profile (you may pass in NULL after having called **set_default_profile()** (p. 680)).

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. 717).

Parameters:

library_name *<<in>> (p. 271)* The library name containing the profile.

profile_name *<<in>> (p. 271)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.domain.DomainParticipant.get_default_profile (p. 680)
com.rti.dds.domain.DomainParticipant.get_default_profile_library (p. 681)

8.61.3.19 String get_default_profile_library ()

*<<**eXtension**>> (p. 270)* Gets the library where the default XML QoS profile is contained for a **com.rti.dds.domain.DomainParticipant** (p. 629).

The default profile library is automatically set when **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 680) is called.

This library can be different than the **com.rti.dds.domain.DomainParticipant** (p. 629) default library (see **com.rti.dds.domain.DomainParticipant.get_default_library** (p. 679)).

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. 680)

8.61.3.20 void set_listener (DomainParticipantListener *l*, int *mask*)

Sets the participant listener.

Parameters:

l *<<in>>* *(p. 271)* Listener to be installed on entity.

mask *<<in>>* *(p. 271)* Changes of communication status to be invoked on the listener.

MT Safety:

Unsafe. This method is not synchronized with the listener callbacks, so it is possible to set a new listener on a participant when the old listener is in a callback.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.61.3.21 DomainParticipantListener get_listener ()

Get the participant listener.

Returns:

Existing listener attached to the **com.rti.dds.domain.DomainParticipant** (p. 629).

See also:

get_listener (abstract) (p. 915)

8.61.3.22 void get_publishers (PublisherSeq *publishers*)

<<*eXtension*>> *(p. 270)* 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 RETCODE_OUT_OF_RESOURCES.

MT Safety:

Safe.

Parameters:

publishers <<*inout*>> *(p. 271)* a PublisherSeq object where the set or list of publishers will be returned

Returns:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

8.61.3.23 void get_subscribers (SubscriberSeq *subscribers*)

<<*eXtension*>> *(p. 270)* 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 RETCODE_OUT_OF_RESOURCES.

MT Safety:

Safe.

Parameters:

subscribers <<*inout*>> *(p. 271)* a SubscriberSeq object where the set or list of subscribers will be returned

Returns:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

8.61.3.24 Subscriber get_builtin_subscriber ()

Accesses the **built-in com.rti.dds.subscription.Subscriber** (p. 1478).

Each **com.rti.dds.domain.DomainParticipant** (p. 629) contains several built-in **com.rti.dds.topic.Topic** (p. 1545) objects as well as corresponding **com.rti.dds.subscription.DataReader** (p. 473) objects to access them. All of these **com.rti.dds.subscription.DataReader** (p. 473) objects belong to a single built-in **com.rti.dds.subscription.Subscriber** (p. 1478).

The built-in Topics are used to communicate information about other **com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.subscription.DataReader** (p. 473), and **com.rti.dds.publication.DataWriter** (p. 538) 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. 629) is deleted.

Returns:

The built-in **com.rti.dds.subscription.Subscriber** (p. 1478) singleton.

See also:

builtin.SubscriptionBuiltinTopicData
builtin.PublicationBuiltinTopicData
builtin.ParticipantBuiltinTopicData (p. 1227)
builtin.TopicBuiltinTopicData

8.61.3.25 FlowController lookup_flowcontroller (String *name*)

*<<**eXtension**>>* *(p. 270)* Looks up an existing locally-created **com.rti.dds.publication.FlowController** (p. 942), based on its name.

Looks up a previously created **com.rti.dds.publication.FlowController** (p. 942), including the built-in ones. Once a **com.rti.dds.publication.FlowController** (p. 942) 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. 271)* Name of **com.rti.dds.publication.FlowController** (p. 942) to search for. Limited to 255 characters. Cannot be NULL.

Returns:

The flow controller if it has already been created locally, or NULL otherwise.

8.61.3.26 Topic find_topic (String *topic_name*, Duration_t *timeout*)

Finds an existing (or ready to exist) **com.rti.dds.topic.Topic** (p. 1545), 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. 1545) to be created.

If the requested **com.rti.dds.topic.Topic** (p. 1545) already exists, it is returned. Otherwise, **find_topic()** (p. 685) waits until another thread creates it or else returns when the specified timeout occurs.

find_topic() (p. 685) is useful when multiple threads are concurrently creating and looking up topics. In that case, one thread can call **find_topic()** (p. 685) and, if another thread has not yet created the **topic** (p. 350) 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. 686).

The **com.rti.dds.domain.DomainParticipant** (p. 629) must already be enabled.

Note: *Each* **com.rti.dds.topic.Topic** (p. 1545) obtained by **com.rti.dds.domain.DomainParticipant.find_topic** (p. 685) must also be deleted by means of **com.rti.dds.domain.DomainParticipant.delete_topic** (p. 673). If **com.rti.dds.topic.Topic** (p. 1545) is obtained multiple times by means of **com.rti.dds.domain.DomainParticipant.find_topic** (p. 685) or **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670), it must also be deleted that same number of times using **com.rti.dds.domain.DomainParticipant.delete_topic** (p. 673).

Parameters:

topic_name *<<in>>* *(p. 271)* Name of the **com.rti.dds.topic.Topic** (p. 1545) to search for. Cannot be NULL.

timeout *<<in>>* *(p. 271)* The time to wait if the **com.rti.dds.topic.Topic** (p. 1545) does not exist already. Cannot be NULL.

Returns:

the **topic** (p. 350), if it exists, or NULL

8.61.3.27 TopicDescription lookup_topicdescription (String *topic_name*)

Looks up an existing, locally created **com.rti.dds.topic.TopicDescription** (p. 1561), based on its name.

com.rti.dds.topic.TopicDescription (p. 1561) is the base class for **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.MultiTopic** (p. 1208) and **com.rti.dds.topic.ContentFilteredTopic** (p. 458). So you can narrow the **com.rti.dds.topic.TopicDescription** (p. 1561) returned from this operation to a **com.rti.dds.topic.Topic** (p. 1545) or **com.rti.dds.topic.ContentFilteredTopic** (p. 458) as appropriate.

Unlike **com.rti.dds.domain.DomainParticipant.find_topic** (p. 685), which logically returns a new **com.rti.dds.topic.Topic** (p. 1545) object that must be independently deleted, *this* operation returns a reference to the original local object.

The **com.rti.dds.domain.DomainParticipant** (p. 629) does not have to be enabled when you call **lookup_topicdescription()** (p. 686).

The returned **topic** (p. 350) may be either enabled or disabled.

MT Safety:

UNSAFE. It is not safe to lookup a **topic** (p. 350) description while another thread is creating that **topic** (p. 350).

Parameters:

topic_name *<<in>>* *(p. 271)* Name of **com.rti.dds.topic.TopicDescription** (p. 1561) to search for. This string must be no more than 255 characters; it cannot be NULL.

Returns:

The **topic** (p. 350) description, if it has already been created locally, otherwise it returns NULL.

8.61.3.28 void ignore_participant (InstanceHandle_t *handle*)

Instructs RTI Connext to locally ignore a remote **com.rti.dds.domain.DomainParticipant** (p. 629).

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** (p. 350), **publication** (p. 338), or **subscription** (p. 343) that originates on that **com.rti.dds.domain.DomainParticipant** (p. 629).

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 **builtin.ParticipantBuiltinTopicData** (p. 1227) to provide access control.

Application data can be associated with a **com.rti.dds.domain.DomainParticipant** (p. 629) by means of the **USER_DATA** (p. 126) policy. This application data is propagated as a field in the built-in **topic** (p. 350) and can be used by an application to implement its own access control policy.

The **com.rti.dds.domain.DomainParticipant** (p. 629) to ignore is identified by the `handle` argument. This `handle` is the one that appears in the **com.rti.dds.subscription.SampleInfo** (p. 1404) retrieved when reading the data-samples available for the built-in **com.rti.dds.subscription.DataReader** (p. 473) to the `com.rti.dds.domain.DomainParticipant` (p. 629) **topic** (p. 350). The built-in **com.rti.dds.subscription.DataReader** (p. 473) is read with the same com.rti.dds.topic.example.FooDataReader.read and com.rti.dds.topic.example.FooDataReader.take operations used for any **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

handle *<<in>> (p. 271)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) of the **com.rti.dds.domain.DomainParticipant** (p. 629) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_OUT_OF_RESOURCES, RETCODE_NOT_ENABLED

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)
ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME
com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.61.3.29 void ignore_topic (InstanceHandle_t *handle*)

Instructs RTI Connext to locally ignore a **com.rti.dds.topic.Topic** (p. 1545).

This means it will locally ignore any **publication** (p. 338), or **subscription** (p. 343) to the **com.rti.dds.topic.Topic** (p. 1545).

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. 1545) to ignore is identified by the `handle` argument. This is the handle of a **com.rti.dds.topic.Topic** (p. 1545) that appears in the **com.rti.dds.subscription.SampleInfo** (p. 1404) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 473) for the **com.rti.dds.topic.Topic** (p. 1545).

Parameters:

handle *<<in>> (p. 271)* Handle of the **com.rti.dds.topic.Topic** (p. 1545) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED

See also:

builtin.TopicBuiltinTopicData
TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME
com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.61.3.30 void ignore_publication (InstanceHandle_t *handle*)

Instructs RTI Connext to locally ignore a **publication** (p. 338).

A **publication** (p. 338) is defined by the association of a **topic** (p. 350) name, user data, and partition set on the **com.rti.dds.publication.Publisher** (p. 1277) (see builtin.PublicationBuiltinTopicData). After this call, any data written by that publication's **com.rti.dds.publication.DataWriter** (p. 538) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The **publication** (p. 338) (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. 1404) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 473) for the **com.rti.dds.publication** (p. 338) **topic** (p. 350).

- To ignore a *local* DataWriter, the `handle` can be obtained by calling **com.rti.dds.infrastructure.Entity.get_instance_handle** (p. 917) for the local DataWriter.

There is no way to reverse this operation.

Parameters:

handle *<<in>>* *(p. 271)* Handle of the **com.rti.dds.publication.DataWriter** (p. 538) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED

See also:

builtin.PublicationBuiltinTopicData
PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME
com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.61.3.31 void ignore_subscription (InstanceHandle_t *handle*)

Instructs RTI Connext to locally ignore a **subscription** (p. 343).

A **subscription** (p. 343) is defined by the association of a **topic** (p. 350) name, user data, and partition set on the **com.rti.dds.subscription.Subscriber** (p. 1478) (see builtin.SubscriptionBuiltinTopicData). After this call, any data received related to that subscription's **com.rti.dds.subscription.DataReader** (p. 473) will be ignored.

This operation can be used to ignore local *and* remote DataReaders.

The **subscription** (p. 343) 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. 1404) retrieved when reading data samples from the built-in **com.rti.dds.subscription.DataReader** (p. 473) for the **com.rti.dds.subscription** (p. 343) **topic** (p. 350).

- To ignore a *local* DataReader, the `handle` can be obtained by calling **com.rti.dds.infrastructure.Entity.get_instance_handle** (p. 917) for the local DataReader.

There is no way to reverse this operation.

Parameters:

handle *<<in>>* *(p. 271)* Handle of the **com.rti.dds.subscription.DataReader** (p. 473) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED

See also:

builtin.SubscriptionBuiltinTopicData
SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME
com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.61.3.32 int get_domain_id ()

Get the unique **domain** (p. 317) identifier.

This operation retrieves the `domain_id` used to create the **com.rti.dds.domain.DomainParticipant** (p. 629). The `domain_id` identifies the DDS **domain** (p. 317) to which the **com.rti.dds.domain.DomainParticipant** (p. 629) belongs. Each DDS **domain** (p. 317) represents a separate data 'communication plane' isolated from other domains.

Returns:

the unique `domainId` that was used to create the **domain** (p. 317)

See also:

com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 714)
com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile (p. 730)

8.61.3.33 void assert_liveliness ()

Manually asserts the liveliness of this **com.rti.dds.domain.DomainParticipant** (p. 629).

This is used in combination with the **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164) to indicate to RTI Connext that the entity remains active.

You need to use this operation if the **com.rti.dds.domain.DomainParticipant** (p. 629) conatins **com.rti.dds.publication.DataWriter** (p. 538) entities with the **com.rti.dds.infrastructure.LivelinessQosPolicy.kind** (p. 1167) set to LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS and it only affects the liveliness of those **com.rti.dds.publication.DataWriter** (p. 538) entities. Otherwise, it has no effect.

Note: writing data via the com.rti.dds.topic.example.FooDataWriter.write or com.rti.dds.topic.example.FooDataWriter.write_w_timestamp operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 538) itself and its **com.rti.dds.domain.DomainParticipant** (p. 629). Consequently the use of **assert_liveliness()** (p. 690) is only needed if the application is not writing data regularly.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_NOT_ENABLED

See also:

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1164)

8.61.3.34 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. 629).

This operation deletes all contained **com.rti.dds.publication.Publisher** (p. 1277) (including an implicit Publisher, if one exists), **com.rti.dds.subscription.Subscriber** (p. 1478) (including implicit subscriber), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208) objects.

Prior to deleting each contained entity, this operation will recursively call the corresponding delete_contained_entities operation on each contained entity (if applicable). This pattern is applied recursively. In this manner the operation **delete_contained_entities()** (p. 691) on the **com.rti.dds.domain.DomainParticipant** (p. 629) will end up deleting all the entities recursively contained in the **com.rti.dds.domain.DomainParticipant** (p. 629), that is also the **com.rti.dds.publication.DataWriter** (p. 538),

com.rti.dds.subscription.DataReader (p. 473), as well as the **com.rti.dds.subscription.QueryCondition** (p. 1324) and **com.rti.dds.subscription.ReadCondition** (p. 1326) objects belonging to the contained **com.rti.dds.subscription.DataReader** (p. 473).

The operation will fail with RETCODE_PRECONDITION_NOT_MET if any of the contained entities is in a state where it cannot be deleted.

If **delete_contained_entities()** (p. 691) completes successfully, the application may delete the **com.rti.dds.domain.DomainParticipant** (p. 629) knowing that it has no contained entities.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

8.61.3.35 void add_peer (String *peer_desc_string*)

<<***eXtension***>> *(p. 270)* 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. 629) will try to communicate.

This method may be called at any time after this **com.rti.dds.domain.DomainParticipant** (p. 629) has been created (before or after it has been enabled).

If this method is called after **com.rti.dds.infrastructure.Entity.enable** (p. 915), an attempt will be made to contact the new peer(s) immediately.

If this method is called *before* the **DomainParticipant** (p. 629) is enabled, the peer description will simply be added to the list that was populated by **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 626); the first attempted contact will take place after this **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) 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. 627) is set to true. Thus, this **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) *attempts* to discover, use the built-in participant **topic** (p. 350): ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME.

To remove specific peer locators, you may use **com.rti.dds.domain.DomainParticipant.remove_peer** (p. 693). 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. 629) that has been discovered, use **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 686).

Adding a peer description with this method has no effect on the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 626) that may be subsequently retrieved with **com.rti.dds.domain.DomainParticipant.get_qos()** (p. 679) (because **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 624) is immutable).

Parameters:

peer_desc_string *<<in>> (p. 271)* New peer descriptor to be added. The format is specified in **Peer Descriptor Format** (p. 56). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

Peer Descriptor Format (p. 56)
com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 626)
ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME
com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.61.3.36 void remove_peer (String *peer_desc_string*)

*<<**eXtension**>> (p. 270)* Remove one or more peer participants from the list

of peers with which this **com.rti.dds.domain.DomainParticipant** (p. 629) will try to communicate.

This method may be called any time after this **com.rti.dds.domain.DomainParticipant** (p. 629) has been enabled

Calling this method has the following effects:

- If a **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) 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. 629) 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. 692).

Removing a peer description with this method has no effect on the **com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers** (p. 626) that may be subsequently retrieved with **com.rti.dds.domain.DomainParticipant.get_qos()** (p. 679) (because **com.rti.dds.infrastructure.DiscoveryQosPolicy** (p. 624) is immutable).

Parameters:

peer_desc_string *<<in>> (p. 271)* Peer descriptor to be removed. The format is specified in **Peer Descriptor Format** (p. 56). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

Peer Descriptor Format (p. 56)
com.rti.dds.infrastructure.DiscoveryQosPolicy.initial_peers (p. 626)
com.rti.dds.domain.DomainParticipant.add_peer (p. 692)

8.61.3.37 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. 538) and to set the reception-timestamp for the data updates that it receives.

Parameters:

current_time *<<**inout**>>* *(p. 271)* Current time to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.61.3.38 void get_discovered_participants (InstanceHandleSeq *participant_handles*)

Returns list of discovered **com.rti.dds.domain.DomainParticipant** (p. 629) s.

This operation retrieves the list of **com.rti.dds.domain.DomainParticipant** (p. 629) s that have been discovered in the **domain** (p. 317) and that the application has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_participant** (p. 686) operation.

Parameters:

participant_handles *<<**inout**>>* *(p. 271)*
com.rti.dds.infrastructure.InstanceHandleSeq (p. 1083) to be filled with handles of the discovered **com.rti.dds.domain.DomainParticipant** (p. 629) s

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED

8.61.3.39 void get_discovered_participant_data (ParticipantBuiltinTopicData *participant_data*, InstanceHandle_t *participant_handle*)

Returns **builtin.ParticipantBuiltinTopicData** (p. 1227) for the specified **com.rti.dds.domain.DomainParticipant** (p. 629) .

This operation retrieves information on a **com.rti.dds.domain.DomainParticipant** (p. 629) that has been discovered on the network. The participant must be in the same **domain** (p. 317) 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. 686) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 629). Otherwise, the operation will fail with PRECONDITION_NOT_MET.

Use the operation **com.rti.dds.domain.DomainParticipant.get_discovered_participants** (p. 695) to find the **com.rti.dds.domain.DomainParticipant** (p. 629) s that are currently discovered.

Note: This operation does not retrieve the **builtin.ParticipantBuiltinTopicData.property** (p. 1228). This information is available through **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 503) (if a reader listener is installed on the **builtin.ParticipantBuiltinTopicDataDataReader** (p. 1230)).

Parameters:

participant_data *<<inout>> (p. 271)* **builtin.ParticipantBuiltinTopicData** (p. 1227) to be filled with the specified **com.rti.dds.domain.DomainParticipant** (p. 629) 's data.

participant_handle *<<in>> (p. 271)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) of **com.rti.dds.domain.DomainParticipant** (p. 629).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)
com.rti.dds.domain.DomainParticipant.get_discovered_participants (p. 695)

8.61.3.40 void get_discovered_topics (InstanceHandleSeq *topic_handles*)

Returns list of discovered **com.rti.dds.topic.Topic** (p. 1545) objects.

This operation retrieves the list of **com.rti.dds.topic.Topic** (p. 1545) s that have been discovered in the **domain** (p. 317) and that the

application has not indicated should be "ignored" by means of the **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 687) operation.

Parameters:

topic_handles *<<inout>> (p. 271)* **com.rti.dds.infrastructure.InstanceHandleSeq** (p. 1083) to be filled with handles of the discovered **com.rti.dds.topic.Topic** (p. 1545) objects

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED

8.61.3.41 void get_discovered_topic_data (TopicBuiltinTopicData *topic_data*, InstanceHandle_t *topic_handle*)

Returns builtin.TopicBuiltinTopicData for the specified **com.rti.dds.topic.Topic** (p. 1545).

This operation retrieves information on a **com.rti.dds.topic.Topic** (p. 1545) 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. 687) operation.

The `topic_handle` must correspond to such a **topic** (p. 350). Otherwise, the operation will fail with RETCODE_PRECONDITION_NOT_MET.

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with RETCODE_UNSUPPORTED.

Use the operation **com.rti.dds.domain.DomainParticipant.get_discovered_topics** (p. 696) to find the topics that are currently discovered.

Parameters:

topic_data *<<inout>> (p. 271)* builtin.TopicBuiltinTopicData to be filled with the specified **com.rti.dds.topic.Topic** (p. 1545)'s data.

topic_handle *<<in>> (p. 271)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) of **com.rti.dds.topic.Topic** (p. 1545).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED

See also:

builtin.TopicBuiltinTopicData

com.rti.dds.domain.DomainParticipant.get_discovered_topics (p. 696)

8.61.3.42 boolean contains_entity (InstanceHandle_t *a_handle*)

Completes successfully with true if the referenced **com.rti.dds.infrastructure.Entity** (p. 912) is contained by the **com.rti.dds.domain.DomainParticipant** (p. 629).

This operation checks whether or not the given `a_handle` represents an **com.rti.dds.infrastructure.Entity** (p. 912) that was created from the **com.rti.dds.domain.DomainParticipant** (p. 629). The containment applies recursively. That is, it applies both to entities (**com.rti.dds.topic.TopicDescription** (p. 1561), **com.rti.dds.publication.Publisher** (p. 1277), or **com.rti.dds.subscription.Subscriber** (p. 1478)) created directly using the **com.rti.dds.domain.DomainParticipant** (p. 629) as well as entities created using a contained **com.rti.dds.publication.Publisher** (p. 1277), or **com.rti.dds.subscription.Subscriber** (p. 1478) as the factory, and so forth.

The `instance` handle for an **com.rti.dds.infrastructure.Entity** (p. 912) may be obtained from built-in **topic** (p. 350) data, from various statuses, or from the operation **com.rti.dds.infrastructure.Entity.get_instance_handle** (p. 917).

Parameters:

a_handle *<<in>>* *(p. 271)* **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) of the **com.rti.dds.infrastructure.Entity** (p. 912) to be checked.

Returns:

true if **com.rti.dds.infrastructure.Entity** (p. 912) is contained by the **com.rti.dds.domain.DomainParticipant** (p. 629), or false otherwise.

8.61.3.43 void register_contentfilter (String *filter_name*, ContentFilter *contentfilter*)

*<<**eXtension**>>* *(p. 270)* Register a content filter which can be used to create a **com.rti.dds.topic.ContentFilteredTopic** (p. 458).

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. 458) and the corresponding **com.rti.dds.subscription.DataReader** (p. 473).
- In each application that writes the data to the applications mentioned above.

For example, suppose Application A on the **subscription** (p. 343) 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** (p. 343) 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.DomainParticipant** (p. 629).

Parameters:

filter_name *<<in>> (p. 271)* Name of the filter. The name must be unique within the **com.rti.dds.domain.DomainParticipant** (p. 629) and must not exceed 255 characters. Cannot be NULL.

contentfilter *<<in>> (p. 271)* Content filter to be registered. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.domain.DomainParticipant.unregister_contentfilter (p. 700)

8.61.3.44 ContentFilter lookup_contentfilter (String *filter_name*)

<<eXtension>> (p. 270) Lookup a content filter previously registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 698).

Parameters:

filter_name *<<in>> (p. 271)* 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. 629) with

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 698). Otherwise, return the **com.rti.dds.topic.ContentFilter** (p. 454) that has been previosuly registered with the given filter_name.

See also:

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 698)

8.61.3.45 void unregister_contentfilter (String *filter_name*)

<<eXtension>> *(p. 270)* Unregister a content filter previously registered with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 698).

A `filter_name` can be unregistered only if it has been previously registered to the **com.rti.dds.domain.DomainParticipant** (p. 629) with **com.rti.dds.domain.DomainParticipant.register_contentfilter** (p. 698).

The unregistration of filter is not allowed if there are any existing **com.rti.dds.topic.ContentFilteredTopic** (p. 458) objects that are using the filter. If the operation is called on a filter with existing **com.rti.dds.topic.ContentFilteredTopic** (p. 458) objects attached to it, this operation will fail with RETCODE_PRECONDITION_NOT_MET.

If there are still existing discovered **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) s, finalize method of the filter will be called on those discovered **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

Parameters:

filter_name *<<in>>* *(p. 271)* Name of the filter. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET

See also:

com.rti.dds.domain.DomainParticipant.register_contentfilter (p. 698)

8.61.3.46 Publisher get_implicit_publisher ()

<<*eXtension*>> *(p. 270)* Returns the implicit **com.rti.dds.publication.Publisher** (p. 1277). If an implicit Publisher does not already exist, this creates one.

There can only be one implicit Publisher per **DomainParticipant** (p. 629).

The implicit Publisher is created with **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) 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. 691), or **com.rti.dds.domain.DomainParticipant.delete_publisher** (p. 658) with the implicit publisher as a parameter. Additionally, when a **DomainParticipant** (p. 629) is deleted, if there are no attached DataWriters that belong to the implicit Publisher, the implicit Publisher will be implicitly deleted.

MT Safety:

UNSAFE. It is not safe to create an implicit Publisher while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipant.set_default_publisher_qos** (p. 644).

Returns:

The implicit publisher

See also:

DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_publisher (p. 656)

8.61.3.47 Subscriber get_implicit_subscriber ()

<<*eXtension*>> *(p. 270)* Returns the implicit **com.rti.dds.subscription.Subscriber** (p. 1478). If an implicit Subscriber does not already exist, this creates one.

There can only be one implicit Subscriber per **DomainParticipant** (p. 629).

The implicit Subscriber is created with **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) 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. 691), or **com.rti.dds.domain.DomainParticipant.delete_subscriber** (p. 661) with the subscriber as a parameter. Additionally, when a **DomainParticipant** (p. 629) 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. 649).

Returns:

The implicit subscriber

See also:

DomainParticipant.PUBLISHER_QOS_DEFAULT (p. 149)
com.rti.dds.domain.DomainParticipant.create_subscriber (p. 659)

8.62 DomainParticipantAdapter Class Reference

<<*eXtension*>> *(p. 270)* 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_inconsistent_topic** (**Topic** topic, **InconsistentTopicStatus** status)

  Handle the StatusKind.INCONSISTENT_TOPIC_STATUS status.

- ^ void **on_offered_deadline_missed** (**DataWriter** writer, **OfferedDeadlineMissedStatus** status)

  Handles the StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

- ^ void **on_offered_incompatible_qos** (**DataWriter** writer, **OfferedIncompatibleQosStatus** status)

  Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

- ^ void **on_liveliness_lost** (**DataWriter** writer, **LivelinessLostStatus** status)

  Handles the StatusKind.LIVELINESS_LOST_STATUS status.

- ^ void **on_publication_matched** (**DataWriter** writer, **PublicationMatchedStatus** status)

  Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

- ^ void **on_reliable_reader_activity_changed** (**DataWriter** writer, **ReliableReaderActivityChangedStatus** status)

  <<**eXtension**>> (p. 270) *A matched reliable reader has become active or become inactive.*

- ^ void **on_reliable_writer_cache_changed** (**DataWriter** writer, **ReliableWriterCacheChangedStatus** status)

  <<**eXtension**>> (p. 270) *A change has occurred in the writer's cache of unacknowledged samples.*

ˆ void **on_instance_replaced** (**DataWriter** writer, **InstanceHandle_t** handle)

Notifies when an instance is replaced in DataWriter queue.

8.62.1 Detailed Description

*<<**eXtension**>> (p. 270)* 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.

8.62.2 Member Function Documentation

8.62.2.1 void on_inconsistent_topic (Topic *topic*, InconsistentTopicStatus *status*)

Handle the StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1545) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1545) of the same **topic** (p. 350) name.

Parameters:

topic *(p. **350**) <<**out**>> (p. 271)* Locally created **com.rti.dds.topic.Topic** (p. 1545) that triggers the listener callback

status *<<**out**>> (p. 271)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1545)

Implements **TopicListener** (p. 1565).

8.62.2.2 void on_offered_deadline_missed (DataWriter *writer*, OfferedDeadlineMissedStatus *status*)

Handles the StatusKind.OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **DEADLINE** (p. 50) 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. 538) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current deadline missed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 567).

8.62.2.3 void on_offered_incompatible_qos (DataWriter *writer*, OfferedIncompatibleQosStatus *status*)

Handles the StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriterQos** (p. 588) of the **com.rti.dds.publication.DataWriter** (p. 538) was incompatible with what was requested by a **com.rti.dds.subscription.DataReader** (p. 473). This callback is called when a **com.rti.dds.publication.DataWriter** (p. 538) has discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition, but with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

writer <<*out*>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<*out*>> *(p. 271)* Current incompatible qos status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 568).

8.62.2.4 void on_liveliness_lost (DataWriter *writer*, LivelinessLostStatus *status*)

Handles the StatusKind.LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **LIVELINESS** (p. 78) qos policy was not respected; this

com.rti.dds.subscription.DataReader (p. 473) entities will consider the **com.rti.dds.publication.DataWriter** (p. 538) as no longer "alive/active". This callback will not be called when an already not alive **com.rti.dds.publication.DataWriter** (p. 538) simply renames not alive for another liveliness period.

Parameters:

writer *<<out>> (p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status *<<out>> (p. 271)* Current liveliness lost status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 568).

8.62.2.5 void on_publication_matched (DataWriter *writer*, PublicationMatchedStatus *status*)

Handles the StatusKind.PUBLICATION_MATCHED_STATUS status.

This callback is called when the **com.rti.dds.publication.DataWriter** (p. 538) has found a **com.rti.dds.subscription.DataReader** (p. 473) that matches the **com.rti.dds.topic.Topic** (p. 1545), has a common partition and compatible QoS, or has ceased to be matched with a **com.rti.dds.subscription.DataReader** (p. 473) that was previously considered to be matched.

Parameters:

writer *<<out>> (p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status *<<out>> (p. 271)* Current **publication** (p. 338) match status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 569).

8.62.2.6 void on_reliable_reader_activity_changed (DataWriter *writer*, ReliableReaderActivityChangedStatus *status*)

<<eXtension>> (p. 270) A matched reliable reader has become active or become inactive.

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current reliable reader activity changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 570).

8.62.2.7 void on_reliable_writer_cache_changed (DataWriter *writer*, ReliableWriterCacheChangedStatus *status*)

<<***eXtension***>> *(p. 270)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

status <<***out***>> *(p. 271)* Current reliable writer cache changed status of locally created **com.rti.dds.publication.DataWriter** (p. 538)

Implements **DataWriterListener** (p. 569).

8.62.2.8 void on_instance_replaced (DataWriter *writer*, InstanceHandle_t *handle*)

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. 538) 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.

Parameters:

writer <<***out***>> *(p. 271)* Locally created **com.rti.dds.publication.DataWriter** (p. 538) that triggers the listener callback

handle <<***out***>> *(p. 271)* Handle of the replaced instance

Implements **DataWriterListener** (p. 570).

8.63 DomainParticipantFactory Class Reference

*<<**singleton**>>* *(p. 271)* *<<**interface**>>* *(p. 271)* Allows creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 629) objects.

Public Member Functions

- abstract **DomainParticipant create_participant** (int domainId, **DomainParticipantQos** qos, **DomainParticipantListener** listener, int mask)

  *Creates a new **com.rti.dds.domain.DomainParticipant** (p. 629) object.*

- abstract void **delete_participant** (**DomainParticipant** a_participant)

  *Deletes an existing **com.rti.dds.domain.DomainParticipant** (p. 629).*

- abstract void **get_default_participant_qos** (**DomainParticipantQos** qos)

  *Initializes the **com.rti.dds.domain.DomainParticipantQos** (p. 736) instance with default values.*

- abstract void **set_default_participant_qos** (**DomainParticipantQos** qos)

  *Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 736) values for this **domain** (p. 317) participant factory.*

- abstract void **set_default_participant_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 736) values for this **domain** (p. 317) participant factory based on the input XML QoS profile.*

- abstract **DomainParticipant lookup_participant** (int domainId)

  *Locates an existing **com.rti.dds.domain.DomainParticipant** (p. 629).*

- 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. 270) *Loads the XML QoS profiles.*

- abstract void **reload_profiles** ()

  <<**eXtension**>> (p. 270) *Reloads the XML QoS profiles.*

- abstract void **unload_profiles** ()

  <<**eXtension**>> (p. 270) *Unloads the XML QoS profiles.*

- abstract String **get_default_library** ()

  <<**eXtension**>> (p. 270) *Gets the default XML library associated with a* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- abstract void **set_default_library** (String library_name)

  <<**eXtension**>> (p. 270) *Sets the default XML library for a* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- abstract String **get_default_profile** ()

  <<**eXtension**>> (p. 270) *Gets the default XML profile associated with a* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- abstract void **set_default_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Sets the default XML profile for a* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- abstract String **get_default_profile_library** ()

  <<**eXtension**>> (p. 270) *Gets the library where the default XML profile is contained for a* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708).*

- abstract void **get_participant_qos_from_profile** (**DomainParticipantQos** qos, String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Gets the* ***com.rti.dds.domain.DomainParticipantQos*** *(p. 736) 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. 270) *Gets the* ***com.rti.dds.publication.PublisherQos*** *(p. 1303) 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. 270) *Gets the* ***com.rti.dds.subscription.SubscriberQos*** *(p. 1506) 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. 270) *Gets the* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) 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. 270) *Gets the* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) values associated with the input XML QoS profile while applying* ***topic*** *(p. 350) filters to the input* ***topic*** *(p. 350) name.*

^ abstract void **get_datareader_qos_from_profile** (**DataReaderQos** qos, String library_name, String profile_name)

<<**eXtension**>> (p. 270) *Gets the* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) 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. 270) *Gets the* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) values associated with the input XML QoS profile while applying* ***topic*** *(p. 350) filters to the input* ***topic*** *(p. 350) name.*

^ abstract void **get_topic_qos_from_profile** (**TopicQos** qos, String library_name, String profile_name)

<<**eXtension**>> (p. 270) *Gets the* ***com.rti.dds.topic.TopicQos*** *(p. 1566) 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. 270) *Gets the* ***com.rti.dds.topic.TopicQos*** *(p. 1566) values associated with the input XML QoS profile while applying* ***topic*** *(p. 350) filters to the input* ***topic*** *(p. 350) name.*

^ abstract void **get_qos_profile_libraries** (**StringSeq** library_names)

<<**eXtension**>> (p. 270) *Gets the names of all XML QoS profile libraries associated with the* ***com.rti.dds.domain.DomainParticipantFactory*** *(p. 708)*

^ abstract void **get_qos_profiles** (**StringSeq** profile_names, String library_name)

<<**eXtension**>> (p. 270) *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. 270) *Creates a new* ***com.rti.dds.domain.DomainParticipant*** *(p. 629) object using the* ***com.rti.dds.domain.DomainParticipantQos*** *(p. 736) associated with the input XML QoS profile.*

- abstract void **unregister_thread** ()

  <<**eXtension**>> (p. 270) *Allows the user to release thread specific resources kept by the middleware.*

Static Public Member Functions

- static final **DomainParticipantFactory get_instance** ()

  Gets the singleton instance of this class.

- static final void **finalize_instance** ()

  <<**eXtension**>> (p. 270) *Destroys the singleton instance of this class.*

Static Public Attributes

- static final **DomainParticipantQos PARTICIPANT_QOS_DEFAULT**

  Special value for creating a ***DomainParticipant*** *(p. 629) 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. 712).*

8.63.1 Detailed Description

<<***singleton***>> *(p. 271)* <<***interface***>> *(p. 271)* Allows creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 629) objects.

The sole purpose of this class is to allow the creation and destruction of **com.rti.dds.domain.DomainParticipant** (p. 629) objects. This class itself is a <<***singleton***>> *(p. 271)*, and accessed via the **get_instance()** (p. 712) method, and destroyed with **finalize_instance()** (p. 713) method.

A single application can participate in multiple domains by instantiating multiple **com.rti.dds.domain.DomainParticipant** (p. 629) objects.

An application may even instantiate multiple participants in the same **domain** (p. 317). Participants in the same **domain** (p. 317) 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** (p. 317), 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. 1714)). The port numbers used by a participant are calculated based on both the participant index and the **domain** (p. 317) ID, so if all participants on the same node have different participant indexes, they can coexist in the same **domain** (p. 317).

- You cannot mix entities from different participants. For example, you cannot delete a **topic** (p. 350) on a different participant than you created it from, and you cannot ask a subscriber to create a reader for a **topic** (p. 350) 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** (p. 317) 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. 629)

8.63.2 Member Function Documentation

8.63.2.1 static final DomainParticipantFactory get_instance () [static]

Gets the singleton instance of this class.

MT Safety:

On non-Linux systems: UNSAFE for multiple threads to simultaneously make the FIRST call to either **com.rti.dds.domain.DomainParticipantFactory.get_instance()** (p. 712) or **com.rti.dds.domain.DomainParticipantFactory.finalize_instance()** (p. 713). Subsequent calls are thread safe. (On Linux systems, these calls are thread safe.)

DomainParticipantFactory.TheParticipantFactory (p. 144) can be used as an alias for the singleton factory returned by this operation.

Returns:

The singleton **com.rti.dds.domain.DomainParticipantFactory** (p. 708) instance.

See also:

DomainParticipantFactory.TheParticipantFactory (p. 144)

8.63.2.2 static final void finalize_instance () [static]

<<***eXtension***>> *(p. 270)* 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. 712) will return a new lifecycle of the singleton.

MT Safety:

On non-Linux systems: UNSAFE for multiple threads to simultaneously make the FIRST call to either **com.rti.dds.domain.DomainParticipantFactory.get_instance()** (p. 712) or **com.rti.dds.domain.DomainParticipantFactory.finalize_instance()** (p. 713). Subsequent calls are thread safe. (On Linux systems, these calls are thread safe.)

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET

8.63.2.3 abstract DomainParticipant create_participant (int *domainId*, DomainParticipantQos *qos*, DomainParticipantListener *listener*, int *mask*) [pure virtual]

Creates a new **com.rti.dds.domain.DomainParticipant** (p. 629) object.

Precondition:

The specified QoS policies must be consistent or the operation will fail and no **com.rti.dds.domain.DomainParticipant** (p. 629) will be created.

If you want to create multiple participants on a given host in the same **domain** (p. 317), make sure each one has a different participant index (set in the **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709)). 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** (p. 317) ID).

Note that if there is a single participant per host in a given **domain** (p. 317), the participant index can be left at the default value (-1).

Parameters:

domainId <<*in*>> *(p. 271)* ID of the **domain** (p. 317) that the application intends to join. [**range**] [>=0], and does not violate guidelines stated in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1396).

qos <<*in*>> *(p. 271)* the DomainParticipant's QoS. The special value **DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 145) can be used to indicate that the **com.rti.dds.domain.DomainParticipant** (p. 629) should be created with the default **com.rti.dds.domain.DomainParticipantQos** (p. 736) set in the **com.rti.dds.domain.DomainParticipantFactory** (p. 708). Cannot be NULL.

listener <<*in*>> *(p. 271)* the **domain** (p. 317) participant's listener.

mask <<*in*>> *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

domain (p. 317) participant or NULL on failure

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.domain.DomainParticipantQos (p. 736) for rules on consistency among QoS
DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 145)
NDDS_DISCOVERY_PEERS (p. 55)
com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile() (p. 730)
com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos() (p. 716)
com.rti.dds.domain.DomainParticipant.set_listener() (p. 682)

8.63.2.4 abstract void delete_participant (DomainParticipant *a_participant*) [pure virtual]

Deletes an existing **com.rti.dds.domain.DomainParticipant** (p. 629).

Precondition:

All **domain** (p. 317) entities belonging to the participant must have already been deleted. Otherwise it fails with the error RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.domain.DomainParticipant** (p. 629) will not be called after this method returns successfully.

Parameters:

a_participant *<<in>>* *(p. 271)* **com.rti.dds.domain.DomainParticipant** (p. 629) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

8.63.2.5 abstract void get_default_participant_qos (DomainParticipantQos *qos*) [pure virtual]

Initializes the **com.rti.dds.domain.DomainParticipantQos** (p. 736) 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. 716), or **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos_with_profile** (p. 717), or else, if the call was never made, the default values listed in **com.rti.dds.domain.DomainParticipantQos** (p. 736).

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters:

qos <<*out*>> *(p. 271)* the **domain** (p. 317) participant's QoS Cannot be NULL.

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a **domain** (p. 317) participant factory while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 716)

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 145)
com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 714)

8.63.2.6 abstract void set_default_participant_qos (DomainParticipantQos *qos*) [pure virtual]

Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 736) values for this **domain** (p. 317) participant factory.

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 **domain** (p. 317) participant factory while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 716)

Parameters:

qos <<*inout*>> *(p. 271)* Qos to be filled up. The special value **DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 145) 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. 716) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 145)
com.rti.dds.domain.DomainParticipantFactory.create_participant (p. 714)

8.63.2.7 abstract void set_default_participant_qos_with_profile (String *library_name*, String *profile_name*) [pure virtual]

<<*eXtension*>> *(p. 270)* Sets the default **com.rti.dds.domain.DomainParticipantQos** (p. 736) values for this **domain** (p. 317) 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. 629) if **DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT** (p. 145) is specified as the `qos` parameter when **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 714) is called.

Precondition:

The **com.rti.dds.domain.DomainParticipantQos** (p. 736) contained in

the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a **domain** (p. 317) participant factory while another thread may be simultaneously calling **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 716)

Parameters:

library_name <<***in***>> *(p. 271)* 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. 721)).

profile_name <<***in***>> *(p. 271)* 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. 722)).

If the input profile cannot be found the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 145)
com.rti.dds.domain.DomainParticipantFactory.create_participant_with_profile (p. 730)

8.63.2.8 abstract DomainParticipant lookup_participant (int *domainId*) [pure virtual]

Locates an existing **com.rti.dds.domain.DomainParticipant** (p. 629).

If no such **com.rti.dds.domain.DomainParticipant** (p. 629) exists, the operation will return NULL value.

If multiple **com.rti.dds.domain.DomainParticipant** (p. 629) entities belonging to that domainId exist, then the operation will return one of them. It is not specified which one.

Parameters:

domainId *<<in>>* *(p. 271)* ID of the **domain** (p. 317) participant to lookup.

Returns:

domain (p. 317) participant if it exists, or NULL

8.63.2.9 abstract void get_qos (DomainParticipantFactoryQos *qos*) [pure virtual]

Gets the value for participant factory QoS.

Parameters:

qos *<<inout>>* *(p. 271)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.10 abstract void set_qos (DomainParticipantFactoryQos *qos*) [pure virtual]

Sets the value for a participant factory QoS.

The **com.rti.dds.domain.DomainParticipantFactoryQos.entity_factory** (p. 732) can be changed. The other policies are immutable.

Note that despite having QoS, the **com.rti.dds.domain.DomainParticipantFactory** (p. 708) is not an **com.rti.dds.infrastructure.Entity** (p. 912).

Parameters:

qos *<<in>>* *(p. 271)* Set of policies to be applied to **com.rti.dds.domain.DomainParticipantFactory** (p. 708). 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. 719) Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY if immutable policy is changed, or RETCODE_INCONSISTENT_POLICY if policies are inconsistent

See also:

com.rti.dds.domain.DomainParticipantFactoryQos (p. 732) for rules on consistency among QoS

8.63.2.11 abstract void load_profiles () [pure virtual]

<<*eXtension*>> *(p. 270)* Loads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **com.rti.dds.domain.DomainParticipant** (p. 629) is created or explicitly, after a call to this method.

This has the same effect as **com.rti.dds.domain.DomainParticipantFactory.reload_profiles()** (p. 720).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1247)

8.63.2.12 abstract void reload_profiles () [pure virtual]

<<*eXtension*>> *(p. 270)* Reloads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **com.rti.dds.domain.DomainParticipant** (p. 629) is created or explicitly, after a call to this method.

This has the same effect as **com.rti.dds.domain.DomainParticipantFactory.load_profiles()** (p. 720).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1247)

8.63.2.13 abstract void unload_profiles () [pure virtual]

<<*eXtension*>> *(p. 270)* 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. 104)

See also:

com.rti.dds.infrastructure.ProfileQosPolicy (p. 1247)

8.63.2.14 abstract String get_default_library () [pure virtual]

<<***eXtension***>> *(p. 270)* Gets the default XML library associated with a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Returns:

The default library or null if the default library was not set.

See also:

com.rti.dds.domain.DomainParticipantFactory.set_default_-library (p. 721)

8.63.2.15 abstract void set_default_library (String *library_name*) [pure virtual]

<<***eXtension***>> *(p. 270)* Sets the default XML library for a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Any API requiring a library_name as a parameter can use null to refer to the default library.

See also:

com.rti.dds.domain.DomainParticipantFactory.set_default_profile (p. 722) for more information.

Parameters:

library_name <<***in***>> *(p. 271)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.domain.DomainParticipantFactory.get_default_-library (p. 721)

8.63.2.16 abstract String get_default_profile () [pure virtual]

<<eXtension>> *(p. 270)* Gets the default XML profile associated with a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Returns:

The default profile or null if the default profile was not set.

See also:

com.rti.dds.domain.DomainParticipantFactory.set_default_profile (p. 722)

8.63.2.17 abstract void set_default_profile (String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> *(p. 270)* Sets the default XML profile for a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

This method specifies the profile that will be used as the default the next time a default **DomainParticipantFactory** (p. 708) profile is needed during a call to a **DomainParticipantFactory** (p. 708) method. When calling a **com.rti.dds.domain.DomainParticipantFactory** (p. 708) 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. 717) (you may pass in NULL after having called **set_default_profile()** (p. 722)).

Parameters:

library_name *<<in>>* *(p. 271)* The library name containing the profile.

profile_name *<<in>>* *(p. 271)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.domain.DomainParticipantFactory.get_default_-profile (p. 722)
com.rti.dds.domain.DomainParticipantFactory.get_default_-profile_library (p. 723)

8.63.2.18 abstract String get_default_profile_library () [pure virtual]

<<eXtension>> *(p. 270)* Gets the library where the default XML profile is contained for a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

The default profile library is automatically set when **com.rti.dds.domain.DomainParticipantFactory.set_default_profile** (p. 722) is called.

This library can be different than the **com.rti.dds.domain.DomainParticipantFactory** (p. 708) default library (see **com.rti.dds.domain.DomainParticipantFactory.get_-default_library** (p. 721)).

Returns:

The default profile library or null if the default profile was not set.

See also:

com.rti.dds.domain.DomainParticipantFactory.set_default_profile (p. 722)

8.63.2.19 abstract void get_participant_qos_from_profile (DomainParticipantQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> *(p. 270)* Gets the **com.rti.dds.domain.DomainParticipantQos** (p. 736) values associated with the input XML QoS profile.

Parameters:

qos *<<out>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 721)).

profile_name *<<in>>* *(p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.20 abstract void get_publisher_qos_from_profile (PublisherQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> *(p. 270)* Gets the **com.rti.dds.publication.PublisherQos** (p. 1303) values associated with the input XML QoS profile.

Parameters:

qos *<<out>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 721)).

profile_name *<<in>>* *(p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.21 abstract void get_subscriber_qos_from_profile (SubscriberQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> *(p. 270)* Gets the **com.rti.dds.subscription.SubscriberQos** (p. 1506) values associated with the input XML QoS profile.

Parameters:

qos *<<**out**>> (p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>> (p. 271)* 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. 721)).

profile_name *<<**in**>> (p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.22 abstract void get_datawriter_qos_from_profile (DataWriterQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

*<<**eXtension**>> (p. 270)* Gets the **com.rti.dds.publication.DataWriterQos** (p. 588) values associated with the input XML QoS profile.

Parameters:

qos *<<**out**>> (p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>> (p. 271)* 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. 721)).

profile_name *<<**in**>> (p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.23 abstract void get_datawriter_qos_from_profile_w_topic_name (DataWriterQos *qos*, String *library_name*, String *profile_name*, String *topic_name*) [pure virtual]

<<eXtension>> *(p. 270)* Gets the **com.rti.dds.publication.DataWriterQos** (p. 588) values associated with the input XML QoS profile while applying **topic** (p. 350) filters to the input **topic** (p. 350) name.

Parameters:

qos *<<out>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 721)).

profile_name *<<in>>* *(p. 271)* 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. 722)).

topic_name *<<in>>* *(p. 271)* 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 RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.24 abstract void get_datareader_qos_from_profile (DataReaderQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> *(p. 270)* Gets the **com.rti.dds.subscription.DataReaderQos** (p. 518) values associated with the input XML QoS profile.

Parameters:

qos *<<out>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 721)).

profile_name *<<**in**>>* *(p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.25 abstract void get_datareader_qos_from_profile_w_topic_name (DataReaderQos *qos*, String *library_name*, String *profile_name*, String *topic_name*) [pure virtual]

*<<**eXtension**>>* *(p. 270)* Gets the **com.rti.dds.subscription.DataReaderQos** (p. 518) values associated with the input XML QoS profile while applying **topic** (p. 350) filters to the input **topic** (p. 350) name.

Parameters:

qos *<<**out**>>* *(p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>>* *(p. 271)* 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. 721)).

profile_name *<<**in**>>* *(p. 271)* 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. 722)).

topic_name *<<**in**>>* *(p. 271)* 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 RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.26 abstract void get_topic_qos_from_profile (TopicQos *qos*, String *library_name*, String *profile_name*) [pure virtual]

<<eXtension>> (p. 270) Gets the **com.rti.dds.topic.TopicQos** (p. 1566) values associated with the input XML QoS profile.

Parameters:

qos *<<out>> (p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 271)* 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. 721)).

profile_name *<<in>> (p. 271)* 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. 722)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.27 abstract void get_topic_qos_from_profile_w_topic_name (TopicQos *qos*, String *library_name*, String *profile_name*, String *topic_name*) [pure virtual]

<<eXtension>> (p. 270) Gets the **com.rti.dds.topic.TopicQos** (p. 1566) values associated with the input XML QoS profile while applying **topic** (p. 350) filters to the input **topic** (p. 350) name.

Parameters:

qos *<<out>> (p. 271)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 271)* 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. 721)).

profile_name *<<in>> (p. 271)* 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. 722)).

topic_name *<<**in**>> (p. 271)* 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 RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.63.2.28 abstract void get_qos_profile_libraries (StringSeq *library_names*) [pure virtual]

*<<**eXtension**>> (p. 270)* Gets the names of all XML QoS profile libraries associated with the **com.rti.dds.domain.DomainParticipantFactory** (p. 708)

Parameters:

library_names *<<**out**>> (p. 271)* **com.rti.dds.infrastructure.StringSeq** (p. 1470) to be filled with names of XML QoS profile libraries. Cannot be NULL.

8.63.2.29 abstract void get_qos_profiles (StringSeq *profile_names*, String *library_name*) [pure virtual]

*<<**eXtension**>> (p. 270)* Gets the names of all XML QoS profiles associated with the input XML QoS profile library.

Parameters:

profile_names *<<**out**>> (p. 271)* **com.rti.dds.infrastructure.StringSeq** (p. 1470) to be filled with names of XML QoS profiles. Cannot be NULL.

library_name *<<**in**>> (p. 271)* 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. 721)).

8.63.2.30 abstract DomainParticipant create_participant_with_profile (int *domainId*, String *library_name*, String *profile_name*, DomainParticipantListener *listener*, int *mask*) [pure virtual]

<<*eXtension*>> *(p. 270)* Creates a new **com.rti.dds.domain.DomainParticipant** (p. 629) object using the **com.rti.dds.domain.DomainParticipantQos** (p. 736) associated with the input XML QoS profile.

Precondition:

The **com.rti.dds.domain.DomainParticipantQos** (p. 736) in the input profile must be consistent, or the operation will fail and no **com.rti.dds.domain.DomainParticipant** (p. 629) will be created.

If you want to create multiple participants on a given host in the same **domain** (p. 317), make sure each one has a different participant index (set in the **com.rti.dds.infrastructure.WireProtocolQosPolicy** (p. 1709)). 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** (p. 317) ID).

Note that if there is a single participant per host in a given **domain** (p. 317), the participant index can be left at the default value (-1).

Parameters:

domainId <<*in*>> *(p. 271)* ID of the **domain** (p. 317) that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1396).

library_name <<*in*>> *(p. 271)* 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. 721)).

profile_name <<*in*>> *(p. 271)* 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. 722)).

listener <<*in*>> *(p. 271)* the DomainParticipant's listener.

mask <<*in*>> *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

domain (p. 317) participant or NULL on failure

See also:

Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.domain.DomainParticipantQos (p. 736) for rules on consistency among QoS
DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT (p. 145)
NDDS_DISCOVERY_PEERS (p. 55)
com.rti.dds.domain.DomainParticipantFactory.create_participant() (p. 714)
com.rti.dds.domain.DomainParticipantFactory.get_default_participant_qos() (p. 716)
com.rti.dds.domain.DomainParticipant.set_listener() (p. 682)

8.63.2.31 abstract void unregister_thread () [pure virtual]

<<***eXtension***>> *(p. 270)* 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. 104)

8.64 DomainParticipantFactoryQos Class Reference

QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Inheritance diagram for DomainParticipantFactoryQos::

Public Attributes

- final **EntityFactoryQosPolicy entity_factory**
  Entity factory policy, ***ENTITY_FACTORY*** *(p. 69).*
- final **SystemResourceLimitsQosPolicy resource_limits**
  <<**eXtension**>> (p. 270) *System resource limits,* ***SYSTEM_RESOURCE_LIMITS*** *(p. 111).*
- final **ProfileQosPolicy profile**
  <<**eXtension**>> (p. 270) *Qos profile policy,* ***PROFILE*** *(p. 87).*
- final **LoggingQosPolicy logging**
  <<**eXtension**>> (p. 270) *Logging qos policy,* ***LOGGING*** *(p. 80).*

8.64.1 Detailed Description

QoS policies supported by a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 708)

See also:

QoS Policies (p. 90) and allowed ranges within each Qos.

8.64.2 Member Data Documentation

8.64.2.1 final EntityFactoryQosPolicy entity_factory

Entity factory policy, **ENTITY_FACTORY** (p. 69).

8.64.2.2 final SystemResourceLimitsQosPolicy resource_limits

<<*eXtension*>> *(p. 270)* System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 111).

8.64.2.3 final ProfileQosPolicy profile

<<*eXtension*>> *(p. 270)* Qos profile policy, **PROFILE** (p. 87).

8.64.2.4 final LoggingQosPolicy logging

<<*eXtension*>> *(p. 270)* Logging qos policy, **LOGGING** (p. 80).

8.65 DomainParticipantListener Interface Reference

<<*interface*>> *(p. 271)* Listener for participant status.

Inheritance diagram for DomainParticipantListener::

8.65.1 Detailed Description

<<*interface*>> *(p. 271)* Listener for participant status.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

Status:

Status Kinds (p. 106)

This is the interface that can be implemented by an application-provided class and then registered with the **com.rti.dds.domain.DomainParticipant** (p. 629) such that the application can be notified by RTI Connext of relevant status changes.

The **com.rti.dds.domain.DomainParticipantListener** (p. 734) 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. 734) 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. 628) 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. 628) if one is installed. Otherwise, RTI Connext will notify the Listener attached to the **com.rti.dds.domain.DomainParticipant** (p. 629).

Important: Because a **com.rti.dds.domain.DomainParticipantListener** (p. 734) 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. 734) callback.

Note: Due to a thread-safety issue, the destruction of a **DomainParticipantListener** (p. 734) from an enabled **DomainParticipant** (p. 629) should be avoided – even if the **DomainParticipantListener** (p. 734) has been removed from the **DomainParticipant** (p. 629). (This limitation does not affect the Java API.)

See also:

com.rti.dds.infrastructure.Listener (p. 1154)
com.rti.dds.domain.DomainParticipant.set_listener (p. 682)

8.66 DomainParticipantQos Class Reference

QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 629) entity.

Inheritance diagram for DomainParticipantQos::

Public Attributes

- final **UserDataQosPolicy user_data**

  User data policy, ***USER_DATA*** *(p. 126).*

- final **EntityFactoryQosPolicy entity_factory**

  Entity factory policy, ***ENTITY_FACTORY*** *(p. 69).*

- final **WireProtocolQosPolicy wire_protocol**

  <<**eXtension**>> (p. 270) *Wire Protocol policy,* ***WIRE_PROTOCOL*** *(p. 128).*

- final **TransportBuiltinQosPolicy transport_builtin**

  <<**eXtension**>> (p. 270) *Transport Builtin policy,* ***TRANSPORT_BUILTIN*** *(p. 115).*

- final **TransportUnicastQosPolicy default_unicast**

  <<**eXtension**>> (p. 270) *Default Unicast Transport policy,* ***TRANSPORT_UNICAST*** *(p. 123).*

- final **DiscoveryQosPolicy discovery**

  <<**eXtension**>> (p. 270) *Discovery policy,* ***DISCOVERY*** *(p. 54).*

- final **DomainParticipantResourceLimitsQosPolicy resource_limits**

  <<**eXtension**>> (p. 270) *Domain participant resource limits policy,* ***DOMAIN_PARTICIPANT_RESOURCE_LIMITS*** *(p. 63).*

- final **EventQosPolicy event**

  <<**eXtension**>> (p. 270) *Event policy,* ***EVENT*** *(p. 71).*

- final **ReceiverPoolQosPolicy receiver_pool**

  <<**eXtension**>> (p. 270) *Receiver pool policy,* ***RECEIVER_POOL*** *(p. 100).*

^ final **DatabaseQosPolicy database**

<<**eXtension**>> (p. 270) *Database policy,* ***DATABASE*** *(p. 44).*

^ final **DiscoveryConfigQosPolicy discovery_config**

<<**eXtension**>> (p. 270) *Discovery config policy,* ***DISCOVERY_CONFIG*** *(p. 52).*

^ final **PropertyQosPolicy property**

<<**eXtension**>> (p. 270) *Property policy,* ***PROPERTY*** *(p. 88).*

^ final **EntityNameQosPolicy participant_name**

<<**eXtension**>> (p. 270) *The participant name.* ***ENTITY_NAME*** *(p. 70)*

^ final **TypeSupportQosPolicy type_support**

<<**eXtension**>> (p. 270) *Type support data,* ***TYPESUPPORT*** *(p. 124).*

^ final **TransportMulticastMappingQosPolicy multicast_mapping**

<<**eXtension**>> (p. 270) *The multicast mapping policy. DDSTransportMulticastMappingQosModule*

8.66.1 Detailed Description

QoS policies supported by a **com.rti.dds.domain.DomainParticipant** (p. 629) entity.

Certain members must be set in a consistent manner:

Length of **com.rti.dds.domain.DomainParticipantQos.user_data** (p. 738) .value <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .participant_user_data_max_length

For **com.rti.dds.domain.DomainParticipantQos.discovery_config** (p. 739) .publication_writer

high_watermark <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .local_writer_allocation .max_count heartbeats_per_max_samples <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .local_writer_allocation.max_count

For **com.rti.dds.domain.DomainParticipantQos.discovery_config** (p. 739) .suscription_writer

high_watermark <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .local_reader_allocation.max_count heartbeats_per_max_samples

<= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .local_reader_allocation.max_count

If any of the above are not true, **com.rti.dds.domain.DomainParticipant.set_qos** (p. 677) and **com.rti.dds.domain.DomainParticipant.set_qos_with_profile** (p. 678) and **com.rti.dds.domain.DomainParticipantFactory.set_default_participant_qos** (p. 716) will fail with RETCODE_INCONSISTENT_POLICY, and **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 714) will fail.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

See also:

QoS Policies (p. 90) and allowed ranges within each Qos.
NDDS_DISCOVERY_PEERS (p. 55)

8.66.2 Member Data Documentation

8.66.2.1 final UserDataQosPolicy user_data

User data policy, **USER_DATA** (p. 126).

8.66.2.2 final EntityFactoryQosPolicy entity_factory

Entity factory policy, **ENTITY_FACTORY** (p. 69).

8.66.2.3 final WireProtocolQosPolicy wire_protocol

<<*eXtension*>> *(p. 270)* Wire Protocol policy, **WIRE_PROTOCOL** (p. 128).

The wire protocol (RTPS) attributes associated with the participant.

8.66.2.4 final TransportBuiltinQosPolicy transport_builtin

<<*eXtension*>> *(p. 270)* Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 115).

8.66.2.5 final TransportUnicastQosPolicy default_unicast

<<*eXtension*>> *(p. 270)* Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 123).

8.66.2.6 final DiscoveryQosPolicy discovery

<<*eXtension*>> *(p. 270)* Discovery policy, **DISCOVERY** (p. 54).

8.66.2.7 final DomainParticipantResourceLimitsQosPolicy resource_limits

<<*eXtension*>> *(p. 270)* Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 63).

8.66.2.8 final EventQosPolicy event

<<*eXtension*>> *(p. 270)* Event policy, **EVENT** (p. 71).

8.66.2.9 final ReceiverPoolQosPolicy receiver_pool

<<*eXtension*>> *(p. 270)* Receiver pool policy, **RECEIVER_POOL** (p. 100).

8.66.2.10 final DatabaseQosPolicy database

<<*eXtension*>> *(p. 270)* Database policy, **DATABASE** (p. 44).

8.66.2.11 final DiscoveryConfigQosPolicy discovery_config

<<*eXtension*>> *(p. 270)* Discovery config policy, **DISCOVERY_CONFIG** (p. 52).

8.66.2.12 final PropertyQosPolicy property

<<*eXtension*>> *(p. 270)* Property policy, **PROPERTY** (p. 88).

8.66.2.13 final EntityNameQosPolicy participant_name

<<*eXtension*>> *(p. 270)* The participant name. **ENTITY_NAME** (p. 70)

8.66.2.14 final TypeSupportQosPolicy type_support

<<*eXtension*>> *(p. 270)* Type support data, **TYPESUPPORT** (p. 124).

Optional value that is passed to a type plugin's on_participant_attached function.

8.66.2.15 final TransportMulticastMappingQosPolicy multicast_mapping

<<***eXtension***>> *(p. 270)* The multicast mapping policy. DDSTransportMulticastMappingQosModule

8.67 DomainParticipantResourceLimitsQosPolicy Class Reference

Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) 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*** *(p. 350).*

^ 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. 538)) 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. 736) *and builtin.ParticipantBuiltinTopicData.*

^ int **topic_data_max_length**

*Maximum length of **topic*** (p. 350) *data in **com.rti.dds.topic.TopicQos*** (p. 1566)*, builtin.TopicBuiltinTopicData, builtin.PublicationBuiltinTopicData and builtin.SubscriptionBuiltinTopicData.*

^ int **publisher_group_data_max_length**

Maximum length of group data in ***com.rti.dds.publication.PublisherQos*** *(*p. 1303*) and builtin.PublicationBuiltinTopicData.*

^ int **subscriber_group_data_max_length**

Maximum length of group data in ***com.rti.dds.subscription.SubscriberQos*** *(*p. 1506*) and builtin.SubscriptionBuiltinTopicData.*

^ int **writer_user_data_max_length**

Maximum length of user data in ***com.rti.dds.publication.DataWriterQos*** *(*p. 588*) and builtin.PublicationBuiltinTopicData.*

^ int **reader_user_data_max_length**

Maximum length of user data in ***com.rti.dds.subscription.DataReaderQos*** *(*p. 518*) and builtin.SubscriptionBuiltinTopicData.*

^ int **max_partitions**

Maximum number of partition name strings allowable in a ***com.rti.dds.infrastructure.PartitionQosPolicy*** *(*p. 1233*).*

^ int **max_partition_cumulative_characters**

Maximum number of combined characters allowable in all partition names in a ***com.rti.dds.infrastructure.PartitionQosPolicy*** *(*p. 1233*).*

^ int **type_code_max_serialized_length**

Maximum size of serialized string for type code.

^ int **contentfilter_property_max_length**

This field is the maximum length of all data related to a Content-filtered ***topic*** *(*p. 350*).*

^ int **channel_seq_max_length**

Maximum number of channels that can be specified in ***com.rti.dds.infrastructure.MultiChannelQosPolicy*** *(*p. 1205*) for MultiChannel DataWriters.*

^ int **channel_filter_expression_max_length**

Maximum length of a channel ***com.rti.dds.infrastructure.ChannelSettings_t.filter_expression*** *(*p. 442*) in a MultiChannel DataWriter.*

^ int **participant_property_list_max_length**

Maximum number of properties associated with the ***com.rti.dds.domain.DomainParticipant*** *(*p. 629*).*

- int **participant_property_string_max_length**

  *Maximum string length of the properties associated with the **com.rti.dds.domain.DomainParticipant** (p. 629).*

- int **writer_property_list_max_length**

  *Maximum number of properties associated with a **com.rti.dds.publication.DataWriter** (p. 538).*

- int **writer_property_string_max_length**

  *Maximum string length of the properties associated with a **com.rti.dds.publication.DataWriter** (p. 538).*

- int **reader_property_list_max_length**

  *Maximum number of properties associated with a **com.rti.dds.subscription.DataReader** (p. 473).*

- int **reader_property_string_max_length**

  *Maximum string length of the properties associated with a **com.rti.dds.subscription.DataReader** (p. 473).*

- int **max_endpoint_groups**

  *Maximum number of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909) allowable in a **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 392).*

- int **max_endpoint_group_cumulative_characters**

  *Maximum number of combined role_name characters allowable in all **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909) in a **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 392).*

8.67.1 Detailed Description

Various settings that configure how a **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) is allowed to dynamically allocate memory as needed as users create local Entities such as **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) objects or as the participant discovers new applications. By setting fixed values for the maximum parameters in this QoS policy, you can bound the memory that can be allocated by a **com.rti.dds.domain.DomainParticipant** (p. 629). In addition, by setting the initial values to the maximum values, you can prevent DomainParticipants from allocating memory after the initialization period.

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. 1233), the maximum length of data store in the **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680) and **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1064), and many others can be changed from their defaults using this QoS policy. However, it is important that all DomainParticipants that need to communicate with each other use the *same set* of maximum values. Otherwise, when these parameters are propagated from one **com.rti.dds.domain.DomainParticipant** (p. 629) to another, a **com.rti.dds.domain.DomainParticipant** (p. 629) 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.DomainParticipantResourceLimitsQosPolicy.type_code_max_serialized_length** (p. 755).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.67.2 Member Data Documentation

8.67.2.1 final AllocationSettings_t local_writer_allocation

Allocation settings applied to local DataWriters.

[**default**] initial_count = 16; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.2 final AllocationSettings_t local_reader_allocation

Allocation settings applied to local DataReaders.

[**default**] initial_count = 16; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.3 final AllocationSettings_t local_publisher_allocation

Allocation settings applied to local Publisher.

[**default**] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.4 final AllocationSettings_t local_subscriber_allocation

Allocation settings applied to local Subscriber.

[**default**] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.5 final AllocationSettings_t local_topic_allocation

Allocation settings applied to local Topic.

[**default**] initial_count = 16; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.6 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 = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.7 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 = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.8 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 = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.9 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 = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.10 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 = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.11 final AllocationSettings_t ignored_entity_allocation

Allocation settings applied to ignored entities.

[**default**] initial_count = 8; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.12 final AllocationSettings_t content_filtered_topic_allocation

Allocation settings applied to content filtered **topic** (p. 350).

[**default**] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.13 final AllocationSettings_t content_filter_allocation

Allocation settings applied to content filter.

[**default**] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102); incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.14 final AllocationSettings_t read_condition_allocation

Allocation settings applied to read condition pool.

[**default**] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), incremental_count = -1

[**range**] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.15 final AllocationSettings_t query_condition_allocation

Allocation settings applied to query condition pool.

[default] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.16 final AllocationSettings_t outstanding_asynchronous_sample_allocation

Allocation settings applied to the maximum number of samples (from all **com.rti.dds.publication.DataWriter** (p. 538)) waiting to be asynchronously written.

[default] initial_count = 64; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.17 final AllocationSettings_t flow_controller_allocation

Allocation settings applied to flow controllers.

[default] initial_count = 4; max_count = **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), incremental_count = -1

[range] See allowed ranges in struct **com.rti.dds.infrastructure.AllocationSettings_t** (p. 385)

8.67.2.18 int local_writer_hash_buckets

Hash_Buckets settings applied to local DataWriters.

[default] 4

[range] [1, 10000]

8.67.2.19 int local_reader_hash_buckets

Number of hash buckets for local DataReaders.

[default] 4

[range] [1, 10000]

8.67.2.20 int local_publisher_hash_buckets

Number of hash buckets for local Publisher.

[default] 1

[range] [1, 10000]

8.67.2.21 int local_subscriber_hash_buckets

Number of hash buckets for local Subscriber.

[default] 1

[range] [1, 10000]

8.67.2.22 int local_topic_hash_buckets

Number of hash buckets for local Topic.

[default] 4

[range] [1, 10000]

8.67.2.23 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]

8.67.2.24 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]

8.67.2.25 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]

8.67.2.26 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]

8.67.2.27 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]

8.67.2.28 int ignored_entity_hash_buckets

Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

8.67.2.29 int content_filtered_topic_hash_buckets

Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

8.67.2.30 int content_filter_hash_buckets

Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

8.67.2.31 int flow_controller_hash_buckets

Number of hash buckets for flow controllers.

[**default**] 1

[**range**] [1, 10000]

8.67.2.32 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**] 8

[**range**] [4, 1 million]

8.67.2.33 int participant_user_data_max_length

Maximum length of user data in **com.rti.dds.domain.DomainParticipantQos** (p. 736) and builtin.ParticipantBuiltinTopicData.

[**default**] 256

[**range**] [0,0x7fffffff]

8.67.2.34 int topic_data_max_length

Maximum length of **topic** (p. 350) data in **com.rti.dds.topic.TopicQos** (p. 1566), builtin.TopicBuiltinTopicData, builtin.PublicationBuiltinTopicData and builtin.SubscriptionBuiltinTopicData.

[**default**] 256

[**range**] [0,0x7fffffff]

8.67.2.35 int publisher_group_data_max_length

Maximum length of group data in **com.rti.dds.publication.PublisherQos** (p. 1303) and builtin.PublicationBuiltinTopicData.

[**default**] 256

[**range**] [0,0x7fffffff]

8.67.2.36 int subscriber_group_data_max_length

Maximum length of group data in **com.rti.dds.subscription.SubscriberQos** (p. 1506) and builtin.SubscriptionBuiltinTopicData.

[default] 256

[range] [0,0x7fffffff]

8.67.2.37 int writer_user_data_max_length

Maximum length of user data in **com.rti.dds.publication.DataWriterQos** (p. 588) and builtin.PublicationBuiltinTopicData.

[default] 256

[range] [0,0x7fffffff]

8.67.2.38 int reader_user_data_max_length

Maximum length of user data in **com.rti.dds.subscription.DataReaderQos** (p. 518) and builtin.SubscriptionBuiltinTopicData.

[default] 256

[range] [0,0x7fffffff]

8.67.2.39 int max_partitions

Maximum number of partition name strings allowable in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1233).

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]

8.67.2.40 int max_partition_cumulative_characters

Maximum number of combined characters allowable in all partition names in a **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1233).

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]

8.67.2.41 int type_code_max_serialized_length

Maximum size of serialized string for type code.

This parameter limits the size of the type code that a **com.rti.dds.domain.DomainParticipant** (p. 629) 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 type codes larger than the default maximum of 2048 bytes. So it is common for users to set this parameter to a larger value. However, as with all parameters in this QoS policy defining maximum sizes for variable-length elements, all DomainParticipants in the same **domain** (p. 317) should use the same value for this parameter.

[**default**] 2048

[**range**] [0,0xffff]

8.67.2.42 int contentfilter_property_max_length

This field is the maximum length of all data related to a Content-filtered **topic** (p. 350).

This is the sum of the length of the content filter name, the length of the related **topic** (p. 350) 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]

8.67.2.43 int channel_seq_max_length

Maximum number of channels that can be specified in **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) for MultiChannel DataWriters.

[default] 32

[range] [0,0xffff]

8.67.2.44 int channel_filter_expression_max_length

Maximum length of a channel **com.rti.dds.infrastructure.ChannelSettings_t.filter_expression** (p. 442) in a MultiChannel DataWriter.

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

8.67.2.45 int participant_property_list_max_length

Maximum number of properties associated with the **com.rti.dds.domain.DomainParticipant** (p. 629).

[default] 32

[range] [0,0xffff]

8.67.2.46 int participant_property_string_max_length

Maximum string length of the properties associated with the **com.rti.dds.domain.DomainParticipant** (p. 629).

The string length is defined as the cumulative length in bytes of all the pair (name,value) associated with the **com.rti.dds.domain.DomainParticipant** (p. 629) properties.

[default] 1024

[range] [0,0xffff]

8.67.2.47 int writer_property_list_max_length

Maximum number of properties associated with a **com.rti.dds.publication.DataWriter** (p. 538).

[range] [0,0xffff]

[**default**] 32

8.67.2.48 int writer_property_string_max_length

Maximum string length of the properties associated with a **com.rti.dds.publication.DataWriter** (p. 538).

The string length is defined as the cumulative length in bytes of all the pair (name,value) associated with the data writer properties.

[**default**] 1024

[**range**] [0,0xffff]

8.67.2.49 int reader_property_list_max_length

Maximum number of properties associated with a **com.rti.dds.subscription.DataReader** (p. 473).

[**default**] 32

[**range**] [0,0xffff]

8.67.2.50 int reader_property_string_max_length

Maximum string length of the properties associated with a **com.rti.dds.subscription.DataReader** (p. 473).

The string length is defined as the cumulative length in bytes of all the pair (name,value) associated with a **com.rti.dds.subscription.DataReader** (p. 473) properties.

[**default**] 1024

[**range**] [0,0xffff]

8.67.2.51 int max_endpoint_groups

Maximum number of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909) allowable in a **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 392).

[**default**] 32

[**range**] [0,65535]

8.67.2.52 int max_endpoint_group_cumulative_characters

Maximum number of combined role_name characters allowable in all **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909) in a **com.rti.dds.infrastructure.AvailabilityQosPolicy** (p. 392).

The maximum number of combined characters should account for a terminating NULL character for each role_name string.

[default] 1024

[range] [0,65535]

8.68 DoubleSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 761) that returns a java.lang.Double.*

- final Object **set** (int index, Object element)

  *A wrapper for **setDouble()** (p. 762).*

- final void **add** (int index, Object element)

  A wrapper for addDouble(int, int).

8.68.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < double >.

Instantiates:

<<***generic***>> (p. 271) **com.rti.dds.util.Sequence** (p. 1432)

See also:

double
com.rti.dds.util.Sequence (p. 1432)

8.68.2 Constructor & Destructor Documentation

8.68.2.1 DoubleSeq ()

Constructs an empty sequence of doubles with an initial maximum of zero.

8.68.2.2 DoubleSeq (int *initialMaximum*)

Constructs an empty sequence of doubles with the given initial maximum.

8.68.2.3 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

8.68.3 Member Function Documentation

8.68.3.1 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.

8.68.3.2 final boolean addAllDouble (double[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.68.3.3 final void addDouble (double *element*)

Append the element to the end of the sequence.

8.68.3.4 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.

8.68.3.5 final double getDouble (int *index*)

Returns the double at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.68.3.6 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.

8.68.3.7 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.

8.68.3.8 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.

8.68.3.9 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. 764), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.68.3.10 final Object get (int *index*) [virtual]

A wrapper for **getDouble(int)** (p. 761) that returns a java.lang.Double.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.68.3.11 final Object set (int *index*, Object *element*) [virtual]

A wrapper for **setDouble()** (p. 762).

Exceptions:

ClassCastException if the element is not of type Double.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.68.3.12 final void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.69 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. 473) entities that join the network later.

Inheritance diagram for DurabilityQosPolicy::

Public Attributes

- **DurabilityQosPolicyKind kind**

  The kind of durability.

- boolean **direct_communication**

  <<**eXtension**>> (p. 270) *Indicates whether or not a TRANSIENT or PERSISTENT* ***com.rti.dds.subscription.DataReader*** *(p. 473) should receive samples directly from a TRANSIENT or PERSISTENT* ***com.rti.dds.publication.DataWriter*** *(p. 538)*

8.69.1 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. 473) entities that join the network later.

Entity:

com.rti.dds.topic.Topic (p. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

See also:

DURABILITY_SERVICE (p. 66)

8.69.2 Usage

It is possible for a **com.rti.dds.publication.DataWriter** (p. 538) to start publishing data before all (or any) **com.rti.dds.subscription.DataReader** (p. 473) entities have joined the network.

Moreover, a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 65) policy is set to **DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS** (p. 771).

8.69.2.1 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. 768) of **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771) or **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772) there is a corresponding "built-in" **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.publication.DataWriter** (p. 538) 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. 538), there is a built-in durable **com.rti.dds.subscription.DataReader** (p. 473) 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.DataWriter** (p. 538) and

com.rti.dds.subscription.DataReader (p. 473) entities. For each transient or persistent **com.rti.dds.topic.Topic** (p. 1545), the built-in fictitious Persistence Service **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.publication.DataWriter** (p. 538) have their QoS configured from the QoS of your application's **com.rti.dds.publication.DataWriter** (p. 538) and **com.rti.dds.subscription.DataReader** (p. 473) entities that communicate on that **com.rti.dds.topic.Topic** (p. 1545).

For a given **com.rti.dds.topic.Topic** (p. 1545), the usual request/offered semantics apply to the matching between any **com.rti.dds.publication.DataWriter** (p. 538) in the **domain** (p. 317) that writes the **com.rti.dds.topic.Topic** (p. 1545) and the built-in transient/persistent **com.rti.dds.subscription.DataReader** (p. 473) for that **com.rti.dds.topic.Topic** (p. 1545); similarly for the built-in transient/persistent **com.rti.dds.publication.DataWriter** (p. 538) for a **com.rti.dds.topic.Topic** (p. 1545) and any **com.rti.dds.subscription.DataReader** (p. 473) for the **com.rti.dds.topic.Topic** (p. 1545). As a consequence, a **com.rti.dds.publication.DataWriter** (p. 538) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **com.rti.dds.subscription.DataReader** (p. 473) that has an incompatible QoS will not get data from it.

Incompatibilities between local **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.publication.DataWriter** (p. 538) entities and the corresponding fictitious built-in transient/persistent entities cause the **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459) and **StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1459) to change and the corresponding **Listener** (p. 1154) invocations and/or signaling of **com.rti.dds.infrastructure.Condition** (p. 451) objects as they would with your application's own entities.

The value of **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 774) 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

2. While in the NOT_ALIVE_DISPOSED state, the system detects that there are no more 'live' **com.rti.dds.publication.DataWriter** (p. 538) entities writing the instance. That is, all existing writers either unregister the instance (call unregister) or lose their liveliness,

and

3. A time interval longer that **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 774) has elapsed since the moment RTI Connext detected that the previous two conditions were met.

The utility of **com.rti.dds.infrastructure.DurabilityServiceQosPolicy.service_cleanup_delay** (p. 774) 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.

8.69.3 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 DURABILITY kind are considered ordered such that **DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS** (p. 771) < **DurabilityQosPolicyKind.TRANSIENT_LOCAL_DURABILITY_QOS** (p. 771) < **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771) < **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772).

8.69.4 Member Data Documentation

8.69.4.1 DurabilityQosPolicyKind kind

The kind of durability.

[default] **DurabilityQosPolicyKind.VOLATILE_DURABILITY_QOS** (p. 771)

8.69.4.2 boolean direct_communication

<<***eXtension***>> *(p. 270)* Indicates whether or not a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 473) should receive samples directly from a TRANSIENT or PERSISTENT **com.rti.dds.publication.DataWriter** (p. 538)

When direct_communication is set to true, a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 473) will receive samples from both the original **com.rti.dds.publication.DataWriter** (p. 538) configured with TRANSIENT or PERSISTENT durability and the

com.rti.dds.publication.DataWriter (p. 538) 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. 538) and the persistence service, the middleware will discard the duplicate.

When direct_communication is set to false, a TRANSIENT or PERSISTENT **com.rti.dds.subscription.DataReader** (p. 473) will only receive samples from the **com.rti.dds.publication.DataWriter** (p. 538) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[**default**] true

8.70 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. 473*) that is unknown by the* ***com.rti.dds.publication.DataWriter*** *(*p. 538*) 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. 473*).*

- 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. 473*).*

- static final **DurabilityQosPolicyKind PERSISTENT_DURABILITY_QOS**

  Data is kept on permanent storage, so that they can outlive a system session.

8.70.1 Detailed Description

Kinds of durability.

QoS:

com.rti.dds.infrastructure.DurabilityQosPolicy (p. 765)

8.70.2 Member Data Documentation

8.70.2.1 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. 473) that is unknown by the **com.rti.dds.publication.DataWriter** (p. 538) at the time the instance is written.

In other words, RTI Connext will only attempt to provide the data to existing subscribers.

8.70.2.2 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. 473).

Which particular samples are kept depends on other QoS such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356). RTI Connext is only required to keep the data in memory of the **com.rti.dds.publication.DataWriter** (p. 538) that wrote the data.

Data is not required to survive the **com.rti.dds.publication.DataWriter** (p. 538).

For this setting to be effective, you must also set the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) to **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341).

8.70.2.3 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. 473).

Which particular samples are kept depends on other QoS such as **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356). 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. 538).

Data will survive the **com.rti.dds.publication.DataWriter** (p. 538).

8.70.2.4 final DurabilityQosPolicyKind PERSISTENT_DURABILITY_QOS [static]

Data is kept on permanent storage, so that they can outlive a system session.

8.71 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. 765) setting of **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772) or **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771).

Inheritance diagram for DurabilityServiceQosPolicy::

Public Attributes

- final **Duration_t service_cleanup_delay**

  *[**Not supported (optional)**] 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**

  Part of history QoS policy to apply when feeding a late joiner.

- 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.

8.71.1 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. 765) setting of **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772) or **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771).

Entity:

com.rti.dds.topic.Topic (p. 1545), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

See also:

DURABILITY (p. 65)
HISTORY (p. 75)
RESOURCE_LIMITS (p. 102)

8.71.2 Usage

When a DataWriter's **com.rti.dds.infrastructure.DurabilityQosPolicy.kind** (p. 768) is **DurabilityQosPolicyKind.PERSISTENT_DURABILITY_QOS** (p. 772) or **DurabilityQosPolicyKind.TRANSIENT_DURABILITY_QOS** (p. 771), 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. 538) to **com.rti.dds.subscription.DataReader** (p. 473) 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. 538), such as how much data to store. For example, it configures the **HISTORY** (p. 75) and the **RESOURCE_LIMITS** (p. 102) 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.

8.71.3 Member Data Documentation

8.71.3.1 final Duration_t service_cleanup_delay

Initial value:

```
new Duration_t(
        Duration_t.DURATION_INFINITE_SEC, Duration_t.DURATION_INFINITE_NSEC)
```

[**Not supported (optional)**] Controls when the service is able to remove all information regarding a data instances.

[**default**] 0

8.71.3.2 HistoryQosPolicyKind history_kind

The kind of history to apply in recouping durable data.

[default] **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075)

8.71.3.3 int history_depth

Part of history QoS policy to apply when feeding a late joiner.

[default] 1

8.71.3.4 int max_samples

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.71.3.5 int max_instances

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.71.3.6 int max_samples_per_instance

Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.72 Duration_t Class Reference

Type for *duration* representation.

Inherits Struct, and java.io.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** ()

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.

ˆ static final int **DURATION_AUTO_NSEC**

An auto nano-second period of time.

8.72.1 Detailed Description

Type for *duration* representation.

Represents a time interval.

8.72.2 Constructor & Destructor Documentation

8.72.2.1 Duration_t (Duration_t *duration*)

Copy constructor.

Parameters:

duration The duration instance to copy. It must not be null.

8.72.2.2 Duration_t (int *sec*, int *nanosec*)

Parameters:

sec must be >=0

nanosec must be >=0

Exceptions:

RETCODE_BAD_PARAMETER *(*p. *1363**)* if either value is negative

8.72.3 Member Function Documentation

8.72.3.1 boolean is_zero ()

Returns:

true if the given duration is of zero length.

8.72.3.2 boolean is_infinite ()

Returns:

true if the given duration is of infinite length.

8.72.3.3 boolean is_auto ()

Returns:

true if the given duration has auto value.

8.72.4 Member Data Documentation

8.72.4.1 final int DURATION_ZERO_SEC [static]

A zero-length second period of time.

8.72.4.2 final int DURATION_ZERO_NSEC [static]

A zero-length nano-second period of time.

8.72.4.3 final int DURATION_INFINITE_SEC [static]

An infinite second period of time.

8.72.4.4 final int DURATION_INFINITE_NSEC [static]

An infinite nano-second period of time.

8.72.4.5 final int DURATION_AUTO_SEC [static]

An auto second period of time.

8.72.4.6 final int DURATION_AUTO_NSEC [static]

An auto nano-second period of time.

8.72.4.7 int sec

seconds

8.72.4.8 int nanosec

nanoseconds

8.73 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. 780) sample.*

- boolean **equals** (Object o)

  Indicate whether the contents of another ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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, including key members.

- void **clear_nonkey_members** ()

  Clear the contents of all data members of this object, not including key members.

- void **clear_member** (String member_name, int member_id)

  Clear the contents of a single 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. 780).*

- void **bind_type** (**TypeCode** type)

  If this ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) object is not yet associated with a data type, set that type now to the given TypeCode.*

- void **unbind_type** ()

  Dissociate this ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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. 780) object to provide access to a complex field of this* ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) object.*

- void **unbind_complex_member** (**DynamicData** value)

  Tear down the association created by a ***com.rti.dds.dynamicdata.DynamicData.bind_complex_member*** *(p. 800) operation, committing any changes to the outer object since then.*

- **TypeCode get_type** ()

  Get the data type, of which this ***com.rti.dds.dynamicdata.DynamicData*** *(p. 780) 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 this sample.

- boolean **member_exists** (String member_name, int member_id)

  Indicates whether a member of a particular name/ID exists in this data 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. 780) 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. 780) 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 int or another type implicitly convertible to it (byte, char, short, short, or ***com.rti.dds.util.Enum*** *(*p. 925*)).*

- int **get_int_array** (int[] array, String member_name, int member_id)

  Get a copy of the given array member.

- void **get_int_seq** (**IntSeq** 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_short_array** (short[] 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.

- 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.

- int **get_byte_array** (byte[] 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.

- 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, short, int, long, or **com.rti.dds.util.Enum** (p. 925)).*

- int **get_long_array** (long[] 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.

- String **get_string** (String member_name, int member_id)

  Get the value of the given field, which is of type 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 int.

- void **set_int_array** (String member_name, int member_id, int[] array)

  Set the contents of the given array member.

- void **set_int_seq** (String member_name, int member_id, **IntSeq** 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_short_array** (String member_name, int member_id, short[] 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_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_byte_array** (String member_name, int member_id, byte[] 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_long** (String member_name, int member_id, long value)

  Set the value of the given field, which is of type long.

- void **set_long_array** (String member_name, int member_id, long[] 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_string** (String member_name, int member_id, String value)

  Set the value of the given field of type 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. 780) object into a member of this object.*

- **DynamicData** (**TypeCode** type, **DynamicDataProperty_t** property)

  *The constructor for new **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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. 849).*

8.73.1 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. 780) represent corresponding objects of the type identified by their TypeCode. 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. 780) objects can represent any complex data type, including those of type kinds **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529), **TCKind.TK_VALUE** (p. 1530), and **TCKind.TK_SPARSE** (p. 1530). 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.

8.73.2 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. When you define a sparse type, you will typically choose both explicitly. 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 MyType {
     short theShort;
     long theLong;
 };
```

IDs work the same way for **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects representing arrays and sequences, since the elements of these collections have no explicit IDs: the ID is one more than the index. (The first element is ID 1, the second is 2, etc.) Array and sequence elements do not have names.

Multi-dimensional arrays are effectively flattened by the **com.rti.dds.dynamicdata.DynamicData** (p. 780) API. 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.

For unions (**TCKind.TK_UNION** (p. 1529)), the ID of a member is the discriminator value corresponding to that member.

8.73.3 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.

8.73.3.1 Lifecycle and Utility Methods

Managing the lifecycle of **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects is simple. You have two choices:

1. Usually, you will go through a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) factory object, which will ensure that the type and property information for the new **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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. 780) 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.

You can also copy **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects:

- com.rti.dds.dynamicdata.DynamicData.copy

You can test them for equality:

- **DynamicData.equals** (p. 795)

com.rti.dds.dynamicdata.DynamicDataTypeSupport (p. 887)	**com.rti.dds.dynamicdata.DynamicData** (p. 780)
com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data (p. 891)	DynamicData.DynamicData

Table 8.1: Lifecycle

And you can print their contents:

- **com.rti.dds.dynamicdata.DynamicData.print** (p. 797)
- **com.rti.dds.dynamicdata.DynamicDataTypeSupport.print_data** (p. 891)

8.73.3.2 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 integer 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.

Type	IDL	C, C++	C#	Java
16-bit integer	short	DDS_Short	short	short
32-bit integer	long	DDS_Long	int	int
64-bit integer	long long	DDS_LongLong	long	long

Table 8.2: Integer Type Names Across Languages

When working with a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object representing an array or sequence, calling one of the "get" methods below for an index that is out of bounds will result in RETCODE_NO_DATA. Calling "set" for an index that is past the end of a sequence will cause that sequence to automatically lengthen (filling with default contents).

In addition to getting or setting a field, you can "clear" its value; that is, set it to a default zero value.

Get	Set
DynamicData.get_int (p. 808)	**DynamicData.set_int** (p. 825)
com.rti.dds.dynamicdata.DynamicData.get_short (p. 810)	**com.rti.dds.dynamicdata.DynamicData.set_short** (p. 827)
DynamicData.get_long (p. 822)	**DynamicData.set_long** (p. 839)
com.rti.dds.dynamicdata.DynamicData.get_float (p. 812)	**com.rti.dds.dynamicdata.DynamicData.set_float** (p. 829)
com.rti.dds.dynamicdata.DynamicData.get_double (p. 814)	**com.rti.dds.dynamicdata.DynamicData.set_double** (p. 831)
com.rti.dds.dynamicdata.DynamicData.get_boolean (p. 816)	**com.rti.dds.dynamicdata.DynamicData.set_boolean** (p. 833)
DynamicData.get_byte (p. 820)	**DynamicData.set_byte** (p. 837)
com.rti.dds.dynamicdata.DynamicData.get_char (p. 818)	**com.rti.dds.dynamicdata.DynamicData.set_char** (p. 835)
com.rti.dds.dynamicdata.DynamicData.get_string (p. 824)	**com.rti.dds.dynamicdata.DynamicData.set_string** (p. 841)

Table 8.3: Basic Types

- **com.rti.dds.dynamicdata.DynamicData.clear_member** (p. 797)
- **com.rti.dds.dynamicdata.DynamicData.clear_all_members** (p. 796)
- **com.rti.dds.dynamicdata.DynamicData.clear_nonkey_members** (p. 796)

8.73.3.3 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.

Get	Set
com.rti.dds.dynamicdata.DynamicData.get_-complex_member (p. 824)	**com.rti.dds.dynamicdata.DynamicData.set_-complex_member** (p. 842)

Table 8.4: Structures, Arrays, and Other Complex Types

- **com.rti.dds.dynamicdata.DynamicData.get_type** (p. 802)
- **com.rti.dds.dynamicdata.DynamicData.get_type_kind** (p. 803)
- **com.rti.dds.dynamicdata.DynamicData.get_member_type** (p. 806)
- **com.rti.dds.dynamicdata.DynamicData.get_member_info** (p. 805)
- **com.rti.dds.dynamicdata.DynamicData.get_member_count** (p. 803)
- **com.rti.dds.dynamicdata.DynamicData.get_member_info_by_-index** (p. 805)
- **com.rti.dds.dynamicdata.DynamicData.member_exists** (p. 803)
- **com.rti.dds.dynamicdata.DynamicData.member_exists_in_type** (p. 804)
- **com.rti.dds.dynamicdata.DynamicData.is_member_key** (p. 807)

8.73.3.4 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. 780) 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. 780) object as a view into an enclosing **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

- **com.rti.dds.dynamicdata.DynamicData.bind_type** (p. 798)
- **com.rti.dds.dynamicdata.DynamicData.unbind_type** (p. 799)
- **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 800)
- **com.rti.dds.dynamicdata.DynamicData.unbind_complex_-member** (p. 802)

Get	Set
DynamicData.get_int_array (p. 808)	**DynamicData.set_int_array** (p. 826)
com.rti.dds.dynamicdata.DynamicData.get_short_array (p. 810)	**com.rti.dds.dynamicdata.DynamicData.set_short_array** (p. 828)
DynamicData.get_long_array (p. 822)	**DynamicData.set_long_array** (p. 840)
com.rti.dds.dynamicdata.DynamicData.get_float_array (p. 812)	**com.rti.dds.dynamicdata.DynamicData.set_float_array** (p. 830)
com.rti.dds.dynamicdata.DynamicData.get_double_array (p. 814)	**com.rti.dds.dynamicdata.DynamicData.set_double_array** (p. 832)
com.rti.dds.dynamicdata.DynamicData.get_boolean_array (p. 816)	**com.rti.dds.dynamicdata.DynamicData.set_boolean** (p. 833)
DynamicData.get_byte_array (p. 820)	**DynamicData.set_byte_array** (p. 838)
com.rti.dds.dynamicdata.DynamicData.get_char_array (p. 818)	**com.rti.dds.dynamicdata.DynamicData.set_char_array** (p. 836)

Table 8.5: Arrays of Basic Types

8.73.3.5 Keys

Keys can be specified in dynamically defined types just as they can in types defined in generated code. However, there are some minor restrictions when *sparse value types* are involved (see **TCKind.TK_SPARSE** (p. 1530)).

- If a type has a member that is of a sparse value type, that member cannot be a key for the enclosing type.

- Sparse value types themselves may have at most a single key field. That field may itself be of any type.

Get	Set
DynamicData.get_int_seq (p. 809)	**DynamicData.set_int_seq** (p. 826)
com.rti.dds.dynamicdata.DynamicData.get_short_seq (p. 811)	**com.rti.dds.dynamicdata.DynamicData.set_short_seq** (p. 828)
DynamicData.get_long_seq (p. 823)	**DynamicData.set_long_seq** (p. 840)
com.rti.dds.dynamicdata.DynamicData.get_float_seq (p. 813)	**com.rti.dds.dynamicdata.DynamicData.set_float_seq** (p. 830)
com.rti.dds.dynamicdata.DynamicData.get_double_seq (p. 815)	**com.rti.dds.dynamicdata.DynamicData.set_double_seq** (p. 832)
com.rti.dds.dynamicdata.DynamicData.get_boolean_seq (p. 817)	**com.rti.dds.dynamicdata.DynamicData.set_boolean_seq** (p. 834)
DynamicData.get_byte_seq (p. 821)	**DynamicData.set_byte_seq** (p. 838)
com.rti.dds.dynamicdata.DynamicData.get_char_seq (p. 819)	**com.rti.dds.dynamicdata.DynamicData.set_char_seq** (p. 836)

Table 8.6: Sequences of Basic Types

8.73.4 Performance

Due to the way in which **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects manage their internal state, it is typically more efficient, when setting the field values of a **com.rti.dds.dynamicdata.DynamicData** (p. 780) for the first time, to do so in the declared order of those fields.

For example, suppose a type definition like the following:

```
struct MyType {
     float my_float;
     sequence<octet> my_bytes;
     short my_short;
 };
```

The richness of the type system makes it difficult to fully characterize the performance differences between all access patterns. Nevertheless, the following are

generally true:

- It will be most performant to set the value of `my_float`, then `my_bytes`, and finally `my_short`.
- The order of modification has a greater impact for types of kind **TCKind.TK_STRUCT** (p. 1529) and **TCKind.TK_VALUE** (p. 1530) than it does for types of kind **TCKind.TK_SPARSE** (p. 1530).
- Modifications to variable-sized types (i.e. those containing strings, sequences, unions, or optional members) are more expensive than modifications to fixed-size types.

MT Safety:

UNSAFE. In general, using a single **com.rti.dds.dynamicdata.DynamicData** (p. 780) object concurrently from multiple threads is *unsafe*.

8.73.5 Constructor & Destructor Documentation

8.73.5.1 DynamicData (TypeCode *type*, DynamicDataProperty_t *property*)

The constructor for new **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

The type parameter may be null. In that case, this **com.rti.dds.dynamicdata.DynamicData** (p. 780) must be *bound* with **com.rti.dds.dynamicdata.DynamicData.bind_type** (p. 798) or **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 800) before it can be used.

If the TypeCode is not null, the newly constructed **com.rti.dds.dynamicdata.DynamicData** (p. 780) object will retain a reference to it. It is *not* safe to delete the TypeCode until all samples that use it have themselves been deleted. You have two options:

- Keep a reference to the TypeCode object yourself, and delete it with **TypeCodeFactory.delete_tc** (p. 1650) after you've deleted all of the objects that use it.
- Do not keep a reference to the TypeCode. 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. 891), and the TypeCode 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. 887) object maintains the TypeCode used by the samples it creates.

Parameters:

type *<<in>>* *(p. 271)* The type of which the new object will represent an object.

property *<<in>>* *(p. 271)* 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. 891)

8.73.6 Member Function Documentation

8.73.6.1 void delete ()

Finalize and deallocate this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.

MT Safety:

UNSAFE.

See also:

DynamicData.DynamicData

8.73.6.2 boolean equals (Object *o*)

Indicate whether the contents of another **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample are the same as those of this one.

This operation compares the data and type of existing members. The types of non-instantiated members may differ in sparse types.

MT Safety:

UNSAFE.

See also:

`http://java.sun.com/javase/6/docs/api/java/lang/Object.html#equals(java.lang.Object`
TCKind.TK_SPARSE (p. 1530)

8.73.6.3 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. 104) or RETCODE_OUT_OF_RESOURCES

Implements **Copyable** (p. 466).

8.73.6.4 void clear_all_members ()

Clear the contents of all data members of this object, including key members.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.clear_nonkey_members (p. 796)
com.rti.dds.dynamicdata.DynamicData.clear_member (p. 797)

8.73.6.5 void clear_nonkey_members ()

Clear the contents of all data members of this object, not including key members.

This method is only applicable to sparse value types.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

TCKind.TK_SPARSE (p. 1530)
com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 796)
com.rti.dds.dynamicdata.DynamicData.clear_member (p. 797)

8.73.6.6 void clear_member (String *member_name*, int *member_id*)

Clear the contents of a single data member of this object.

This method is only applicable to sparse value types.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

TCKind.TK_SPARSE (p. 1530)
com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 796)
com.rti.dds.dynamicdata.DynamicData.clear_nonkey_members (p. 796)

8.73.6.7 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. 891).

Warning: This operation may not display any data for members at the end of a data structure that have not been explicitly set before the data sample is serialized. This will not be a problem on a received data sample, which should always correctly display all members.

MT Safety:

UNSAFE.

Parameters:

fp *<<**in**>>* *(p. 271)* The file to which the object should be printed.

indent *<<**in**>>* *(p. 271)* 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. 104)

See also:

com.rti.dds.dynamicdata.DynamicDataTypeSupport.print_data (p. 891)

8.73.6.8 void get_info (DynamicDataInfo *info_out*)

Fill in the given descriptor with information about this **com.rti.dds.dynamicdata.DynamicData** (p. 780).

MT Safety:

UNSAFE.

Parameters:

info_out *<<**out**>>* *(p. 271)* The descriptor object whose contents will be overwritten by this operation.

8.73.6.9 void bind_type (TypeCode *type*)

If this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object is not yet associated with a data type, set that type now to the given TypeCode.

This advanced operation allows you to reuse a single **com.rti.dds.dynamicdata.DynamicData** (p. 780) object with multiple data types.

```
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. 780) object will retain a reference to the TypeCode object you provide. It is *not* safe to delete the TypeCode until after it is unbound. You have two options:

- Keep a reference to the TypeCode object yourself, and delete it with **TypeCodeFactory.delete_tc** (p. 1650) after you've finished using it.
- Do not keep a reference to the TypeCode. The garbage collector will delete it when it's eligible for collection.

MT Safety:

UNSAFE.

Parameters:

type *<<in>>* *(p. 271)* The type to associate with this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.unbind_type (p. 799)

8.73.6.10 void unbind_type ()

Dissociate this **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.bind_type (p. 798)
com.rti.dds.dynamicdata.DynamicData.clear_all_members (p. 796)

8.73.6.11 void bind_complex_member (DynamicData *value_out*, String *member_name*, int *member_id*)

Use another **com.rti.dds.dynamicdata.DynamicData** (p. 780) object to provide access to a complex field of this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

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. 780) object to that member. This operation will bind the type code of the member to the provided **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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 TypeCode.

Only a single member of a given **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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. 802) 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. 824) 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. 824) 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. 789).

MT Safety:

UNSAFE.

Parameters:

value_out *<<**out**>>* *(p. 271)* The object that you wish to bind to the field.

member_name *<<**in**>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.unbind_complex_member (p. 802)
com.rti.dds.dynamicdata.DynamicData.get_complex_member (p. 824)

8.73.6.12 void unbind_complex_member (DynamicData *value*)

Tear down the association created by a **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 800) 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. 800) on a data sample, you must unbind before writing or deleting the sample.

MT Safety:

UNSAFE.

Parameters:

value *<<in>>* *(p. 271)* The same object you passed to **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 800). This argument is used for error checking purposes.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 800)

8.73.6.13 TypeCode get_type ()

Get the data type, of which this **com.rti.dds.dynamicdata.DynamicData** (p. 780) represents an instance.

MT Safety:

UNSAFE.

8.73.6.14 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. 802) followed by **TypeCode.kind** (p. 1615).

MT Safety:

UNSAFE.

8.73.6.15 int get_member_count ()

Get the number of members in this sample.

For objects of type kind **TCKind.TK_ARRAY** (p. 1529) or **TCKind.TK_SEQUENCE** (p. 1529), this method returns the number of elements in the collection.

For objects of type kind **TCKind.TK_STRUCT** (p. 1529) or **TCKind.TK_VALUE** (p. 1530), it returns the number of fields in the sample, which will always be the same as the number of fields in the type.

For objects of type kind **TCKind.TK_SPARSE** (p. 1530), it returns the number of fields in the sample, which may be less than or equal to the number of fields in the type.

MT Safety:

UNSAFE.

See also:

com.rti.dds.dynamicdata.DynamicData.get_member_info_by_index (p. 805)

8.73.6.16 boolean member_exists (String *member_name*, int *member_id*)

Indicates whether a member of a particular name/ID exists in this data sample.

Only one of the name and/or ID need by specified.

For objects of type kinds other than **TCKind.TK_SPARSE** (p. 1530), the result of this method will always be the same as that of **com.rti.dds.dynamicdata.DynamicData.member_exists_in_type** (p. 804).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

See also:

com.rti.dds.dynamicdata.DynamicData.member_exists_in_type (p. 804)
com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 843)

8.73.6.17 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.

Only one of the name and/or ID need by specified.

For objects of type kinds other than **TCKind.TK_SPARSE** (p. 1530), the result of this method will always be the same as that of **com.rti.dds.dynamicdata.DynamicData.member_exists** (p. 803).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

See also:

com.rti.dds.dynamicdata.DynamicData.member_exists (p. 803)

com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED (p. 843)

8.73.6.18 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. 780) sample.

This operation is valid for objects of TCKind **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), and **TCKind.TK_SPARSE** (p. 1530).

MT Safety:

UNSAFE.

Parameters:

info *<<**out**>>* *(p. 271)* The descriptor object whose contents will be overwritten by this operations.

member_name *<<**in**>>* *(p. 271)* 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. 271)* The ID of the member for which to get the info, or **com.rti.dds.dynamicdata.DynamicData.MEMBER_-ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.get_member_info_by_-index (p. 805)
com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED (p. 843)

8.73.6.19 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. 780) sample.

This operation is valid for objects of TCKind **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), and **TCKind.TK_SPARSE** (p. 1530).

MT Safety:

UNSAFE.

Parameters:

info *<<**out**>> (p. 271)* The descriptor object whose contents will be overwritten by this operations.

index *<<**in**>> (p. 271)* The zero-based of the member for which to get the info.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 805)
com.rti.dds.dynamicdata.DynamicData.get_member_count (p. 803)
com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED (p. 843)

8.73.6.20 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 TCKind **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), and **TCKind.TK_SPARSE** (p. 1530). For type kinds **TCKind.TK_ARRAY** (p. 1529) and **TCKind.TK_SEQUENCE** (p. 1529), 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.get_member_info (p. 805)
com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED (p. 843)

8.73.6.21 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 **TCKind.TK_-STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), or **TCKind.TK_-SPARSE** (p. 1530).

Note to users of sparse types: A key member may only have a single representation and is required to exist in every sample.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.73.6.22 int get_int (String *member_name*, int *member_id*)

Get the value of the given field, which is of type int or another type implicitly convertible to it (byte, char, short, short, or **com.rti.dds.util.Enum** (p. 925)).

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_int (p. 825)

8.73.6.23 int get_int_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 IntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_int_array (p. 826)
DynamicData.get_int_seq (p. 809)

8.73.6.24 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 int.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.set_int_seq (p. 826)
DynamicData.get_int_array (p. 808)

8.73.6.25 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_short (p. 827)

8.73.6.26 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. 1446).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_short_array (p. 828)
com.rti.dds.dynamicdata.DynamicData.get_short_seq (p. 811)

8.73.6.27 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_short_seq (p. 828)
com.rti.dds.dynamicdata.DynamicData.get_short_array (p. 810)

8.73.6.28 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_float (p. 829)

8.73.6.29 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. 936).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_float_array (p. 830)
com.rti.dds.dynamicdata.DynamicData.get_float_seq (p. 813)

8.73.6.30 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_float_seq (p. 830)
com.rti.dds.dynamicdata.DynamicData.get_float_array (p. 812)

8.73.6.31 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_double (p. 831)

8.73.6.32 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. 759).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_double_array (p. 832)
com.rti.dds.dynamicdata.DynamicData.get_double_seq (p. 815)

8.73.6.33 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_double_seq (p. 832)
com.rti.dds.dynamicdata.DynamicData.get_double_array (p. 814)

8.73.6.34 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_boolean (p. 833)

8.73.6.35 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. 405).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_boolean_array (p. 834)
com.rti.dds.dynamicdata.DynamicData.get_boolean_seq (p. 817)

8.73.6.36 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_boolean_seq (p. 834)
com.rti.dds.dynamicdata.DynamicData.get_boolean_array (p. 816)

8.73.6.37 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_char (p. 835)

8.73.6.38 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. 445).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.dynamicdata.DynamicData.set_char_array (p. 836)
com.rti.dds.dynamicdata.DynamicData.get_char_seq (p. 819)

8.73.6.39 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_-UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_-RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_char_seq (p. 836)
com.rti.dds.dynamicdata.DynamicData.get_char_array (p. 818)

8.73.6.40 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_byte (p. 837)

8.73.6.41 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 ByteSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_byte_array (p. 838)
DynamicData.get_byte_seq (p. 821)

8.73.6.42 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.set_byte_seq (p. 838)
DynamicData.get_byte_array (p. 820)

8.73.6.43 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, short, int, long, or **com.rti.dds.util.Enum** (p. 925)).

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_long (p. 839)

8.73.6.44 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 LongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 271)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

DynamicData.set_long_array (p. 840)
DynamicData.get_long_seq (p. 823)

8.73.6.45 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.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 271)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.set_long_seq (p. 840)
DynamicData.get_long_array (p. 822)

8.73.6.46 String get_string (String *member_name*, int *member_id*)

Get the value of the given field, which is of type String.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_string (p. 841)

8.73.6.47 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.

The member may be of type kind **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), **TCKind.TK_UNION** (p. 1529), or **TCKind.TK_SPARSE** (p. 1530). It may be specified by name or by ID.

This method is logically related to **com.rti.dds.dynamicdata.DynamicData.bind_complex_member** (p. 800) 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. 271)* The **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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. 780) sample.

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.set_complex_member (p. 842)
com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 800)

8.73.6.48 void set_int (String *member_name*, int *member_id*, int *value*)

Set the value of the given field, which is of type int.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_int (p. 808)

8.73.6.49 void set_int_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 IntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_int_array (p. 808)
DynamicData.set_int_seq (p. 826)

8.73.6.50 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 int.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_int_seq (p. 809)
DynamicData.set_int_array (p. 826)

8.73.6.51 void set_short (String *member_name*, int *member_id*, short *value*)

Set the value of the given field, which is of type short.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_short (p. 810)

8.73.6.52 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. 1446).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_short_array (p. 810)
com.rti.dds.dynamicdata.DynamicData.set_short_seq (p. 828)

8.73.6.53 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_short_seq (p. 811)
com.rti.dds.dynamicdata.DynamicData.set_short_array (p. 828)

8.73.6.54 void set_float (String *member_name*, int *member_id*, float *value*)

Set the value of the given field, which is of type float.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_float (p. 812)

8.73.6.55 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. 936).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_float_array (p. 812)
com.rti.dds.dynamicdata.DynamicData.set_float_seq (p. 830)

8.73.6.56 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_float_seq (p. 813)
com.rti.dds.dynamicdata.DynamicData.set_float_array (p. 830)

8.73.6.57 void set_double (String *member_name*, int *member_id*, double *value*)

Set the value of the given field, which is of type double.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_double (p. 814)

8.73.6.58 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. 759).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_double_array (p. 814)
com.rti.dds.dynamicdata.DynamicData.set_double_seq (p. 832)

8.73.6.59 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_double_seq (p. 815)
com.rti.dds.dynamicdata.DynamicData.set_double_array (p. 832)

8.73.6.60 void set_boolean (String *member_name*, int *member_id*, boolean *value*)

Set the value of the given field, which is of type boolean.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_boolean (p. 816)

8.73.6.61 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. 405).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_boolean_array (p. 816)
com.rti.dds.dynamicdata.DynamicData.set_boolean_seq (p. 834)

8.73.6.62 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_boolean_seq (p. 817)
com.rti.dds.dynamicdata.DynamicData.set_boolean_array (p. 834)

8.73.6.63 void set_char (String *member_name*, int *member_id*, char *value*)

Set the value of the given field, which is of type char.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_char (p. 818)

8.73.6.64 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. 445).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_char_array (p. 818)
com.rti.dds.dynamicdata.DynamicData.set_char_seq (p. 836)

8.73.6.65 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_char_seq (p. 819)
com.rti.dds.dynamicdata.DynamicData.set_char_array (p. 836)

8.73.6.66 void set_byte (String *member_name*, int *member_id*, byte *value*)

Set the value of the given field, which is of type byte.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_byte (p. 820)

8.73.6.67 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 ByteSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_byte_array (p. 820)
DynamicData.set_byte_seq (p. 838)

8.73.6.68 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_byte_seq (p. 821)
DynamicData.set_byte_array (p. 838)

8.73.6.69 void set_long (String *member_name*, int *member_id*, long *value*)

Set the value of the given field, which is of type long.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_long (p. 822)

8.73.6.70 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 LongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

array *<<in>> (p. 271)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_long_array (p. 822)
DynamicData.set_long_seq (p. 840)

8.73.6.71 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.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<out>>* *(p. 271)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

DynamicData.get_long_seq (p. 823)
DynamicData.set_long_array (p. 840)

8.73.6.72 void set_string (String *member_name*, int *member_id*, String *value*)

Set the value of the given field of type String.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_string (p. 824)

8.73.6.73 void set_complex_member (String *member_name*, int *member_id*, DynamicData *value*)

Copy the state of the given **com.rti.dds.dynamicdata.DynamicData** (p. 780) object into a member of this object.

The member may be of type kind **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530), **TCKind.TK_UNION** (p. 1529), or **TCKind.TK_SPARSE** (p. 1530). 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. 800) to copy from one **com.rti.dds.dynamicdata.DynamicData** (p. 780) object to another efficiently. Suppose the following data structure:

```
struct Bar {
    short theShort;
};

struct Foo {
    Bar theBar;
};
```

Support we have two instances of `Foo`: `foo_dst` and `foo_src`. We want to replace the contents of `foo_dst.theBar` with the contents of `foo_src.theBar`. 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(
    "theBar",
    DynamicData.MEMBER_ID_UNSPECIFIED,
    bar);
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. 271)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 271)* The ID of the member or **com.rti.dds.dynamicdata.DynamicData.MEMBER_ID_UNSPECIFIED** (p. 843) to look up by name. See **Member Names and IDs** (p. 787).

value *<<in>>* *(p. 271)* The source **com.rti.dds.dynamicdata.DynamicData** (p. 780) object whose contents will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_OUT_OF_RESOURCES

See also:

com.rti.dds.dynamicdata.DynamicData.get_complex_member (p. 824)
com.rti.dds.dynamicdata.DynamicData.bind_complex_member (p. 800)

8.73.7 Member Data Documentation

8.73.7.1 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.

8.74 DynamicDataInfo Class Reference

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

Inherits Struct.

Public Member Functions

- **DynamicDataInfo** ()

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.*

- **DynamicDataInfo** (int **member_count**, int **stored_size**, boolean is_optimized_storage)

  *A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.*

Public Attributes

- int **member_count**

  *The number of data members in this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.*

- int **stored_size**

  *The number of bytes currently used to store the data of this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.*

8.74.1 Detailed Description

A descriptor for a **com.rti.dds.dynamicdata.DynamicData** (p. 780) object.

See also:

com.rti.dds.dynamicdata.DynamicData.get_info (p. 798)

8.74.2 Member Data Documentation

8.74.2.1 int member_count

The number of data members in this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.

8.74.2.2 int stored_size

The number of bytes currently used to store the data of this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.

8.75 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. 780) sample's type.*

- String **member_name**

  The string name of the data member.

- boolean **member_exists**

  Indicates whether the corresponding member of the data type actually 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.

8.75.1 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. 805)

8.75.2 Member Data Documentation

8.75.2.1 int member_id

An integer that uniquely identifies the data member within this **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample's type.

For sparse data types, this value will be assigned by the type designer. For types defined in IDL, it will be assigned automatically by the middleware based on the member's declaration order within the type.

See also:

TCKind

8.75.2.2 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.

See also:

com.rti.dds.dynamicdata.DynamicDataMemberInfo.representation_count

8.75.2.3 boolean member_exists

Indicates whether the corresponding member of the data type actually exists in this sample.

For non-sparse data types, this value will always be true.

See also:

TCKind

8.75.2.4 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 TypeCode and getting the TCKind from there.

8.75.2.5 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.

8.75.2.6 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 **TCKind.TK_NULL** (p. 1528) here.

8.76 DynamicDataProperty_t Class Reference

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

Public Member Functions

- **DynamicDataProperty_t** ()

  The constructor.

- **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 this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object, in bytes.*

- int **buffer_max_size** = 65536

  *The maximum amount of memory that this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object may use, in bytes.*

8.76.1 Detailed Description

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

8.76.2 Constructor & Destructor Documentation

8.76.2.1 DynamicDataProperty_t ()

The constructor.

8.76.2.2 DynamicDataProperty_t (int *buffer_initial_size*, int *buffer_max_size*, int *buffer_max_size_increment*)

The constructor.

8.76.3 Member Data Documentation

8.76.3.1 int buffer_initial_size = 0

The initial amount of memory used by this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object, in bytes.

See also:

com.rti.dds.dynamicdata.DynamicDataProperty_t.buffer_max_size (p. 850)

8.76.3.2 int buffer_max_size = 65536

The maximum amount of memory that this **com.rti.dds.dynamicdata.DynamicData** (p. 780) object may use, in bytes.

It will grow to this size from the initial size as needed.

See also:

com.rti.dds.dynamicdata.DynamicDataProperty_t.buffer_initial_size (p. 850)

8.77 DynamicDataReader Class Reference

Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 780).

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. 473).*

- 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. 473).*

- void **read_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **take_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take 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. 473).*

- 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. 473).*

- 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. 473).*

- 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. 473).*

- void **read_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)
- void **take_instance_w_condition** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)
- 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. 473).*

- 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. 473).*

- 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **return_loan** (**DynamicDataSeq** received_data, **SampleInfoSeq** info_seq)

  Indicates to the ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 473).*

- 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`.

8.77.1 Detailed Description

Reads (subscribes to) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 780).

Instantiates `com.rti.dds.subscription.DataReader` (p. 473) < **com.rti.dds.dynamicdata.DynamicData** (p. 780) > .

See also:

com.rti.dds.subscription.DataReader (p. 473)
com.rti.dds.topic.example.FooDataReader
com.rti.dds.dynamicdata.DynamicData (p. 780)

8.77.2 Member Function Documentation

8.77.2.1 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. 473).

This operation offers the same functionality and API as com.rti.dds.topic.example.FooDataReader.take except that the samples returned remain in the **com.rti.dds.subscription.DataReader** (p. 473) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of com.rti.dds.topic.example.FooDataReader.take() 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 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 be ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the instance.

Important: If the samples "returned" by this method are loaned from RTI Connext (see com.rti.dds.topic.example.FooDataReader.take 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. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching ones of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition, com.rti.dds.topic.example.FooDataReader.take,

com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.2.2 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. 473).

The operation will return the list of samples received by the **com.rti.dds.subscription.DataReader** (p. 473) since the last com.rti.dds.topic.example.FooDataReader.take operation that match the specified com.rti.dds.subscription.SampleStateMask, com.rti.dds.subscription.ViewStateMask and com.rti.dds.subscription.InstanceStateMask.

This operation may fail with RETCODE_ERROR if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_outstanding_reads** (p. 530) 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. 1071), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356), **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524) and the characteristics of the data-type that is associated with the **com.rti.dds.subscription.DataReader** (p. 473):

- In the case where the **com.rti.dds.infrastructure.HistoryQosPolicy.kind** (p. 1073) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, the call will return at most **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074) samples per instance.

- The maximum number of samples returned is limited by **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359), and by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_read** (p. 530).

- For multiple instances, the number of samples returned is additionally limited by the product (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360) * **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360))

- If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_infos** (p. 528) is limited, the number of samples returned may also be limited if insufficient **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) 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 insuffificient resources, in order to access remaining samples.

Note that in the case where the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.subscription.DataReader** (p. 473) is bound to a datatype that has no key definition, then there will be at most one instance in the **com.rti.dds.subscription.DataReader** (p. 473). 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 ViewStateKind.NOT_NEW_VIEW_STATE. It will not affect the `instance_state` of the sample's instance.

After com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.take 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. 473). 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. 473) by calling the com.rti.dds.topic.example.FooDataReader.return_loan 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. 1404) objects after the call to com.rti.dds.topic.example.FooDataReader.return_loan. 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.dds.topic.example.FooDataReader.return_loan at some point, you do *not* have to do so before the next com.rti.dds.topic.example.FooDataReader.take call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader**

(p. 473) 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. 473) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when com.rti.dds.topic.example.FooDataReader.take is called, samples are copied to received_data and info_seq. The application will not need to call com.rti.dds.topic.example.FooDataReader.return_loan.

The order of the samples returned to the caller depends on the **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1237).

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is 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. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection is a list were the relative order of samples is preserved also accross different instances. Note that samples belonging to the same instance may or may not be consecutive. This is because to preserve order it may be necessary to mix samples from different instances.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive. [**Not supported (optional)**]

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessS-

copeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection contains at most one sample. The difference in this case is due to the fact that is required that the application is able to read samples belonging to different **com.rti.dds.subscription.DataReader** (p. 473) objects in a specific order. [**Not supported (optional)**]

In any case, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 51) policy:

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 609) is 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. 609) is 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

In addition to the collection of samples, the read and take operations also use a collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures.

8.77.3 SEQUENCES USAGE IN TAKE AND READ

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 Sequence.size())

- the maximum length (`max_len`, see **Sequence.getMaximum** (p. 1433))

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 RETCODE_PRECONDITION_NOT_MET.

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. 473). 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. 538) using com.rti.dds.topic.example.FooDataReader.return_loan.

4. If the initial `max_len>0` then the read or take operation will fail with RETCODE_PRECONDITION_NOT_MET. This avoids the potential hard-to-detect memory leaks caused by an application forgetting to return the loan.

5. If initial `max_len>0` then the read or take operation will copy the `received_data` values and **com.rti.dds.subscription.SampleInfo** (p. 1404) 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 RETCODE_PRECONDITION_NOT_MET. 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. 473), 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. 473). If this is the case, the application will need to use com.rti.dds.topic.example.FooDataReader.return_loan to return the loan once it is no longer using the `received_data` in the collection. When com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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 Foo values and the collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures are of the same length and are in a one-to-one correspondence. Each **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) is InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE or InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE, then the last sample for that instance in the collection (that is, the one whose **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1356). The act of reading/taking a sample sets its sample_state to 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 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 (Foo). If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operations fails with RETCODE_NO_DATA.

For an example on how `take` can be used, please refer to the **receive example** (p. 246).

Parameters:

received_data *<<inout>>* *(p. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is

NULL.

Parameters:

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described above.

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching one of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.1 void read_w_condition (DynamicDataSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

In case the **com.rti.dds.subscription.ReadCondition** (p. 1326) is a plain **com.rti.dds.subscription.ReadCondition** (p. 1326) and not the specialized **com.rti.dds.subscription.QueryCondition** (p. 1324), the operation is equivalent to calling com.rti.dds.topic.example.FooDataReader.read 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. 1326).

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.read.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operation will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read

com.rti.dds.topic.example.FooDataReader.take,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.2 void take_w_condition (DynamicDataSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

This operation is analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except that it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.take.

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.3 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473). This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). The implied order among the samples stored in the **com.rti.dds.subscription.DataReader** (p. 473) is the same as for the com.rti.dds.topic.example.FooDataReader.read operation.

The com.rti.dds.topic.example.FooDataReader.read_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific Foo 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 RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read

8.77.3.4 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' it from the **com.rti.dds.subscription.DataReader** (p. 473) so that it is no longer accessible. This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). This operation is analogous to the com.rti.dds.topic.example.FooDataReader.read_next_sample except for the fact that the sample is removed from the **com.rti.dds.subscription.DataReader** (p. 473).

The com.rti.dds.topic.example.FooDataReader.take_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific Foo 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 RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take

8.77.3.5 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read, 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. 1404) verifies **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1410) == `a_handle`.

The com.rti.dds.topic.example.FooDataReader.read_instance operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.read_instance operation follows the same rules as the

com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.6 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.take, 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.dds.topic.example.FooDataReader.take operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.take_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method fails with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL. The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.7 void read_instance_w_condition (DynamicDataSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *previous_handle*, ReadCondition *condition*)

8.77.3.8 void take_instance_w_condition (DynamicDataSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *previous_handle*, ReadCondition *condition*)

8.77.3.9 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) where all the samples belong to a single instance. The behavior is similar to com.rti.dds.topic.example.FooDataReader.read_instance, 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. 473). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of com.rti.dds.topic.example.FooDataReader.read_next_instance is 'as if' the **com.rti.dds.subscription.DataReader** (p. 473) invoked com.rti.dds.topic.example.FooDataReader.read_instance, 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082) will return the samples for the instance which has the smallest `instance_handle`

among all the instances that contain available samples.

The operation com.rti.dds.topic.example.FooDataReader.read_next_instance is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082), examines the samples returned, and then uses the `instance_handle` returned in the **com.rti.dds.subscription.SampleInfo** (p. 1404) as the value of the `previous_handle` argument to the next call to com.rti.dds.topic.example.FooDataReader.read_next_instance. The iteration continues until com.rti.dds.topic.example.FooDataReader.read_next_instance fails with the value RETCODE_NO_DATA.

Note that it is possible to call the com.rti.dds.topic.example.FooDataReader.read_next_instance operation with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473). 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. 473). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE 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.dds.topic.example.FooDataReader.read_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must

be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<**in**>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<**in**>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states *<<**in**>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<**in**>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<**in**>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.10 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

This operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance, except that

the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states <<*in*>> *(*p. *271)* data samples matching ones of these `sample_states` are returned

view_states <<*in*>> *(*p. *271)* data samples matching ones of these `view_state` are returned

instance_states <<*in*>> *(*p. *271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.11 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read_next_instance, 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. 1326) evaluates to TRUE.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the

pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.12 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

The operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq**

(p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>> (p. 271)* The maximum number of samples to be returned. If the special value ResourceLimitsQosPolicy.LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>> (p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>> (p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, or RETCODE_NO_DATA, RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED

8.77.3.13 void return_loan (DynamicDataSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

This operation indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

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. 473) to which they are returned.

If either of these conditions is not met, the operation will fail with RETCODE_-PRECONDITION_NOT_MET.

The operation com.rti.dds.topic.example.FooDataReader.return_loan allows implementations of the read and take operations to "loan" buffers from the **com.rti.dds.subscription.DataReader** (p. 473) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), the application should not retain them indefinitely.

The use of com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL FooSeq. The method will fail with RETCODE_-BAD_PARAMETER if it is NULL.

Parameters:

info_seq *<<in>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_-PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

8.77.3.14 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.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

key_holder *<<inout>>* *(p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<in>>* *(p. 271)* the `instance` whose key is to be retrieved. If Foo *has* a key, `handle` must represent an existing instance of type Foo known to the **com.rti.dds.subscription.DataReader** (p. 473). Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. If Foo has a key and `handle` represents an instance of another type or an instance of type Foo that has been unregistered, this method will fail with RETCODE_BAD_PARAMETER. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.77.3.15 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. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value HANDLE_NIL.

Parameters:

key_holder *<<in>>* *(p. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If Foo has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.78 DynamicDataSeq Class Reference

An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 780) elements.

Inheritance diagram for DynamicDataSeq::

Public Member Functions

- ^ **DynamicDataSeq** ()

  Construct a new empty ***com.rti.dds.dynamicdata.DynamicDataSeq*** *(p. 881).*

- ^ **DynamicDataSeq** (int initialMaximum)

  Construct a new empty ***com.rti.dds.dynamicdata.DynamicDataSeq*** *(p. 881).*

- ^ **DynamicDataSeq** (Collection elements)

  Construct a new ***com.rti.dds.dynamicdata.DynamicDataSeq*** *(p. 881) containing the same elements as the given collection.*

8.78.1 Detailed Description

An ordered collection of **com.rti.dds.dynamicdata.DynamicData** (p. 780) elements.

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.dynamicdata.DynamicData** (p. 780) > .

See also:

com.rti.dds.util.Sequence (p. 1432)
com.rti.dds.dynamicdata.DynamicData (p. 780)
`http://java.sun.com/javase/6/docs/api/java/util/List.html`

8.78.2 Constructor & Destructor Documentation

8.78.2.1 DynamicDataSeq ()

Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 881).

8.78.2.2 DynamicDataSeq (int *initialMaximum*)

Construct a new empty **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 881).

The new sequence will have the given maximum. (The *maximum* of the sequence is equivalent to what, in an `ArrayList`, is called the *capacity*.)

See also:

Sequence.getMaximum (p. 1433)
`http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html`

8.78.2.3 DynamicDataSeq (Collection *elements*)

Construct a new **com.rti.dds.dynamicdata.DynamicDataSeq** (p. 881) containing the same elements as the given collection.

See also:

`http://java.sun.com/javase/6/docs/api/java/util/Collection.html`

8.79 DynamicDataTypeProperty_t Class Reference

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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. 780) sample created from the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).*

- final **DynamicDataTypeSerializationProperty_t serialization**

  Properties that govern how the data of this type will be serialized on the network.

8.79.1 Detailed Description

A collection of attributes used to configure **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

8.79.2 Constructor & Destructor Documentation

8.79.2.1 DynamicDataTypeProperty_t ()

The constructor.

8.79.2.2 DynamicDataTypeProperty_t (DynamicDataProperty_t *data*, DynamicDataTypeSerializationProperty_t *serialization*)

The constructor.

8.79.3 Member Data Documentation

8.79.3.1 final DynamicDataProperty_t data

Initial value:

```
new DynamicDataProperty_t()
```

These properties will be provided to every new **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample created from the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).

8.79.3.2 final DynamicDataTypeSerializationProperty_t serialization

Initial value:

```
new DynamicDataTypeSerializationProperty_t()
```

Properties that govern how the data of this type will be serialized on the network.

8.80 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**)

  The constructor.

Public Attributes

- boolean **use_42e_compatible_alignment** = false

  Use RTI Connext 4.2e-compatible alignment for large primitive types.

- int **max_size_serialized** = 0xffffffff

  The maximum number of bytes that objects of a given type could consume when serialized on the network.

8.80.1 Detailed Description

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

8.80.2 Constructor & Destructor Documentation

8.80.2.1 DynamicDataTypeSerializationProperty_t ()

The constructor.

8.80.2.2 DynamicDataTypeSerializationProperty_t (boolean *use_42e_compatible_alignment*, int *max_size_serialized*)

The constructor.

8.80.3 Member Data Documentation

8.80.3.1 boolean use_42e_compatible_alignment = false

Use RTI Connext 4.2e-compatible alignment for large primitive types.

In RTI Connext 4.2e, the default alignment for large primitive types – long, 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.

8.80.3.2 int max_size_serialized = 0xffffffff

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 TypeCode, whichever is smaller.

8.81 DynamicDataTypeSupport Class Reference

A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

Inheritance diagram for DynamicDataTypeSupport::

Public Member Functions

- final void **register_type** (**DomainParticipant** participant, String type_name)

  *Associate the TypeCode with the given **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629).*

- final String **get_type_name** ()

  Get the default name of this type.

- final **TypeCode get_data_type** ()

  *Get the TypeCode wrapped by this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).*

- Object **create_data** ()

  *Create a new **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample initialized with the TypeCode and properties of this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).*

- void **destroy_data** (Object data)

  *Finalize and deallocate the **com.rti.dds.dynamicdata.DynamicData** (p. 780) 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 **delete** ()

  *Delete a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) object.*

- **DynamicDataTypeSupport** (**TypeCode** type, **DynamicDataTypeProperty_t** props)

  *Construct a new **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) object.*

Static Public Attributes

- static final **DynamicDataTypeProperty_t** **TYPE_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **com.rti.dds.dynamicdata.DynamicDataTypeProperty_t** (p. 883).*

Static Protected Attributes

- static final RuntimeException **DYNAMICDATA_TYPE_NOT_SUPPORTED** = new **RETCODE_ERROR**("not a top-level type")

8.81.1 Detailed Description

A factory for registering a dynamically defined type and creating **com.rti.dds.dynamicdata.DynamicData** (p. 780) objects.

A **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) has three roles:

1. It associates a TypeCode with policies for managing objects of that type. See the constructor, **DynamicDataTypeSupport.DynamicDataTypeSupport** (p. 889).
2. It registers its type under logical names with a **com.rti.dds.domain.DomainParticipant** (p. 629). See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type** (p. 889).
3. It creates **com.rti.dds.dynamicdata.DynamicData** (p. 780) samples pre-initialized with the type and properties of the type support itself. See **com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data** (p. 891).

8.81.2 Constructor & Destructor Documentation

8.81.2.1 DynamicDataTypeSupport (TypeCode *type*, DynamicDataTypeProperty_t *props*)

Construct a new **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) object.

This step is usually followed by type registration.

The new object created by this constructor retains a reference to the TypeCode that is passed in. It is *not* safe to delete the TypeCode until the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) itself is deleted. You have two options:

- Keep a reference to the TypeCode object yourself, and delete it with **TypeCodeFactory.delete_tc** (p. 1650) after you've deleted the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).
- Do not keep a reference to the TypeCode. The garbage collector will delete it when it's eligible for collection.

Parameters:

type The TypeCode 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 **DynamicDataTypeSupport.TYPE_PROPERTY_DEFAULT** (p. 174).

See also:

com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type (p. 889)

8.81.3 Member Function Documentation

8.81.3.1 final void register_type (DomainParticipant *participant*, String *type_name*)

Associate the TypeCode with the given **com.rti.dds.domain.DomainParticipant** (p. 629) under the given logical name.

Once a type has been registered, it can be referenced by name when creating a **topic** (p. 350). Statically and dynamically defined types behave the same way in this respect.

See also:

com.rti.dds.topic.example.FooTypeSupport.register_type (p. 1060)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)
com.rti.dds.dynamicdata.DynamicDataTypeSupport.unregister_type (p. 890)

8.81.3.2 final void unregister_type (DomainParticipant *participant*, String *type_name*)

Remove the definition of this type from the **com.rti.dds.domain.DomainParticipant** (p. 629).

This operation is optional; all types are automatically unregistered when a **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 1545) is still referring to it.

See also:

com.rti.dds.topic.example.FooTypeSupport.unregister_type
com.rti.dds.dynamicdata.DynamicDataTypeSupport.register_type (p. 889)

8.81.3.3 final String get_type_name ()

Get the default name of this type.

The TypeCode that is wrapped by this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) 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.dds.topic.example.FooTypeSupport.get_type_name

8.81.3.4 final TypeCode get_data_type ()

Get the TypeCode wrapped by this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).

8.81.3.5 Object create_data ()

Create a new **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample initialized with the TypeCode and properties of this **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887).

See also:

com.rti.dds.topic.example.FooTypeSupport.create_data
DynamicData.DynamicData
com.rti.dds.dynamicdata.DynamicDataTypeProperty_t.data (p. 884)

8.81.3.6 void destroy_data (Object *data*)

Finalize and deallocate the **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample.

See also:

com.rti.dds.topic.example.FooTypeSupport.delete_data
com.rti.dds.dynamicdata.DynamicDataTypeSupport.create_data (p. 891)

8.81.3.7 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. 797).

See also:

com.rti.dds.dynamicdata.DynamicData.print (p. 797)

8.81.3.8 void copy_data (DynamicData *dst*, DynamicData *src*)

Deeply copy the given data samples.

8.81.3.9 void delete ()

Delete a **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) object.

A **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) cannot be deleted while it is still in use. For each **com.rti.dds.domain.DomainParticipant** (p. 629) with which the **com.rti.dds.dynamicdata.DynamicDataTypeSupport** (p. 887) 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. 890)
DynamicDataTypeSupport.DynamicDataTypeSupport (p. 889)

8.81.4 Member Data Documentation

8.81.4.1 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.

8.82 DynamicDataWriter Class Reference

Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 780).

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_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.dds.topic.example.FooDataWriter.register_instance.

- void **unregister_instance_w_timestamp** (**DynamicData** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance 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_w_timestamp** (**DynamicData** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.topic.example.FooDataWriter.write 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 data.

- 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. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

- 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`*.*

8.82.1 Detailed Description

Writes (publishes) objects of type **com.rti.dds.dynamicdata.DynamicData** (p. 780).

Instantiates `com.rti.dds.publication.DataWriter` (p. 538) < **com.rti.dds.dynamicdata.DynamicData** (p. 780) > .

See also:

com.rti.dds.publication.DataWriter (p. 538)
com.rti.dds.topic.example.FooDataWriter
com.rti.dds.dynamicdata.DynamicData (p. 780)

8.82.2 Member Function Documentation

8.82.2.1 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082). 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. 900) or **dispose()** (p. 904) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) as the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) paramater 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.dds.topic.example.FooDataWriter.write, com.rti.dds.topic.example.FooDataWriter.write_w_timestamp, com.rti.dds.topic.example.FooDataWriter.dispose and com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) 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. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL** (p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance, com.rti.dds.topic.example.FooDataWriter.get_key_value, **RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP** (p. 1218)

8.82.2.2 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. 51) QoS policy for details.

This operation may fail and return **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) limit has been exceeded.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL** (p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance,
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.82.2.3 void unregister_instance (DynamicData *instance_data*, InstanceHandle_t *handle*)

Reverses the action of com.rti.dds.topic.example.FooDataWriter.register_instance.

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.dds.topic.example.FooDataWriter.register_instance. Otherwise, this operation may fail with RETCODE_BAD_PARAMETER.

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. 538) 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 com.rti.dds.topic.example.FooDataWriter.unregister_instance().

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 897) operation is for the instance as indicated by the `handle`.

If after a com.rti.dds.topic.example.FooDataWriter.unregister_instance, the application wants to modify (com.rti.dds.topic.example.FooDataWriter.write or com.rti.dds.topic.example.FooDataWriter.dispose) an instance, it has to register it again, or else use the special `handle` value **InstanceHandle_t.HANDLE_-**

NIL (p. 1082).

This operation does not indicate that the instance is deleted (that is the purpose of com.rti.dds.topic.example.FooDataWriter.dispose). The operation com.rti.dds.topic.example.FooDataWriter.unregister_instance just indicates that the **com.rti.dds.publication.DataWriter** (p. 538) no longer has anything to say about the instance. **com.rti.dds.subscription.DataReader** (p. 473) entities that are reading the instance may receive a sample with InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE for the instance, unless there are other **com.rti.dds.publication.DataWriter** (p. 538) objects writing that same instance.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 83)). If the **com.rti.dds.publication.DataWriter** (p. 538) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 897) will relinquish that ownership.

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS 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. 1339); if this writer is still unable to unregister after that period, this method will fail with RETCODE_TIMEOUT.

Parameters:

instance_data *<<in>>* *(p. 271)* The instance that should be unregistered. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER . If Foo *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<in>>* *(p. 271)* represents the `instance` to be unregistered. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If Foo 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 RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will report the error RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
FooDataWriter.unregister_instance_w_timestamp
com.rti.dds.topic.example.FooDataWriter.get_key_value
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.82.2.4 void unregister_instance_w_timestamp (DynamicData *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance 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. 51) 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.dds.topic.example.FooDataWriter.unregister_instance operation.

This operation may block and may time out (RETCODE_TIMEOUT) under the same circumtances described for the unregister_instance operation.

Parameters:

instance_data *<<in>> (p. 271)* The instance that should be unregistered. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER. If Foo *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<in>> (p. 271)* represents the `instance` to be unregistered. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If Foo 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 RETCODE_BAD_PARAMETER.

This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<in>> (p. 271)* 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. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
com.rti.dds.topic.example.FooDataWriter.unregister_instance
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.82.2.5 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) QoS policy for details).

As a side effect, this operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 538) itself, the **com.rti.dds.publication.Publisher** (p. 1277) and the **com.rti.dds.domain.DomainParticipant** (p. 629).

Note that the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered.

If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 900) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 101) `kind` is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 102) to be exceeded.

Specifically, this operation may block in the following situations (note that the list may not be exhaustive), even if its **com.rti.dds.infrastructure.HistoryQosPolicyKind** (p. 1075) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS:

- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) * **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074)), then in the situation where the `max_samples` resource limit is exhausted, RTI Connext is allowed to discard samples of some other instance, as long as at least one sample remains for such an instance. If it is still not possible to make space available to store the modification, the writer is allowed to block.

- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360)), then the DataWriter may block regardless of the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074).

- If (**com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1389) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359)), then it is possible for the send_window_size limit to be reached before RTI Connext is allowed to discard samples, in which case the **com.rti.dds.publication.DataWriter** (p. 538) will block.

This operation may also block when using ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS and PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS. In this case, the **com.rti.dds.publication.DataWriter** (p. 538) will queue samples until they are sent by the asynchronous publishing thread. The number of samples that can be stored is determined by the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071). If the asynchronous thread does not send samples fast enough (e.g., when using a slow

com.rti.dds.publication.FlowController (p. 942)), the queue may fill up. In that case, subsequent write calls will block.

If this operation *does* block for any of the above reasons, the **RELIABILITY** (p. 101) `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 **com.rti.dds.publication.DataWriter** (p. 538) is able to store the modification without exceeding the limits, the operation will time out (RETCODE_TIMEOUT).

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

This operation will fail with RETCODE_PRECONDITION_NOT_MET 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).

Parameters:

instance_data *<<**in**>>* *(p. 271)* The data to write.

This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

Parameters:

handle *<<**in**>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_PRECONDITION_NOT_MET, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.subscription.DataReader (p. 473)
com.rti.dds.topic.example.FooDataWriter.write_w_timestamp
DESTINATION_ORDER (p. 51)

8.82.2.6 void write_w_timestamp (DynamicData *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.write 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) 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.dds.topic.example.FooDataWriter.write operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help free up some resources.

This operation may fail with RETCODE_BAD_PARAMETER under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>> (p. 271)* The data to write. This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

handle *<<in>> (p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

source_timestamp *<<in>> (p. 271)* When using 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. 609), 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. 609), the function will return RETCODE_BAD_PARAMETER.

Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.write
com.rti.dds.subscription.DataReader (p. 473)
DESTINATION_ORDER (p. 51)

8.82.2.7 void dispose (DynamicData *instance_data*, InstanceHandle_t *instance_handle*)

Requests the middleware to delete the data.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

The actual deletion is postponed until there is no more use for that data in the whole system.

Applications are made aware of the deletion by means of operations on the **com.rti.dds.subscription.DataReader** (p. 473) objects that already knew that instance. **com.rti.dds.subscription.DataReader** (p. 473) objects that didn't know the instance will never see it.

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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.unregister_instance operation.

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 **dispose()** (p. 904) operation is for the instance as indicated by the `handle`.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write().

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 271)* The data to dispose. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If Foo *has* a key, `instance_data` can be NULL only if `instance_handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If Foo has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type Foo. Otherwise, this method fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if

instance_data is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.82.2.8 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. 473) objects by means of the source_timestamp attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.dispose operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>> (p. 271)* The data to dispose. If Foo *has* a key and instance_handle is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, instance_data is not used. If Foo *has* a key, instance_data can be NULL only if instance_handle is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>> (p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If Foo has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<in>> (p. 271)* 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.82.2.9 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 Foo has no key, this method has no effect and exit with no error.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

key_holder *<<**inout**>>* *(p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<**in**>>* *(p. 271)* the `instance` whose key is to be retrieved. If Foo *has* a key, `handle` must represent a registered instance of type Foo. Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.82.2.10 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. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If Foo has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.83 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. 909) 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.

8.83.1 Detailed Description

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

8.83.2 Constructor & Destructor Documentation

8.83.2.1 EndpointGroup_t ()

Constructor.

8.83.2.2 EndpointGroup_t (EndpointGroup_t *src*)

Copy constructor.

8.83.2.3 EndpointGroup_t (String *role_name*, int *quorum_count*)

Construct an **EndpointGroup_t** (p. 909) with the given parameters.

8.83.3 Member Data Documentation

8.83.3.1 String role_name = null

Defines the role name of the endpoint group.

8.83.3.2 int quorum_count = 0

Defines the minimum number of members that satisfies the endpoint group.

8.84 EndpointGroupSeq Class Reference

A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909).

Inherits ArraySequence.

8.84.1 Detailed Description

A sequence of **com.rti.dds.infrastructure.EndpointGroup_t** (p. 909).

In the context of Collaborative DataWriters, it can be used by a **com.rti.dds.subscription.DataReader** (p. 473) to define a group of remote DataWriters that the **com.rti.dds.subscription.DataReader** (p. 473) will wait to discover before skipping missing samples.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.EndpointGroup_t (p. 909)

8.85 Entity Interface Reference

<<***interface***>> *(*p. *271)* 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. *912).*

- **StatusCondition get_statuscondition** ()

  Allows access to the ***com.rti.dds.infrastructure.StatusCondition*** *(*p. *1452) associated with the* ***com.rti.dds.infrastructure.Entity*** *(*p. *912).*

- int **get_status_changes** ()

  Retrieves the list of communication statuses in the ***com.rti.dds.infrastructure.Entity*** *(*p. *912) that are triggered.*

- **InstanceHandle_t get_instance_handle** ()

  Allows access to the ***com.rti.dds.infrastructure.InstanceHandle_t*** *(*p. *1080) associated with the* ***com.rti.dds.infrastructure.Entity*** *(*p. *912).*

8.85.1 Detailed Description

<<***interface***>> *(*p. *271)* 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. 915), may return the value **RETCODE_NOT_ENABLED** (p. 1369).

QoS:

QoS Policies (p. 90)

Status:

Status Kinds (p. 106)

Listener:

com.rti.dds.infrastructure.Listener (p. 1154)

8.85.2 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Connext.

8.85.2.1 set_qos (abstract)

This operation sets the QoS policies of the **com.rti.dds.infrastructure.Entity** (p. 912).

This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 912) classes (**com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.Subscriber** (p. 1478), and **com.rti.dds.subscription.DataReader** (p. 473)) so that the policies that are meaningful to each **com.rti.dds.infrastructure.Entity** (p. 912) can be set.

Precondition:

Certain policies are immutable (see **QoS Policies** (p. 90)): they can only be set at **com.rti.dds.infrastructure.Entity** (p. 912) creation time or before the entity is enabled. If `set_qos()` is invoked after the **com.rti.dds.infrastructure.Entity** (p. 912) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **RETCODE_IMMUTABLE_POLICY** (p. 1366).
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 **RETCODE_INCONSISTENT_POLICY** (p. 1367).
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 **RETCODE_UNSUPPORTED** (p. 1373).

Postcondition:

The existing set of policies is only changed if the `set_qos()` operation succeeds. This is indicated by a return code of RETCODE_OK. In all other cases, none of the policies are modified.

Each derived **com.rti.dds.infrastructure.Entity** (p. 912) class (**com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.Publisher**

(p. 1277), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.Subscriber** (p. 1478), **com.rti.dds.subscription.DataReader** (p. 473)) has a corresponding special value of the QoS (DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT, DomainParticipant.PUBLISHER_QOS_DEFAULT, DomainParticipant.SUBSCRIBER_QOS_DEFAULT, DomainParticipant.TOPIC_QOS_DEFAULT, Publisher.DATAWRITER_QOS_DEFAULT, Subscriber.DATAREADER_QOS_DEFAULT). 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. 912) should be changed to match the current default QoS set in the **com.rti.dds.infrastructure.Entity** (p. 912)'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. 912) has been modified to match the current default for the **com.rti.dds.infrastructure.Entity** (p. 912)'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 error codes returned in addition to **Standard Return Codes** (p. 104) : **RETCODE_IMMUTABLE_POLICY** (p. 1366), or **RETCODE_INCONSISTENT_POLICY** (p. 1367).

8.85.2.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **com.rti.dds.infrastructure.Entity** (p. 912). This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 912) classes (**com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.Subscriber** (p. 1478), and **com.rti.dds.subscription.DataReader** (p. 473)), so that the policies that are meaningful to each **com.rti.dds.infrastructure.Entity** (p. 912) can be retrieved.

Possible error codes are **Standard Return Codes** (p. 104).

8.85.2.3 set_listener (abstract)

This operation installs a **com.rti.dds.infrastructure.Listener** (p. 1154) on the **com.rti.dds.infrastructure.Entity** (p. 912). 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. 912) classes

(**com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.Subscriber** (p. 1478), and **com.rti.dds.subscription.DataReader** (p. 473)), so that the listener is of the concrete type suitable to the particular **com.rti.dds.infrastructure.Entity** (p. 912).

It is permitted to use null as the value of the listener. The null listener behaves as if the `mask` is **StatusKind.STATUS_MASK_NONE** (p. 109).

Postcondition:

Only one listener can be attached to each **com.rti.dds.infrastructure.Entity** (p. 912). 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.

8.85.2.4 get_listener (abstract)

This operation allows access to the existing **com.rti.dds.infrastructure.Listener** (p. 1154) attached to the **com.rti.dds.infrastructure.Entity** (p. 912).

This operation must be provided by each of the derived **com.rti.dds.infrastructure.Entity** (p. 912) classes (**com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.publication.DataWriter** (p. 538), **com.rti.dds.subscription.Subscriber** (p. 1478), and **com.rti.dds.subscription.DataReader** (p. 473)) so that the listener is of the concrete type suitable to the particular **com.rti.dds.infrastructure.Entity** (p. 912).

If no listener is installed on the **com.rti.dds.infrastructure.Entity** (p. 912), this operation will return null.

8.85.3 Member Function Documentation

8.85.3.1 void enable ()

Enables the **com.rti.dds.infrastructure.Entity** (p. 912).

This operation enables the **Entity** (p. 912). **Entity** (p. 912) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 69) QoS policy on the corresponding factory for the

com.rti.dds.infrastructure.Entity (p. 912).

By default, **ENTITY_FACTORY** (p. 69) is set so that it is not necessary to explicitly call **com.rti.dds.infrastructure.Entity.enable** (p. 915) on newly created entities.

The **com.rti.dds.infrastructure.Entity.enable** (p. 915) operation is idempotent. Calling enable on an already enabled **Entity** (p. 912) returns OK and has no effect.

If a **com.rti.dds.infrastructure.Entity** (p. 912) 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. 917)
- 'factory' operations
- **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917) 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 **RETCODE_NOT_ENABLED** (p. 1369).

It is legal to delete an **com.rti.dds.infrastructure.Entity** (p. 912) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **com.rti.dds.infrastructure.EntityFactoryQosPolicy** (p. 919).

Calling enable on an **Entity** (p. 912) whose factory is not enabled will fail and return **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

If **com.rti.dds.infrastructure.EntityFactoryQosPolicy.autoenable_created_entities** (p. 920) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

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. 104), **Standard Return Codes** (p. 104) or **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

8.85.3.2 StatusCondition get_statuscondition ()

Allows access to the **com.rti.dds.infrastructure.StatusCondition** (p. 1452) associated with the **com.rti.dds.infrastructure.Entity** (p. 912).

The returned condition can then be added to a **com.rti.dds.infrastructure.WaitSet** (p. 1695) so that the application can wait for specific status changes that affect the **com.rti.dds.infrastructure.Entity** (p. 912).

Returns:

the status condition associated with this entity.

8.85.3.3 int get_status_changes ()

Retrieves the list of communication statuses in the **com.rti.dds.infrastructure.Entity** (p. 912) 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. 912) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **com.rti.dds.infrastructure.Entity** (p. 912) that are triggered.

See also:

Status Kinds (p. 106)

8.85.3.4 InstanceHandle_t get_instance_handle ()

Allows access to the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) associated with the **com.rti.dds.infrastructure.Entity** (p. 912).

This operation returns the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) that represents the **com.rti.dds.infrastructure.Entity** (p. 912).

Returns:

the instance handle associated with this entity.

8.86 EntityFactoryQosPolicy Class Reference

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) 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.

8.86.1 Detailed Description

A QoS policy for all **com.rti.dds.infrastructure.Entity** (p. 912) types that can act as factories for one or more other **com.rti.dds.infrastructure.Entity** (p. 912) types.

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 708), **com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.publication.Publisher** (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **YES** (p. 98)

8.86.2 Usage

This policy controls the behavior of the **com.rti.dds.infrastructure.Entity** (p. 912) 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. 473) is created in an enabled state, its existence is immediately propagated for discovery and the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), and thus, it may not be desireable for the **com.rti.dds.subscription.DataReader** (p. 473) 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.

8.86.3 Member Data Documentation

8.86.3.1 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 true indicates that the factory `create_<entity>` operation(s) will automatically invoke the **com.rti.dds.infrastructure.Entity.enable** (p. 915) operation each time a new **com.rti.dds.infrastructure.Entity** (p. 912) is created. Therefore, the **com.rti.dds.infrastructure.Entity** (p. 912) returned by `create_<entity>` will already be enabled. A setting of false indicates that the **com.rti.dds.infrastructure.Entity** (p. 912) will not be automatically enabled. Your application will need to call **com.rti.dds.infrastructure.Entity.enable** (p. 915) itself.

The default setting of `autoenable_created_entities` = true means that, by default, it is not necessary to explicitly call **com.rti.dds.infrastructure.Entity.enable** (p. 915) on newly created

entities.

[**default**] true

8.87 EntityHowTo.MyEntityListener Class Reference

Inheritance diagram for EntityHowTo.MyEntityListener::

8.87.1 Detailed Description

{

8.88 EntityNameQosPolicy Class Reference

Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473). 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.

8.88.1 Detailed Description

Assigns a name and a role name to a **com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473). 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. 629),
com.rti.dds.subscription.DataReader (p. 473),
com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = NO;
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.88.2 Usage

The name and role name can only be 255 characters in length.

8.88.3 Member Data Documentation

8.88.3.1 String name

The name of the entity.

[default] "[ENTITY]" for **com.rti.dds.domain.DomainParticipant** (p. 629). null for **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.publication.DataWriter** (p. 538)

[range] Null terminated string with length not exceeding 255. It can be null.

8.88.3.2 String role_name

The entity role name.

With Collaborative DataWriters this name is used to specify to which endpoint group the **com.rti.dds.publication.DataWriter** (p. 538) belongs.

[range] Null terminated string with length not exceeding 255. It can be null.

[default] null

8.89 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.

8.89.1 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>.

8.89.2 Constructor & Destructor Documentation

8.89.2.1 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. 926)
com.rti.dds.util.Enum.name (p. 927)

8.89.3 Member Function Documentation

8.89.3.1 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.

8.89.3.2 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. 925)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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.89.3.3 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. 927) method rather than accessing this field.

8.89.3.4 final String toString ()

The string value of this enum constant.

See also:

com.rti.dds.util.Enum.name (p. 927)

Returns:

the name of this enum constant

8.90 EnumMember Class Reference

A description of a member of an enumeration.

Inherits java.io.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.

8.90.1 Detailed Description

A description of a member of an enumeration.

See also:

TypeCodeFactory.create_enum_tc (p. 1646)

8.90.2 Constructor & Destructor Documentation

8.90.2.1 EnumMember (String *name*, int *ordinal*)

Constructs an **EnumMember** (p. 928) object initialized with the given values.

8.90.3 Member Data Documentation

8.90.3.1 String name

The name of the enumeration member.

Cannot be null.

8.90.3.2 int ordinal

The value associated the the enumeration member.

8.91 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.

8.91.1 Detailed Description

Settings for event.

In a **com.rti.dds.domain.DomainParticipant** (p. 629), 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.DomainParticipant** (p. 629) 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. 629) 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. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.91.2 Member Data Documentation

8.91.2.1 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 Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For INTEGRITY: 80

For VxWorks: 110

For all others: OS default priority.

Stack Size:

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For INTEGRITY: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

Mask:

[**default**] mask = **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_FLOATING_POINT** (p. 1536) | **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1536)

8.91.2.2 int initial_count

The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

8.91.2.3 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] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_count

8.92 ExclusiveAreaQosPolicy Class Reference

Configures multi-thread concurrency and deadlock prevention capabilities.

Inheritance diagram for ExclusiveAreaQosPolicy::

Public Attributes

- boolean **use_shared_exclusive_area**

  *Whether the **com.rti.dds.infrastructure.Entity** (p. 912) is protected by its own exclusive area or the shared exclusive area.*

8.92.1 Detailed Description

Configures multi-thread concurrency and deadlock prevention capabilities.

An "exclusive area" is an abstraction of a multi-thread-safe region. Each entity is protected by one and only one exclusive area, although a single exclusive area may be shared by multiple entities.

Conceptually, an exclusive area is a mutex or monitor with additional deadlock protection features. If a **com.rti.dds.infrastructure.Entity** (p. 912) has "entered" its exclusive area to perform a protected operation, no other **com.rti.dds.infrastructure.Entity** (p. 912) sharing the same exclusive area may enter it until the first **com.rti.dds.infrastructure.Entity** (p. 912) "exits" the exclusive area.

Entity:

com.rti.dds.publication.Publisher (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

See also:

com.rti.dds.infrastructure.Listener (p. 1154)

8.92.2 Usage

Exclusive Areas (EAs) allow RTI Connext to be multi-threaded while preventing deadlock in multi-threaded applications. EAs prevent a **com.rti.dds.domain.DomainParticipant** (p. 629) object's internal threads from deadlocking with each other when executing internal code as well as when executing the code of user-registered listener callbacks.

Within an EA, all calls to the code protected by the EA are single threaded. Each **com.rti.dds.domain.DomainParticipant** (p. 629), **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.subscription.Subscriber** (p. 1478) entity represents a separate EA. Thus all DataWriters of the same Publisher and all DataReaders of the same Subscriber share the EA of its parent. Note: this means that operations on the DataWriters of the same Publisher and on the DataReaders of the same Subscriber will be serialized, even when invoked from multiple concurrent application threads.

Within an EA, there are limitations on how code protected by a different EA can be accessed. For example, when received data is being processed by user code in the DataReader **Listener** (p. 1154), within a Subscriber EA, the user code may call the com.rti.dds.topic.example.FooDataWriter.write operation of a DataWriter that is protected by the EA of its Publisher, so you can send data in the function called to process received data. However, you cannot create entities or call functions that are protected by the EA of the **com.rti.dds.domain.DomainParticipant** (p. 629). See Chapter 4 in the RTI Connext User's Manual for complete documentation on Exclusive Areas.

With this QoS policy, you can force a **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478) to share the same EA as its **com.rti.dds.domain.DomainParticipant** (p. 629). Using this capability, the restriction of not being able to create entities in a DataReader Listener's on_data_available() callback is lifted. However, the tradeoff is that the application has reduced concurrency through the Entities that share an EA.

Note that the restrictions on calling methods in a different EA only exist for user code that is called in registered DDS Listeners by internal DomainParticipant threads. User code may call all RTI Connext functions for any DDS Entities from their own threads at any time.

8.92.3 Member Data Documentation

8.92.3.1 boolean use_shared_exclusive_area

Whether the **com.rti.dds.infrastructure.Entity** (p. 912) is protected by its own exclusive area or the shared exclusive area.

All writers belonging to the same **com.rti.dds.publication.Publisher** (p. 1277) are protected by the same exclusive area as the **com.rti.dds.publication.Publisher** (p. 1277) itself. The same is true of all readers belonging to the same **com.rti.dds.subscription.Subscriber** (p. 1478). Typically, the publishers and subscribers themselves do not share their exclusive areas with each other; each has its own. This configuration maximizes the concurrency of the system because independent readers and writers do not need to take the same mutexes in order to operate. However, it places some restrictions on the operations that may be invoked from within listener callbacks because of the possibility of a deadlock. See the **com.rti.dds.infrastructure.Listener** (p. 1154) documentation for more details.

If this field is set to false, the default more concurrent behavior will be used. In the event that this behavior is insufficiently flexible for your application, you may set this value to true. In that case, the **com.rti.dds.subscription.Subscriber** (p. 1478) or **com.rti.dds.publication.Publisher** (p. 1277) in question, and all of the readers or writers (as appropriate) created from it, will share a global exclusive area. This global exclusive area is shared by all entities whose value for this QoS field is true. By sharing the same exclusive area across a larger number of entities, the concurrency of the system will be decreased; however, some of the callback restrictions will be relaxed.

[**default**] false

8.93 FloatSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 938*) that returns a java.lang.Float.*

- Object **set** (int index, Object element)

  A wrapper for ***setFloat()*** *(*p. 939*).*

- void **add** (int index, Object element)

  A wrapper for addFloat(int, int).

8.93.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < float >.

Instantiates:

<<***generic***>> *(*p. 271*)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

float
com.rti.dds.util.Sequence (p. 1432)

8.93.2 Constructor & Destructor Documentation

8.93.2.1 FloatSeq ()

Constructs an empty sequence of floats with an initial maximum of zero.

8.93.2.2 FloatSeq (int *initialMaximum*)

Constructs an empty sequence of floats with the given initial maximum.

8.93.2.3 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

8.93.3 Member Function Documentation

8.93.3.1 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.

8.93.3.2 boolean addAllFloat (float[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.93.3.3 void addFloat (float *element*)

Append the element to the end of the sequence.

8.93.3.4 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.

8.93.3.5 float getFloat (int *index*)

Returns the float at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.93.3.6 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.

8.93.3.7 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.

8.93.3.8 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.

8.93.3.9 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. 941), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.93.3.10 Object get (int *index*) [virtual]

A wrapper for **getFloat(int)** (p. 938) that returns a java.lang.Float.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.93.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setFloat()** (p. 939).

Exceptions:

ClassCastException if the element is not of type Float.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.93.3.12 void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.94 FlowController Interface Reference

<<*interface*>> *(p. 271)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **com.rti.dds.publication.DataWriter** (p. 538) instances are allowed to write data.

Public Member Functions

- String **get_name** ()

  Returns the name of the ***com.rti.dds.publication.FlowController*** *(p. 942).*

- **DomainParticipant get_participant** ()

  Returns the ***com.rti.dds.domain.DomainParticipant*** *(p. 629) to which the* ***com.rti.dds.publication.FlowController*** *(p. 942) belongs.*

- void **set_property** (**FlowControllerProperty_t** prop)

  Sets the ***com.rti.dds.publication.FlowController*** *(p. 942) property.*

- void **get_property** (**FlowControllerProperty_t** prop)

  Gets the ***com.rti.dds.publication.FlowController*** *(p. 942) property.*

- void **trigger_flow** ()

  Provides an external trigger to the ***com.rti.dds.publication.FlowController*** *(p. 942).*

Static Public Attributes

- static final String **DEFAULT_FLOW_CONTROLLER_NAME**

  [**default**] *Special value of* ***com.rti.dds.infrastructure.PublishModeQosPolicy.flow_controller_name*** *(p. 1310) 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. 1310) 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. 1310) that refers to the built-in on-demand flow controller.*

8.94.1 Detailed Description

<<***interface***>> *(p. 271)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **com.rti.dds.publication.DataWriter** (p. 538) instances are allowed to write data.

QoS:

com.rti.dds.publication.FlowControllerProperty_t (p. 946)

8.94.2 Member Function Documentation

8.94.2.1 String get_name ()

Returns the name of the **com.rti.dds.publication.FlowController** (p. 942).

Returns:

The name of the **com.rti.dds.publication.FlowController** (p. 942).

8.94.2.2 DomainParticipant get_participant ()

Returns the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.publication.FlowController** (p. 942) belongs.

Returns:

The **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.publication.FlowController** (p. 942) belongs.

8.94.2.3 void set_property (FlowControllerProperty_t *prop*)

Sets the **com.rti.dds.publication.FlowController** (p. 942) property.

This operation modifies the property of the **com.rti.dds.publication.FlowController** (p. 942).

Once a **com.rti.dds.publication.FlowController** (p. 942) has been instantiated, only the **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947) can be changed. The **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947) is immutable.

A new **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 953) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters:

prop *<<in>> (p. 271)* The new **com.rti.dds.publication.FlowControllerProperty_t** (p. 946). Property must be consistent. Immutable fields cannot be changed after **com.rti.dds.publication.FlowController** (p. 942) has been created. The special value **DomainParticipant.FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 150) can be used to indicate that the property of the **com.rti.dds.publication.FlowController** (p. 942) should be changed to match the current default **com.rti.dds.publication.FlowControllerProperty_t** (p. 946) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.publication.FlowControllerProperty_t (p. 946) for rules on consistency among property values.

8.94.2.4 void get_property (FlowControllerProperty_t *prop*)

Gets the **com.rti.dds.publication.FlowController** (p. 942) property.

Parameters:

prop *<<in>> (p. 271)* **com.rti.dds.publication.FlowController** (p. 942) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.94.2.5 void trigger_flow ()

Provides an external trigger to the **com.rti.dds.publication.FlowController** (p. 942).

Typically, a **com.rti.dds.publication.FlowController** (p. 942) 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. 953) property setting.

This function provides an additional, external trigger to the **com.rti.dds.publication.FlowController** (p. 942). This trigger adds **com.rti.dds.publication.FlowControllerTokenBucketProperty_-t.tokens_added_per_period** (p. 952) tokens each time it is called (subject to the other property settings of the **com.rti.dds.publication.FlowController** (p. 942)).

An *on-demand* **com.rti.dds.publication.FlowController** (p. 942) can be created with a com.rti.dds.infrastructure.Duration_t.INFINITE as **com.rti.dds.publication.FlowControllerTokenBucketProperty_-t.period** (p. 953), in which case the only trigger source is external (i.e. the **com.rti.dds.publication.FlowController** (p. 942) is solely triggered by the user on demand).

com.rti.dds.publication.FlowController.trigger_flow (p. 945) can be called on both strict *on-demand* **com.rti.dds.publication.FlowController** (p. 942) and hybrid **com.rti.dds.publication.FlowController** (p. 942) (internally and externally triggered).

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.95 FlowControllerProperty_t Class Reference

Determines the flow control characteristics of the **com.rti.dds.publication.FlowController** (p. 942).

Inherits Struct.

Public Attributes

- **FlowControllerSchedulingPolicy scheduling_policy**
  Scheduling policy.
- **FlowControllerTokenBucketProperty_t token_bucket**
  Settings for the token bucket.

8.95.1 Detailed Description

Determines the flow control characteristics of the **com.rti.dds.publication.FlowController** (p. 942).

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **com.rti.dds.publication.DataWriter** (p. 538) 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. 942), and does not depend on the number of **com.rti.dds.publication.DataWriter** (p. 538) instances the **com.rti.dds.publication.FlowController** (p. 942) is servicing. For instance, the specified flow rate does *not* double simply because two **com.rti.dds.publication.DataWriter** (p. 538) instances are waiting to write.

Entity:

com.rti.dds.publication.FlowController (p. 942)

Properties:

RxO (p. 97) = N/A

Changeable (p. 98) = **NO** (p. 98) for **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947), **YES** (p. 98) for **com.rti.dds.publication.FlowControllerProperty_t.token_bucket** (p. 947). However, the special value

of com.rti.dds.infrastructure.Duration_t.INFINITE as **com.rti.dds.publication.FlowControllerTokenBucketProperty_-t.period** (p. 953) is strictly used to create an *on-demand* **com.rti.dds.publication.FlowController** (p. 942). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.95.2 Member Data Documentation

8.95.2.1 FlowControllerSchedulingPolicy scheduling_policy

Scheduling policy.

Determines the scheduling policy for servicing the **com.rti.dds.publication.DataWriter** (p. 538) instances associated with the **com.rti.dds.publication.FlowController** (p. 942).

[**default**] idref_FlowControllerSchedulingPolicy_EDF_FLOW_-CONTROLLER_SCHED_POLICY

8.95.2.2 FlowControllerTokenBucketProperty_t token_bucket

Settings for the token bucket.

8.96 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.

8.96.1 Detailed Description

Kinds of flow controller scheduling policy.

Samples written by an asynchronous **com.rti.dds.publication.DataWriter** (p. 538) are not sent in the context of the com.rti.dds.topic.example.FooDataWriter.write call. Instead, the middleware puts the samples in a queue for future processing. The **com.rti.dds.publication.FlowController** (p. 942) associated with each asynchronous **DataWriter** (p. 538) instance determines when the samples are actually sent.

Each **com.rti.dds.publication.FlowController** (p. 942) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **com.rti.dds.publication.DataWriter** (p. 538) 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. 946)

8.96.2 Member Data Documentation

8.96.2.1 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.

8.96.2.2 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. 538) at the time of the write call (as specified in the **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1148)). 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. 942) 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. 1149) setting, an EDF_FLOW_CONTROLLER_SCHED_POLICY **com.rti.dds.publication.FlowController** (p. 942) preserves the order in which the user calls com.rti.dds.topic.example.FooDataWriter.write across the DataWriters associated with the flow controller.

Since the **com.rti.dds.infrastructure.LatencyBudgetQosPolicy** (p. 1148) 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. 1149)

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. 1149) and adhere to the **com.rti.dds.publication.DataWriter** (p. 538) in-order data delivery guarantee.

[default] for **com.rti.dds.publication.DataWriter** (p. 538)

8.97 FlowControllerTokenBucketProperty_t Class Reference

com.rti.dds.publication.FlowController (p. 942) 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.

8.97.1 Detailed Description

com.rti.dds.publication.FlowController (p. 942) 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. 942)

Properties:

RxO (p. 97) = N/A

Changeable (p. 98) = **YES** (p. 98). However, the special value of com.rti.dds.infrastructure.Duration_t.INFINITE as **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.period** (p. 953) is strictly used to create an *on-demand* **com.rti.dds.publication.FlowController** (p. 942). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

8.97.2 Member Data Documentation

8.97.2.1 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. 954), determines the maximum allowable data burst.

Use **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[default] ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.97.2.2 int tokens_added_per_period

The number of tokens added to the token bucket per specified period.

com.rti.dds.publication.FlowController (p. 942) 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. 538) instances based on the **com.rti.dds.publication.FlowControllerProperty_t.scheduling_policy** (p. 947).

Use **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) to add the maximum number of tokens allowed by **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.max_tokens** (p. 952).

[**default**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.97.2.3 int tokens_leaked_per_period

The number of tokens removed from the token bucket per specified period.

com.rti.dds.publication.FlowController (p. 942) 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 **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) 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

8.97.2.4 Duration_t period

Period for adding tokens to and removing tokens from the bucket.

com.rti.dds.publication.FlowController (p. 942) 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.INFINITE can be used to create an *on-demand* **com.rti.dds.publication.FlowController** (p. 942), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **com.rti.dds.publication.FlowController.trigger_flow** (p. 945). This external trigger adds **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.tokens_added_per_period** (p. 952) tokens each time it is called (subject to the other property settings).

[default] 1 second

[range] [0,1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.97.2.5 int bytes_per_token

Maximum number of bytes allowed to send for each token available.

com.rti.dds.publication.FlowController (p. 942) 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. 538). That is, in cases where **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.bytes_per_token** (p. 954) 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. 947)).

Where fragmentation is required, the fragment size will be **com.rti.dds.publication.FlowControllerTokenBucketProperty_t.bytes_per_token** (p. 954) or the minimum largest message size across all transports installed with the **com.rti.dds.publication.DataWriter** (p. 538), whichever is less.

Use **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) 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. 538) to transmit all its queued samples to a single destination. A separate token is required to send to each additional destination.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1024,**ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)]

8.98 Foo Class Reference

A representative user-defined data type.

8.98.1 Detailed Description

A representative user-defined data type.

Foo (p. 955) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 955) is usually defined using IDL syntax and placed in a ".idl" file that is then processed using **rtiddsgen** (p. 290). The **rtiddsgen** (p. 290) utility generates the helper classes **com.rti.dds.util.Sequence** (p. 1432) 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.dds.topic.example.FooDataReader and com.rti.dds.topic.example.FooDataWriter types that allow the application to send and receive data of this type.

See also:

com.rti.dds.util.Sequence (p. 1432), com.rti.dds.topic.example.FooDataWriter, com.rti.dds.topic.example.FooDataReader, **com.rti.dds.topic.example.FooTypeSupport** (p. 1060), **rtiddsgen** (p. 290)

8.99 Foo Class Reference

A representative user-defined data type.

Inheritance diagram for Foo::

Public Member Functions

- Object **copy_from** (Object src)

8.99.1 Detailed Description

A representative user-defined data type.

Foo (p. 956) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 956) is usually defined using IDL syntax and placed in a ".idl" file that is then processed using **rtiddsgen** (p. 290). The **rtiddsgen** (p. 290) utility generates the helper classes **com.rti.dds.util.Sequence** (p. 1432) 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.dds.topic.example.FooDataReader and com.rti.dds.topic.example.FooDataWriter types that allow the application to send and receive data of this type.

See also:

com.rti.dds.util.Sequence (p. 1432), com.rti.dds.topic.example.FooDataWriter, com.rti.dds.topic.example.FooDataReader, **com.rti.dds.topic.example.FooTypeSupport** (p. 1060), **rtiddsgen** (p. 290)

8.99.2 Member Function Documentation

8.99.2.1 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. 1063).

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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.100 FooDataReader Class Reference

*<<**interface**>>* *(p. 271)* *<<**generic**>>* *(p. 271)* 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. 473).*

- 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. 473).*

- void **read_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  *Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **take_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take 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. 473).*

- 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. 473).*

- 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. 473).*

^ 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. 473).*

^ 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. 473).*

^ 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. 473).*

^ 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

^ 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

^ void **return_loan** (**FooSeq** received_data, **SampleInfoSeq** info_seq)

*Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).*

^ 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`.

8.100.1 Detailed Description

<<***interface***>> *(p. 271)* <<***generic***>> *(p. 271)* 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. 473)
Foo (p. 956)
com.rti.dds.topic.example.FooDataWriter
rtiddsgen (p. 290)

8.100.2 Member Function Documentation

8.100.2.1 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. 473).

This operation offers the same functionality and API as com.rti.dds.topic.example.FooDataReader.take except that the samples returned remain in the **com.rti.dds.subscription.DataReader** (p. 473) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of com.rti.dds.topic.example.FooDataReader.take() 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 **SampleStateKind.READ_SAMPLE_STATE** (p. 1430). If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to be **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). It will not affect the `instance_state` of the instance.

Important: If the samples "returned" by this method are loaned from RTI Connext (see com.rti.dds.topic.example.FooDataReader.take 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. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching ones of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.2.2 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. 473).

The operation will return the list of samples received by the **com.rti.dds.subscription.DataReader** (p. 473) since the last com.rti.dds.topic.example.FooDataReader.take operation that match the specified com.rti.dds.subscription.SampleStateMask, com.rti.dds.subscription.ViewStateMask and com.rti.dds.subscription.InstanceStateMask.

This operation may fail with RETCODE_ERROR if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_outstanding_reads** (p. 530) 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. 1071), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356), **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524) and the characteristics of the data-type that is associated with the **com.rti.dds.subscription.DataReader** (p. 473):

- In the case where the **com.rti.dds.infrastructure.HistoryQosPolicy.kind** (p. 1073) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, the call will return at most **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074) samples per instance.

- The maximum number of samples returned is limited by **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359), and by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolic samples_per_read** (p. 530).

- For multiple instances, the number of samples returned is additionally limited by the product (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360) ∗ **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.ma instances** (p. 1360))

- If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_infos** (p. 528) is limited, the number of samples returned may also be limited if insufficient **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) 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 insuffificient resources, in order to access remaining samples.

Note that in the case where the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473). 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 **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). It will not affect the `instance_state` of the sample's instance.

After com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.take 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. 473). 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. 473) by calling the com.rti.dds.topic.example.FooDataReader.return_loan 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. 1404) objects after the call to com.rti.dds.topic.example.FooDataReader.return_loan. 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.dds.topic.example.FooDataReader.return_loan at some point, you do *not* have to do so before the next com.rti.dds.topic.example.FooDataReader.take call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy**

(p. 524).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when com.rti.dds.topic.example.FooDataReader.take is called, samples are copied to received_data and info_seq. The application will not need to call com.rti.dds.topic.example.FooDataReader.return_loan.

The order of the samples returned to the caller depends on the **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1237).

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is 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. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection is a list were the relative order of samples is preserved also accross different instances. Note that samples belonging to the same instance may or may not be consecutive. This is because to preserve order it may be necessary to mix samples from different instances.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive. [**Not supported (optional)**]

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection contains at most one sample. The difference in this case is due to the fact that is

required that the application is able to read samples belonging to different **com.rti.dds.subscription.DataReader** (p. 473) objects in a specific order. [**Not supported (optional)**]

In any case, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 51) policy:

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 609) is 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. 609) is 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

In addition to the collection of samples, the read and take operations also use a collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures.

8.100.3 SEQUENCES USAGE IN TAKE AND READ

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 Sequence.size())
- the maximum length (`max_len`, see **Sequence.getMaximum** (p. 1433))

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 RETCODE_PRECONDITION_NOT_MET.
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. 473). 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. 538) using com.rti.dds.topic.example.FooDataReader.return_loan.

4. If the initial `max_len>0` then the read or take operation will fail with RETCODE_PRECONDITION_NOT_MET. This avoids the potential hard-to-detect memory leaks caused by an application forgetting to return the loan.

5. If initial `max_len>0` then the read or take operation will copy the `received_data` values and **com.rti.dds.subscription.SampleInfo** (p. 1404) 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 RETCODE_PRECONDITION_NOT_MET. 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. 473), 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. 473). If this is the case, the application will need to use com.rti.dds.topic.example.FooDataReader.return_loan to return the loan once it is no longer using the `received_data` in the collection. When com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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 **Foo** (p. 956) values and the collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures are of the same length and are in a one-to-one correspondence. Each **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) is **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) or **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088), then the last sample for that instance in the collection (that is, the one whose **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1356). The act of reading/taking a sample sets its sample_state to **SampleStateKind.READ_SAMPLE_STATE** (p. 1430).

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). 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 (**Foo** (p. 956)). If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operations fails with RETCODE_NO_DATA.

For an **example** (p. 366) on how `take` can be used, please refer to the **receive example** (p. 246).

Parameters:

received_data *<<inout>>* *(p. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Parameters:

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER

if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described above.

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching one of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.1 void read_w_condition (FooSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

In case the **com.rti.dds.subscription.ReadCondition** (p. 1326) is a plain **com.rti.dds.subscription.ReadCondition** (p. 1326) and not the specialized **com.rti.dds.subscription.QueryCondition** (p. 1324), the operation is equivalent to calling com.rti.dds.topic.example.FooDataReader.read 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. 1326).

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.read.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operation will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.take,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.2 void take_w_condition (FooSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

This operation is analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except that it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.take.

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_-

PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.3 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473). This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). The implied order among the samples stored in the **com.rti.dds.subscription.DataReader** (p. 473) is the same as for the com.rti.dds.topic.example.FooDataReader.read operation.

The com.rti.dds.topic.example.FooDataReader.read_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<**inout**>> (p. 271)* user data type-specific **Foo** (p. 956) 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. 956). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<**inout**>> (p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo**

(p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read

8.100.3.4 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' it from the **com.rti.dds.subscription.DataReader** (p. 473) so that it is no longer accessible. This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). This operation is analogous to the com.rti.dds.topic.example.FooDataReader.read_next_sample except for the fact that the sample is removed from the **com.rti.dds.subscription.DataReader** (p. 473).

The com.rti.dds.topic.example.FooDataReader.take_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<inout>> (p. 271)* user data type-specific **Foo** (p. 956) 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. 956). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take

8.100.3.5 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read, 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. 1404) verifies **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1410) == `a_handle`.

The com.rti.dds.topic.example.FooDataReader.read_instance operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.read_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_-

PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.6 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.take, 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.dds.topic.example.FooDataReader.take operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.take_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method fails with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL. The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.7 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) where all the samples belong to a single instance. The behavior is similar to com.rti.dds.topic.example.FooDataReader.read_instance, 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. 473). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of com.rti.dds.topic.example.FooDataReader.read_next_instance is 'as if' the **com.rti.dds.subscription.DataReader** (p. 473) invoked com.rti.dds.topic.example.FooDataReader.read_instance, 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

The operation com.rti.dds.topic.example.FooDataReader.read_next_instance is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082), examines the samples returned, and then uses the `instance_handle` returned in the **com.rti.dds.subscription.SampleInfo** (p. 1404) as the value of the `previous_handle` argument to the next call to com.rti.dds.topic.example.FooDataReader.read_next_instance. The iteration continues until com.rti.dds.topic.example.FooDataReader.read_next_instance

fails with the value RETCODE_NO_DATA.

Note that it is possible to call the com.rti.dds.topic.example.FooDataReader.read_next_instance operation with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473). 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. 473). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088) 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.dds.topic.example.FooDataReader.read_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.8 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

This operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.9 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read_next_instance, 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. 1326) evaluates to TRUE.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.10 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the

com.rti.dds.subscription.ReadCondition (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

The operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples

will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, or RETCODE_NO_DATA, RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.100.3.11 void return_loan (FooSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

This operation indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

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. 473) to which they are returned. If either of these conditions is not met, the operation will fail with RETCODE_PRECONDITION_NOT_MET.

The operation com.rti.dds.topic.example.FooDataReader.return_loan allows implementations of the read and take operations to "loan" buffers from the **com.rti.dds.subscription.DataReader** (p. 473) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), the application should not retain them indefinitely.

The use of com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL **FooSeq** (p. 1058). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Parameters:

info_seq *<<in>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

8.100.3.12 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.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

key_holder *<<inout>> (p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 956) has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<in>> (p. 271)* the `instance` whose key is to be retrieved. If **Foo** (p. 956) *has* a key, `handle` must represent an existing instance of type **Foo** (p. 956) known to the **com.rti.dds.subscription.DataReader** (p. 473). Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If **Foo** (p. 956) *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. If **Foo** (p. 956) has a key and `handle` represents an instance of another type or an instance of type **Foo** (p. 956) that has been unregistered, this method will fail with RETCODE_BAD_PARAMETER. If **Foo** (p. 956) has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.100.3.13 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. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value HANDLE_NIL.

Parameters:

key_holder *<<in>>* *(p. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If **Foo** (p. 956) has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.101 FooDataReader Interface Reference

<<*interface*>> *(p. 271)* <<*generic*>> *(p. 271)* User data type-specific data reader.

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. 473).*

- void **take** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, int instance_states, int sample_states, int view_states)

  Access a collection of data-samples from the ***com.rti.dds.subscription.DataReader*** *(p. 473).*

- void **read_w_condition** (**FooSeq** received_data, **SampleInfo** info_seq, int max_samples, **ReadCondition** condition)

  Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

- void **take_w_condition** (**FooSeq** received_data, **SampleInfo** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take 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. 473).*

- 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. 473).*

- 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. 473).*

- 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. 473).*

- void **read_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.topic.example.FooDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **take_instance_w_condition** (**FooSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** condition)

  *Accesses via com.rti.dds.topic.example.FooDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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. 473).*

- 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. 473).*

- 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- void **return_loan** (**FooSeq** received_data, **SampleInfoSeq** info_seq)

  Indicates to the ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 473).*

- 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`*.*

8.101.1 Detailed Description

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* 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. 473)
Foo
com.rti.dds.topic.example.FooDataWriter
rtiddsgen (p. 290)

8.101.2 Member Function Documentation

8.101.2.1 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. 473).

This operation offers the same functionality and API as com.rti.dds.topic.example.FooDataReader.take except that the samples returned remain in the **com.rti.dds.subscription.DataReader** (p. 473) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of com.rti.dds.topic.example.FooDataReader.take() 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 **SampleStateKind.READ_SAMPLE_STATE** (p. 1430). If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to be **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). It will not affect the `instance_state` of the instance.

Important: If the samples "returned" by this method are loaned from RTI Connext (see com.rti.dds.topic.example.FooDataReader.take 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. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching ones of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.2.2 void take (FooSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, int *instance_states*, int *sample_states*, int *view_states*)

Access a collection of data-samples from the **com.rti.dds.subscription.DataReader** (p. 473).

The operation will return the list of samples received by the **com.rti.dds.subscription.DataReader** (p. 473) since the last com.rti.dds.topic.example.FooDataReader.take operation that match the specified com.rti.dds.subscription.SampleStateMask, com.rti.dds.subscription.ViewStateMask and com.rti.dds.subscription.InstanceStateMask.

This operation may fail with RETCODE_ERROR if **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_outstanding_reads** (p. 530) 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. 1071), **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356), **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524) and the characteristics of the data-type that is associated with the **com.rti.dds.subscription.DataReader** (p. 473):

- In the case where the **com.rti.dds.infrastructure.HistoryQosPolicy.kind** (p. 1073) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS, the call will return at most **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074) samples per instance.

- The maximum number of samples returned is limited by **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359), and by **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolic samples_per_read** (p. 530).

- For multiple instances, the number of samples returned is additionally limited by the product

(**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360) * **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360))

^ If **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_infos** (p. 528) is limited, the number of samples returned may also be limited if insufficient **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) 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 insuffificient resources, in order to access remaining samples.

Note that in the case where the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.subscription.DataReader** (p. 473) is bound to a datatype that has no key definition, then there will be at most one instance in the **com.rti.dds.subscription.DataReader** (p. 473). 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 **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). It will not affect the `instance_state` of the sample's instance.

After com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.take 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. 473). 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. 473) by calling the com.rti.dds.topic.example.FooDataReader.return_loan 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. 1404) objects after the call to com.rti.dds.topic.example.FooDataReader.return_loan. 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.dds.topic.example.FooDataReader.return_loan at some point, you do *not* have to do so before the next com.rti.dds.topic.example.FooDataReader.take call. However, failure to return the loan will eventually deplete the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) is specified by the **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356) and the **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy** (p. 524).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when com.rti.dds.topic.example.FooDataReader.take is called, samples are copied to received_data and info_seq. The application will not need to call com.rti.dds.topic.example.FooDataReader.return_loan.

The order of the samples returned to the caller depends on the **com.rti.dds.infrastructure.PresentationQosPolicy** (p. 1237).

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is 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. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection is a list were the relative order of samples is preserved also accross different instances. Note that samples belonging to the same instance may or may not be consecutive. This is because to preserve order it may be necessary to mix samples from different instances.

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to false, then returned collection is a list where

samples belonging to the same data instance are consecutive. [**Not supported (optional)**]

- If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is set to true, then the returned collection contains at most one sample. The difference in this case is due to the fact that is required that the application is able to read samples belonging to different **com.rti.dds.subscription.DataReader** (p. 473) objects in a specific order. [**Not supported (optional)**]

In any case, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 51) policy:

- If **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 609) is 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. 609) is 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

In addition to the collection of samples, the read and take operations also use a collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures.

8.101.3 SEQUENCES USAGE IN TAKE AND READ

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 Sequence.size())
- the maximum length (`max_len`, see **Sequence.getMaximum** (p. 1433))

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 RETCODE_PRECONDITION_NOT_MET.

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. 473). 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. 538) using com.rti.dds.topic.example.FooDataReader.return_loan.

4. If the initial `max_len>0` then the read or take operation will fail with RETCODE_PRECONDITION_NOT_MET. This avoids the potential hard-to-detect memory leaks caused by an application forgetting to return the loan.

5. If initial `max_len>0` then the read or take operation will copy the `received_data` values and **com.rti.dds.subscription.SampleInfo** (p. 1404) 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 RETCODE_PRECONDITION_NOT_MET. 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. 473), 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. 473). If this is the case, the application will need to use com.rti.dds.topic.example.FooDataReader.return_loan to return the loan once it is no longer using the `received_data` in the collection. When com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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 Foo values and the collection of **com.rti.dds.subscription.SampleInfo** (p. 1404) structures are of the same length and are in a one-to-one correspondence. Each **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1404) is **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) or **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088), then the last sample for that instance in the collection (that is, the one whose **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1356). The act of reading/taking a sample sets its sample_state to **SampleStateKind.READ_SAMPLE_STATE** (p. 1430).

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690). 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 (Foo). If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operations fails with RETCODE_NO_DATA.

For an **example** (p. 349) on how `take` can be used, please refer to the **receive example** (p. 246).

Parameters:

received_data *<<inout>>* *(p. 271)* User data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received

data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Parameters:

info_seq *<<inout>>* *(p. 271)* A **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described above.

sample_states *<<in>>* *(p. 271)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 271)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 271)* Data samples matching one of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.1 void read_w_condition (FooSeq *received_data*, SampleInfo *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via com.rti.dds.topic.example.FooDataReader.read the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

In case the **com.rti.dds.subscription.ReadCondition** (p. 1326) is a plain **com.rti.dds.subscription.ReadCondition** (p. 1326) and not the specialized **com.rti.dds.subscription.QueryCondition** (p. 1324), the operation is equivalent to calling com.rti.dds.topic.example.FooDataReader.read 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. 1326).

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.read.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the operation will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read

com.rti.dds.topic.example.FooDataReader.take,
com.rti.dds.topic.example.FooDataReader.take_w_condition
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.2 void take_w_condition (FooSeq *received_data*, SampleInfo *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

This operation is analogous to com.rti.dds.topic.example.FooDataReader.read_w_condition except that it accesses samples via the com.rti.dds.topic.example.FooDataReader.take operation.

The specified **com.rti.dds.subscription.ReadCondition** (p. 1326) must be attached to the **com.rti.dds.subscription.DataReader** (p. 473); otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

The samples are accessed with the same semantics as com.rti.dds.topic.example.FooDataReader.take.

This operation is especially useful in combination with **com.rti.dds.subscription.QueryCondition** (p. 1324) to filter data samples based on the content.

If the **com.rti.dds.subscription.DataReader** (p. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition,
com.rti.dds.topic.example.FooDataReader.read
com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.3 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473). This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). The implied order among the samples stored in the **com.rti.dds.subscription.DataReader** (p. 473) is the same as for the com.rti.dds.topic.example.FooDataReader.read operation.

The com.rti.dds.topic.example.FooDataReader.read_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific Foo 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 RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read

8.101.3.4 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. 473).

This operation copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' it from the **com.rti.dds.subscription.DataReader** (p. 473) so that it is no longer accessible. This operation also copies the corresponding **com.rti.dds.subscription.SampleInfo** (p. 1404). This operation is analogous to the com.rti.dds.topic.example.FooDataReader.read_next_sample except for the fact that the sample is removed from the **com.rti.dds.subscription.DataReader** (p. 473).

The com.rti.dds.topic.example.FooDataReader.take_next_sample operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.take 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.dds.topic.example.FooDataReader.read_next_sample 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. 473), the operation will fail with RETCODE_NO_DATA and nothing is copied.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific Foo 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 RETCODE_BAD_PARAMETER if it is NULL.

sample_info *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfo** (p. 1404) object where the next received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfo** (p. 1404). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take

8.101.3.5 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read, 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. 1404) verifies **com.rti.dds.subscription.SampleInfo.instance_handle** (p. 1410) == `a_handle`.

The com.rti.dds.topic.example.FooDataReader.read_instance operation is semantically equivalent to the com.rti.dds.topic.example.FooDataReader.read operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.read_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the

pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>> (p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.6 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.take, 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.dds.topic.example.FooDataReader.take operation, except in building the collection, the **com.rti.dds.subscription.DataReader** (p. 473) 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.dds.topic.example.FooDataReader.take_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method fails with RETCODE_NO_DATA.

This operation may fail with RETCODE_BAD_PARAMETER if the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) `a_handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

a_handle *<<in>>* *(p. 271)* The specified instance to return samples for. Must be a valid non-NULL **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL. The method will fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.7 void read_instance_w_condition (FooSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *previous_handle*, ReadCondition *condition*)

Accesses via com.rti.dds.topic.example.FooDataReader.read_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read_instance, 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. 1326) evaluates to TRUE.

The behavior of the com.rti.dds.topic.example.FooDataReader.read_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples

will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.8 void take_instance_w_condition (FooSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, InstanceHandle_t *previous_handle*, ReadCondition *condition*)

Accesses via com.rti.dds.topic.example.FooDataReader.take_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.take_instance, 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. 1326) evaluates to TRUE.

The operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_instance_w_condition, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

The behavior of the com.rti.dds.topic.example.FooDataReader.take_instance_w_condition operation follows the same rules as the

com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, or RETCODE_NO_DATA, RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance

ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.9 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) where all the samples belong to a single instance. The behavior is similar to com.rti.dds.topic.example.FooDataReader.read_instance, 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. 473). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of com.rti.dds.topic.example.FooDataReader.read_next_instance is 'as if' the **com.rti.dds.subscription.DataReader** (p. 473) invoked com.rti.dds.topic.example.FooDataReader.read_instance, 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

The operation com.rti.dds.topic.example.FooDataReader.read_next_instance is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **InstanceHandle_t.HANDLE_NIL** (p. 1082), examines the samples returned, and then uses the `instance_handle` returned in the **com.rti.dds.subscription.SampleInfo** (p. 1404)

as the value of the `previous_handle` argument to the next call to com.rti.dds.topic.example.FooDataReader.read_next_instance. The iteration continues until com.rti.dds.topic.example.FooDataReader.read_next_instance fails with the value RETCODE_NO_DATA.

Note that it is possible to call the com.rti.dds.topic.example.FooDataReader.read_next_instance operation with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473). 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. 473). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088) 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.dds.topic.example.FooDataReader.read_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples

will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.10 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. 473).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

This operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

sample_states *<<in>>* *(p. 271)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 271)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 271)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.11 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.dds.topic.example.FooDataReader.read_next_instance the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473). The behavior is identical to com.rti.dds.topic.example.FooDataReader.read_next_instance, 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. 1326) evaluates to TRUE.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET RETCODE_NO_DATA or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.12 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.dds.topic.example.FooDataReader.take_next_instance the samples that match the criteria specified in the

com.rti.dds.subscription.ReadCondition (p. 1326).

This operation accesses a collection of data values from the **com.rti.dds.subscription.DataReader** (p. 473) and 'removes' them from the **com.rti.dds.subscription.DataReader** (p. 473).

The operation has the same behavior as com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition, except that the samples are 'taken' from the **com.rti.dds.subscription.DataReader** (p. 473) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation com.rti.dds.topic.example.FooDataReader.read_next_instance, it is possible to call com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition with a `previous_handle` that does not correspond to an instance currently managed by the **com.rti.dds.subscription.DataReader** (p. 473).

The behavior of the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation follows the same rules as the com.rti.dds.topic.example.FooDataReader.read operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to com.rti.dds.topic.example.FooDataReader.read, the com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition operation may 'loan' elements to the output collections, which must then be returned by means of com.rti.dds.topic.example.FooDataReader.return_loan.

Similar to the com.rti.dds.topic.example.FooDataReader.read, 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. 473) has no samples that meet the constraints, the method will fail with RETCODE_NO_DATA.

Parameters:

received_data *<<inout>>* *(p. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples will be returned. Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

info_seq *<<inout>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info will be returned. Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

max_samples *<<in>>* *(p. 271)* The maximum number of samples to be returned. If the special value **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) is provided, as many samples

will be returned as are available, up to the limits described in the documentation for com.rti.dds.topic.example.FooDataReader.take().

previous_handle *<<in>>* *(p. 271)* 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. 1080). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

condition *<<in>>* *(p. 271)* the **com.rti.dds.subscription.ReadCondition** (p. 1326) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET, or RETCODE_NO_DATA, RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance
ResourceLimitsQosPolicy.LENGTH_UNLIMITED (p. 102)

8.101.3.13 void return_loan (FooSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

This operation indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

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. 473) to which they are returned. If either of these conditions is not met, the operation will fail with RETCODE_PRECONDITION_NOT_MET.

The operation com.rti.dds.topic.example.FooDataReader.return_loan allows implementations of the read and take operations to "loan" buffers from the **com.rti.dds.subscription.DataReader** (p. 473) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), the application should not retain them indefinitely.

The use of com.rti.dds.topic.example.FooDataReader.return_loan 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.dds.topic.example.FooDataReader.return_loan, 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. 271)* user data type-specific **com.rti.dds.util.Sequence** (p. 1432) object where the received data samples was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL FooSeq. The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Parameters:

info_seq *<<in>>* *(p. 271)* a **com.rti.dds.subscription.SampleInfoSeq** (p. 1414) object where the received sample info was obtained from earlier invocation of read or take on the **com.rti.dds.subscription.DataReader** (p. 473). Must be a valid non-NULL **com.rti.dds.subscription.SampleInfoSeq** (p. 1414). The method will fail with RETCODE_BAD_PARAMETER if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

8.101.3.14 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.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.subscription.DataReader** (p. 473).

Parameters:

key_holder *<<inout>>* *(p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<in>>* *(p. 271)* the `instance` whose key is to be retrieved. If Foo *has* a key, `handle` must represent an existing instance of type Foo known to the **com.rti.dds.subscription.DataReader** (p. 473). Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. If Foo has a key and `handle` represents an instance of another type or an instance of type Foo that has been unregistered, this method will fail with RETCODE_BAD_PARAMETER. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.101.3.15 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. This operation does not register the instance in question. If the instance has not been previously registered, or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value HANDLE_NIL.

Parameters:

key_holder <<*in*>> *(p. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If Foo has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.102 FooDataWriter Class Reference

<<***interface***>> *(p. 271)* <<***generic***>> *(p. 271)* 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.dds.topic.example.FooDataWriter.register_instance and com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp 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.dds.topic.example.FooDataWriter.register_instance.

- void **unregister_instance_w_timestamp** (**Foo** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance 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.dds.topic.example.FooDataWriter.unregister_instance and ***FooDataWriter.unregister_instance_w_timestamp*** *(p. 1028) 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.dds.topic.example.FooDataWriter.write 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.dds.topic.example.FooDataWriter.write and com.rti.dds.topic.example.FooDataWriter.write_w_timestamp 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 data.

- 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. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

- void **dispose_w_params** (**Foo** instance_data, **WriteParams_t** params)

  Performs the same function as com.rti.dds.topic.example.FooDataWriter.dispose and com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp 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`.

8.102.1 Detailed Description

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* 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. 538)
Foo (p. 956)
com.rti.dds.topic.example.FooDataReader
rtiddsgen (p. 290)

8.102.2 Member Function Documentation

8.102.2.1 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082). 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. 1029) or **dispose()** (p. 1034) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) as the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) paramater 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.dds.topic.example.FooDataWriter.write, com.rti.dds.topic.example.FooDataWriter.write_w_timestamp, com.rti.dds.topic.example.FooDataWriter.dispose and com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) 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. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL**

(p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance, com.rti.dds.topic.example.FooDataWriter.get_key_value, **RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP** (p. 1218)

8.102.2.2 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. 51) QoS policy for details.

This operation may fail and return **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) limit has been exceeded.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL** (p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance,
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.102.2.3 InstanceHandle_t register_instance_w_params (Foo *instance_data*, WriteParams_t *params*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.register_instance and com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp except that it also provides the values contained in `params`.

8.102.2.4 void unregister_instance (Foo *instance_data*, InstanceHandle_t *handle*)

Reverses the action of com.rti.dds.topic.example.FooDataWriter.register_instance.

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.dds.topic.example.FooDataWriter.register_instance. Otherwise, this operation may fail with RETCODE_BAD_PARAMETER.

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. 538) 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 com.rti.dds.topic.example.FooDataWriter.unregister_instance().

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 1025) operation is for the instance as indicated by the `handle`.

If after a com.rti.dds.topic.example.FooDataWriter.unregister_instance, the application wants to modify (com.rti.dds.topic.example.FooDataWriter.write or com.rti.dds.topic.example.FooDataWriter.dispose) an instance, it has to register it again, or else use the special `handle` value **InstanceHandle_t.HANDLE_NIL** (p. 1082).

This operation does not indicate that the instance is deleted (that is the purpose of com.rti.dds.topic.example.FooDataWriter.dispose). The operation com.rti.dds.topic.example.FooDataWriter.unregister_instance just indicates that the **com.rti.dds.publication.DataWriter** (p. 538) no longer has anything to say about the instance. **com.rti.dds.subscription.DataReader** (p. 473) entities that are reading the instance may receive a sample with **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088) for the instance, unless there are other **com.rti.dds.publication.DataWriter** (p. 538) objects writing that same instance.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 83)). If the **com.rti.dds.publication.DataWriter** (p. 538) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1025) will relinquish that ownership.

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339)

is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS 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. 1339); if this writer is still unable to unregister after that period, this method will fail with RETCODE_TIMEOUT.

Parameters:

instance_data *<<**in**>>* *(p. 271)* The instance that should be unregistered. If **Foo** (p. 956) *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER . If **Foo** (p. 956) *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<**in**>>* *(p. 271)* represents the `instance` to be unregistered. If **Foo** (p. 956) *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 956) 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 RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If **Foo** (p. 956) *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will report the error RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
FooDataWriter.unregister_instance_w_timestamp (p. 1028)
com.rti.dds.topic.example.FooDataWriter.get_key_value
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.102.2.5 void unregister_instance_w_timestamp (Foo *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance 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. 51) 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.dds.topic.example.FooDataWriter.unregister_instance operation.

This operation may block and may time out (RETCODE_TIMEOUT) under the same circumtances described for the unregister_instance operation.

Parameters:

instance_data *<<in>> (p. 271)* The instance that should be unregistered. If **Foo** (p. 956) *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER. If **Foo** (p. 956) *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<in>> (p. 271)* represents the `instance` to be unregistered. If **Foo** (p. 956) *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 956) 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 RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If **Foo** (p. 956) *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<in>> (p. 271)* 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. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
com.rti.dds.topic.example.FooDataWriter.unregister_instance
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.102.2.6 void unregister_instance_w_params (Foo *instance_data*, WriteParams_t *params*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance and **FooDataWriter.unregister_instance_w_timestamp** (p. 1028) except that it also provides the values contained in `params`.

8.102.2.7 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) QoS policy for details).

As a side effect, this operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 538) itself, the **com.rti.dds.publication.Publisher** (p. 1277) and the **com.rti.dds.domain.DomainParticipant** (p. 629).

Note that the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 1029) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 101) `kind` is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 102) to be exceeded.

Specifically, this operation may block in the following situations (note that the list may not be exhaustive), even if its **com.rti.dds.infrastructure.HistoryQosPolicyKind** (p. 1075) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS:

- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) * **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074)), then in the situation where the `max_samples` resource limit is exhausted, RTI Connext is allowed to discard samples of some other instance, as long as at least one sample remains for such an instance. If it is still not possible to make space available to store the modification, the writer is allowed to block.
- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360)), then the DataWriter may block regardless of the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074).
- If (**com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1389) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy**
samples (p. 1359)), then it is possible for the send_window_size limit to be reached before RTI Connext is allowed to discard samples, in which case the **com.rti.dds.publication.DataWriter** (p. 538) will block.

This operation may also block when using ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS and PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS. In this case, the **com.rti.dds.publication.DataWriter** (p. 538) will queue samples until they are sent by the asynchronous publishing thread. The number of samples that can be stored is determined by the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071). If the asynchronous thread does not send samples fast enough (e.g., when using a slow **com.rti.dds.publication.FlowController** (p. 942)), the queue may fill up. In that case, subsequent write calls will block.

If this operation *does* block for any of the above reasons, the **RELIABILITY** (p. 101) `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 **com.rti.dds.publication.DataWriter** (p. 538) is able to

store the modification without exceeding the limits, the operation will time out (RETCODE_TIMEOUT).

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

This operation will fail with RETCODE_PRECONDITION_NOT_MET 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).

Parameters:

instance_data *<<**in**>>* *(p. 271)* The data to write.

This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

Parameters:

handle *<<**in**>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If **Foo** (p. 956) *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type **Foo** (p. 956). Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_PRECONDITION_NOT_MET, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.subscription.DataReader (p. 473)
com.rti.dds.topic.example.FooDataWriter.write_w_timestamp
DESTINATION_ORDER (p. 51)

8.102.2.8 void write_w_timestamp (Foo *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.write 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) 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.dds.topic.example.FooDataWriter.write operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help free up some resources.

This operation may fail with RETCODE_BAD_PARAMETER under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>>* *(p. 271)* The data to write. This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If **Foo** (p. 956) *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type **Foo** (p. 956). Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

source_timestamp *<<in>>* *(p. 271)* When using 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. 609), 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. 609), the function will return RETCODE_-BAD_PARAMETER.

Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_-ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.write
com.rti.dds.subscription.DataReader (p. 473)
DESTINATION_ORDER (p. 51)

8.102.2.9 void write_w_params (Foo *instance_data*, WriteParams_t *params*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.write and com.rti.dds.topic.example.FooDataWriter.write_w_timestamp except that it also provides the values contained in `params`.

Allows provision of the instance handle, source timestamp, publication priority, and cookie, in `params`.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.write operation.

The cookie is a sequence of bytes tagging the data being written, and is used to retrieve the data when it is not available to the writer when needed.

This operation may block and time out (RETCODE_-TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance_w_params may help free up some resources.

This operation may fail with RETCODE_BAD_PARAMETER under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>> (p. 271)* The data to write. This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

params *<<in>>* *(p. 271)*

The handle is either returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If **Foo** (p. 956) *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type **Foo** (p. 956). Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

The source_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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.write
com.rti.dds.subscription.DataReader (p. 473)
DESTINATION_ORDER (p. 51)

8.102.2.10 void dispose (Foo *instance_data*, InstanceHandle_t *instance_handle*)

Requests the middleware to delete the data.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

The actual deletion is postponed until there is no more use for that data in the whole system.

Applications are made aware of the deletion by means of operations on the **com.rti.dds.subscription.DataReader** (p. 473) objects that already knew that instance. **com.rti.dds.subscription.DataReader** (p. 473) objects that didn't know the instance will never see it.

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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.unregister_instance operation.

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 **dispose()** (p. 1034) operation is for the instance as indicated by the `handle`.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write().

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 271)* The data to dispose. If **Foo** (p. 956) *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 956) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_**

NIL (p. 1082). If **Foo** (p. 956) *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 956) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 956). Otherwise, this method fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If **Foo** (p. 956) *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.102.2.11 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.dispose operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 271)* The data to dispose. If **Foo** (p. 956)

has a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 956) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>> (p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If **Foo** (p. 956) *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 956) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 956). Otherwise, this method may fail with RETCODE_BAD_PARAMETER This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If **Foo** (p. 956) *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<in>> (p. 271)* 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.102.2.12 void dispose_w_params (Foo *instance_data*, WriteParams_t *params*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.dispose and com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp except that it also provides the values contained in `params`.

8.102.2.13 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 **Foo** (p. 956) has no key, this method has no effect and exit with no error.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

key_holder *<<inout>> (p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 956) has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<in>> (p. 271)* the `instance` whose key is to be retrieved. If **Foo** (p. 956) *has* a key, `handle` must represent a registered instance of type **Foo** (p. 956). Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If **Foo** (p. 956) *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.102.2.14 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. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If **Foo** (p. 956) has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.103 FooDataWriter Interface Reference

*<<**interface**>>* *(p. 271)* *<<**generic**>>* *(p. 271)* User data type specific data writer.

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`.

- void **unregister_instance** (**Foo** instance_data, **InstanceHandle_t** handle)

  Reverses the action of com.rti.dds.topic.example.FooDataWriter.register_instance.

- void **unregister_instance_w_timestamp** (**Foo** instance_data, **InstanceHandle_t** handle, **Time_t** source_timestamp)

  Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance except that it also provides the value for the `source_timestamp`.

- 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.dds.topic.example.FooDataWriter.write except that it also provides the value for the `source_timestamp`.

- void **dispose** (**Foo** instance_data, **InstanceHandle_t** instance_handle)

  Requests the middleware to delete the data.

- 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. 473)* *objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404)*.

- 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`.

8.103.1 Detailed Description

<<***interface***>> *(p. 271)* <<***generic***>> *(p. 271)* 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. 538)
Foo
com.rti.dds.topic.example.FooDataReader
rtiddsgen (p. 290)

8.103.2 Member Function Documentation

8.103.2.1 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082). 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. 1047) or **dispose()** (p. 1051) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) as the **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) paramater 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.dds.topic.example.FooDataWriter.write, com.rti.dds.topic.example.FooDataWriter.write_w_timestamp, com.rti.dds.topic.example.FooDataWriter.dispose and com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) 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. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL** (p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance, com.rti.dds.topic.example.FooDataWriter.get_key_value, **RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP** (p. 1218)

8.103.2.2 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. 51) QoS policy for details.

This operation may fail and return **InstanceHandle_t.HANDLE_NIL** (p. 1082) if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) limit has been exceeded.

This operation can only be called after **com.rti.dds.publication.DataWriter** (p. 538) has been enabled. Otherwise, **InstanceHandle_t.HANDLE_NIL** (p. 1082) will be returned.

Parameters:

instance_data *<<in>> (p. 271)* 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. 271)* 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. 1080), such as write, dispose, unregister_instance, or return **InstanceHandle_t.HANDLE_NIL** (p. 1082) on failure. If the instance_data is of a data type that has no keys, this function always return **InstanceHandle_t.HANDLE_NIL** (p. 1082).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance,
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.103.2.3 void unregister_instance (Foo *instance_data*, InstanceHandle_t *handle*)

Reverses the action of com.rti.dds.topic.example.FooDataWriter.register_instance.

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.dds.topic.example.FooDataWriter.register_instance. Otherwise, this operation may fail with RETCODE_BAD_PARAMETER.

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. 538) 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 com.rti.dds.topic.example.FooDataWriter.unregister_instance().

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 1044) operation is for the instance as indicated by the `handle`.

If after a com.rti.dds.topic.example.FooDataWriter.unregister_instance, the application wants to modify (com.rti.dds.topic.example.FooDataWriter.write or com.rti.dds.topic.example.FooDataWriter.dispose) an instance, it has to register it again, or else use the special `handle` value **InstanceHandle_t.HANDLE_-**

NIL (p. 1082).

This operation does not indicate that the instance is deleted (that is the purpose of com.rti.dds.topic.example.FooDataWriter.dispose). The operation com.rti.dds.topic.example.FooDataWriter.unregister_instance just indicates that the **com.rti.dds.publication.DataWriter** (p. 538) no longer has anything to say about the instance. **com.rti.dds.subscription.DataReader** (p. 473) entities that are reading the instance may receive a sample with InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE for the instance, unless there are other **com.rti.dds.publication.DataWriter** (p. 538) objects writing that same instance.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 83)). If the **com.rti.dds.publication.DataWriter** (p. 538) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1044) will relinquish that ownership.

If **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS 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. 1339); if this writer is still unable to unregister after that period, this method will fail with RETCODE_TIMEOUT.

Parameters:

instance_data *<<in>>* *(p. 271)* The instance that should be unregistered. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER . If Foo *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<in>>* *(p. 271)* represents the `instance` to be unregistered. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If Foo 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 RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will report the error RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
FooDataWriter.unregister_instance_w_timestamp (p. 1046)
com.rti.dds.topic.example.FooDataWriter.get_key_value
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.103.2.4 void unregister_instance_w_timestamp (Foo *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.unregister_instance 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. 51) 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.dds.topic.example.FooDataWriter.unregister_instance operation.

This operation may block and may time out (RETCODE_TIMEOUT) under the same circumtances described for the unregister_instance operation.

Parameters:

instance_data *<<in>> (p. 271)* The instance that should be unregistered. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 RETCODE_BAD_PARAMETER. If Foo *has* a key, `instance_data` can be NULL only if `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

handle *<<in>> (p. 271)* represents the `instance` to be unregistered. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` is not used and `instance` is deduced from `instance_data`. If Foo 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 RETCODE_BAD_PARAMETER.

This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<**in**>>* *(p. 271)* 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. 104), RETCODE_TIMEOUT or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
com.rti.dds.topic.example.FooDataWriter.unregister_instance
com.rti.dds.topic.example.FooDataWriter.get_key_value

8.103.2.5 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) QoS policy for details).

As a side effect, this operation asserts liveliness on the **com.rti.dds.publication.DataWriter** (p. 538) itself, the **com.rti.dds.publication.Publisher** (p. 1277) and the **com.rti.dds.domain.DomainParticipant** (p. 629).

Note that the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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. 1047) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 101) `kind` is set to ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 102) to be exceeded.

Specifically, this operation may block in the following situations (note that the list may not be exhaustive), even if its **com.rti.dds.infrastructure.HistoryQosPolicyKind** (p. 1075) is HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS:

- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360) * **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074)), then in the situation where the `max_samples` resource limit is exhausted, RTI Connext is allowed to discard samples of some other instance, as long as at least one sample remains for such an instance. If it is still not possible to make space available to store the modification, the writer is allowed to block.

- If (**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_instances** (p. 1360)), then the **DataWriter** (p. 538) may block regardless of the **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074).

- If (**com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1389) < **com.rti.dds.infrastructure.ResourceLimitsQosPolicy**
samples (p. 1359)), then it is possible for the send_window_size limit to be reached before RTI Connext is allowed to discard samples, in which case the **com.rti.dds.publication.DataWriter** (p. 538) will block.

This operation may also block when using ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS and PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS. In this case, the **com.rti.dds.publication.DataWriter** (p. 538) will queue samples until they are sent by the asynchronous publishing thread. The number of samples that can be stored is determined by the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071). If the asynchronous thread does not send samples fast enough (e.g., when using a slow **com.rti.dds.publication.FlowController** (p. 942)), the queue may fill up. In that case, subsequent write calls will block.

If this operation *does* block for any of the above reasons, the **RELIABILITY** (p. 101) `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 **com.rti.dds.publication.DataWriter** (p. 538) is able to store the modification without exceeding the limits, the operation will time out (RETCODE_TIMEOUT).

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

This operation will fail with RETCODE_PRECONDITION_NOT_MET 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).

Parameters:

instance_data *<<**in**>>* *(p. 271)* The data to write.

This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

Parameters:

handle *<<**in**>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_PRECONDITION_NOT_MET, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.subscription.DataReader (p. 473)
com.rti.dds.topic.example.FooDataWriter.write_w_timestamp
DESTINATION_ORDER (p. 51)

8.103.2.6 void write_w_timestamp (Foo *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as com.rti.dds.topic.example.FooDataWriter.write 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). (Refer to **com.rti.dds.subscription.SampleInfo** (p. 1404) and **DESTINATION_ORDER** (p. 51) 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.dds.topic.example.FooDataWriter.write operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help free up some resources.

This operation may fail with RETCODE_BAD_PARAMETER under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>> (p. 271)* The data to write. This method will fail with RETCODE_BAD_PARAMETER if `instance_data` is NULL.

handle *<<in>> (p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082), `handle` must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

source_timestamp *<<in>> (p. 271)* When using 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. 609), 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. 609), the function will return RETCODE_BAD_PARAMETER.

Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES, or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.write
com.rti.dds.subscription.DataReader (p. 473)
DESTINATION_ORDER (p. 51)

8.103.2.7 void dispose (Foo *instance_data*, InstanceHandle_t *instance_handle*)

Requests the middleware to delete the data.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

The actual deletion is postponed until there is no more use for that data in the whole system.

Applications are made aware of the deletion by means of operations on the **com.rti.dds.subscription.DataReader** (p. 473) objects that already knew that instance. **com.rti.dds.subscription.DataReader** (p. 473) objects that didn't know the instance will never see it.

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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.unregister_instance operation.

The special value **InstanceHandle_t.HANDLE_NIL** (p. 1082) 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 `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with RETCODE_BAD_PARAMETER.

RTI Connext will not detect the error when the `handle` is any value other than **InstanceHandle_t.HANDLE_NIL** (p. 1082), 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 **dispose()** (p. 1051) operation is for the instance as indicated by the `handle`.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write().

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>> (p. 271)* The data to dispose. If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If Foo *has* a key, `instance_data` can be NULL only if `instance_handle` is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>> (p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If Foo has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type Foo. Otherwise, this method fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if

instance_data is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.103.2.8 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. 473) objects by means of the source_timestamp attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the com.rti.dds.topic.example.FooDataWriter.dispose operation.

This operation may block and time out (RETCODE_TIMEOUT) under the same circumtances described for com.rti.dds.topic.example.FooDataWriter.write.

If there are no instance resources left, this operation may fail with RETCODE_OUT_OF_RESOURCES. Calling com.rti.dds.topic.example.FooDataWriter.unregister_instance may help freeing up some resources.

Parameters:

instance_data *<<in>> (p. 271)* The data to dispose. If Foo *has* a key and instance_handle is **InstanceHandle_t.HANDLE_NIL** (p. 1082), only the fields that represent the key are examined by the function. Otherwise, instance_data is not used. If Foo *has* a key, instance_data can be NULL only if instance_handle is not **InstanceHandle_t.HANDLE_NIL** (p. 1082). Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

instance_handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). If Foo *has* a key and `instance_handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), `instance_handle` is not used and `instance` is deduced from `instance_data`. If Foo has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type Foo. Otherwise, this method may fail with RETCODE_BAD_PARAMETER This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL. If Foo *has* a key, `instance_handle` cannot be **InstanceHandle_t.HANDLE_NIL** (p. 1082) if `instance_data` is NULL. Otherwise, this method will fail with RETCODE_BAD_PARAMETER.

source_timestamp *<<in>>* *(p. 271)* 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. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_TIMEOUT, RETCODE_OUT_OF_RESOURCES or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.103.2.9 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 Foo has no key, this method has no effect and exit with no error.

For keyed data types, this operation may fail with RETCODE_BAD_PARAMETER if the `handle` does not correspond to an existing data-object known to the **com.rti.dds.publication.DataWriter** (p. 538).

Parameters:

key_holder *<<inout>>* *(p. 271)* a user data type specific key holder, whose `key` fields are filled by this operation. If Foo has no key, this method has no effect. This method will fail with RETCODE_BAD_PARAMETER if `key_holder` is NULL.

handle *<<in>>* *(p. 271)* the `instance` whose key is to be retrieved. If Foo *has* a key, `handle` must represent a registered instance of type Foo. Otherwise, this method will fail with RETCODE_BAD_PARAMETER. If Foo *has* a key and `handle` is **InstanceHandle_t.HANDLE_NIL** (p. 1082), this method will fail with RETCODE_BAD_PARAMETER. This method will fail with RETCODE_BAD_PARAMETER if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.103.2.10 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. 271)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If Foo has no key, this method has no effect and returns **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.104 FooSeq Class Reference

<<*interface*>> *(p. 271)* <<*generic*>> *(p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo (p. 955)`.

Inheritance diagram for FooSeq::

Public Member Functions

- **FooSeq** ()
- **FooSeq** (int initialMaximum)
- **FooSeq** (Collection elements)

8.104.1 Detailed Description

<<*interface*>> *(p. 271)* <<*generic*>> *(p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo (p. 955)`.

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo (p. 955)>`.

For any user-data type `Foo (p. 955)` that an application defines for the purpose of data-distribution with RTI Connext, a `FooSeq (p. 1056)` is generated. We refer to an IDL `sequence<Foo (p. 955)> as FooSeq (p. 1056)`.

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. 383)` 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 **Sequence.setMaximum** (p. 1433).

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.dds.topic.example.FooDataWriter, com.rti.dds.topic.example.FooDataReader,

com.rti.dds.topic.example.FooTypeSupport (p. 1060), **rtiddsgen** (p. 290)

8.104.2 Constructor & Destructor Documentation

8.104.2.1 FooSeq ()

Construct a new empty sequence with an initial maximum of 0.

8.104.2.2 FooSeq (int *initialMaximum*)

Construct a new empty sequence with the given initial maximum.

8.104.2.3 FooSeq (Collection *elements*)

Construct a new sequence containing the same elements as the given collection and having an initial maximum equal to its size.

8.105 FooSeq Class Reference

<<***interface***>> *(p. 271)* <<***generic***>> *(p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 956).

Inheritance diagram for FooSeq::

Public Member Functions

- Object **copy_from** (Object src)

Package Attributes

- **Sequence _loanedInfoSequence** = null

8.105.1 Detailed Description

<<***interface***>> *(p. 271)* <<***generic***>> *(p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 956).

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo` (p. 956)`>`.

For any user-data type `Foo` (p. 956) that an application defines for the purpose of data-distribution with RTI Connext, a `FooSeq` (p. 1058) is generated. We refer to an IDL `sequence<Foo` (p. 956)`> as FooSeq` (p. 1058).

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. 383) 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 **Sequence.setMaximum** (p. 1433).

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.dds.topic.example.FooDataWriter, com.rti.dds.topic.example.FooDataReader, **com.rti.dds.topic.example.FooTypeSupport** (p. 1060), **rtiddsgen** (p. 290)

8.105.2 Member Function Documentation

8.105.2.1 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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.105.3 Member Data Documentation

8.105.3.1 Sequence _loanedInfoSequence = null [package]

When a memory loan has been taken out in the lower layers of NDDS, store a pointer to the native sequence here. That way, when we call finish(), we can give the memory back.

8.106 FooTypeSupport Class Reference

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* User data type specific interface.

Static Public Member Functions

- static void **register_type** (**DomainParticipant** participant, String type_name)

  Allows an application to communicate to RTI Connext the existence of a data type.

- static String **type_name** ()

  Get the default name for this type.

8.106.1 Detailed Description

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* 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:

rtiddsgen (p. 290)

8.106.2 Member Function Documentation

8.106.2.1 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 **TypeSupport** (p. 1651) can be registered multiple times with a **com.rti.dds.domain.DomainParticipant** (p. 629) using the same

or different values for the `type_name`. If `register_type` is called multiple times on the same **TypeSupport** (p. 1651) with the same **com.rti.dds.domain.DomainParticipant** (p. 629) and `type_name`, the second (and subsequent) registrations are ignored by the operation fails with RETCODE_OK.

Precondition:

Cannot use the same `type_name` to register two different **TypeSupport** (p. 1651) with the same **com.rti.dds.domain.DomainParticipant** (p. 629), or else the operation will fail and RETCODE_PRECONDITION_NOT_MET will be returned.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to register the data type `Foo` (p. 955) with. Cannot be NULL.

type_name *<<in>> (p. 271)* the type name under with the data type `Foo` (p. 955) is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1545). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670).) The name may not be NULL or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_OUT_OF_RESOURCES.

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. 670)

8.106.2.2 static String type_name () [static]

Get the default name for this type.

Can be used for calling **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) or creating **com.rti.dds.topic.Topic** (p. 1545)

Returns:

default name for this type

See also:

com.rti.dds.topic.example.FooTypeSupport.register_type (p. 1060)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.107 FooTypeSupport Class Reference

Inherits TypeSupportImpl.

Public Member Functions

- ^ Object **copy_data** (Object destination, Object source)

8.107.1 Detailed Description

A collection of useful methods for dealing with objects of type **Foo** (p. 956).

8.107.2 Member Function Documentation

8.107.2.1 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. 956) type.

8.108 GroupDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Inheritance diagram for GroupDataQosPolicy::

Public Attributes

- final **ByteSeq value**

  a sequence of octets

8.108.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Entity:

com.rti.dds.publication.Publisher (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **YES** (p. 98)

See also:

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.108.2 Usage

The additional information is attached to a **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478). This extra data is not used by RTI Connext itself. When a remote application discovers the **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478), 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. 1559) and **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680), 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. 501), **com.rti.dds.publication.DataWriterListener** (p. 566) and operations such as **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 688) and **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 689), 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. 1233), 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. 753) and **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPo group_data_max_length** (p. 754).

8.108.3 Member Data Documentation

8.108.3.1 final ByteSeq value

a sequence of octets

[**default**] Empty (zero-sized)

[**range**] Octet sequence of length [0,max_length]

8.109 GuardCondition Class Reference

<<*interface*>> *(p. 271)* A specific **com.rti.dds.infrastructure.Condition** (p. 451) whose `trigger_value` is completely under the control of the application.

Inherits AbstractNativeObject, 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.

8.109.1 Detailed Description

<<*interface*>> *(p. 271)* A specific **com.rti.dds.infrastructure.Condition** (p. 451) whose `trigger_value` is completely under the control of the application.

The **com.rti.dds.infrastructure.GuardCondition** (p. 1066) provides a way for an application to manually wake up a **com.rti.dds.infrastructure.WaitSet** (p. 1695). This is accomplished by attaching the **com.rti.dds.infrastructure.GuardCondition** (p. 1066) to the **com.rti.dds.infrastructure.WaitSet** (p. 1695) and then setting the `trigger_value` by means of the **com.rti.dds.infrastructure.GuardCondition.set_trigger_value** (p. 1067) operation.

Important: The **com.rti.dds.infrastructure.GuardCondition** (p. 1066) allocates native resources. When **com.rti.dds.infrastructure.GuardCondition** (p. 1066) is no longer being used, user should call **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1067) explicitly to properly cleanup all native resources.

See also:

com.rti.dds.infrastructure.WaitSet (p. 1695)

8.109.2 Constructor & Destructor Documentation

8.109.2.1 GuardCondition ()

No argument constructor.

Construct a new guard condition with trigger value false.

Exceptions:

RETCODE_OUT_OF_RESOURCES *(p. 1370)* if a new **com.rti.dds.infrastructure.GuardCondition** (p. 1066) could not be allocated.

Important: The **com.rti.dds.infrastructure.GuardCondition** (p. 1066) allocates native resources. When **com.rti.dds.infrastructure.GuardCondition** (p. 1066) is no longer being used, user should call **com.rti.dds.infrastructure.GuardCondition.delete** (p. 1067) explicitly to properly cleanup all native resources.

8.109.3 Member Function Documentation

8.109.3.1 void set_trigger_value (boolean *value*)

Set the guard condition trigger value.

Parameters:

value *<<in>>* *(p. 271)* the new trigger value.

8.109.3.2 boolean get_trigger_value ()

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implements **Condition** (p. 451).

8.109.3.3 void delete ()

Destructor.

Releases the resources associated with this object.

Calling this method multiple times on the same object is safe; subsequent deletions will have no effect.

8.110 GUID_t Class Reference

Type for *GUID* (Global Unique Identifier) representation.

Inherits Struct.

Public Member Functions

- **GUID_t** (**GUID_t** guid)
  Copy constructor.
- **GUID_t** (byte[] **value**)
  Constructor.

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.

8.110.1 Detailed Description

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

8.110.2 Constructor & Destructor Documentation

8.110.2.1 GUID_t (GUID_t *guid*)

Copy constructor.

Parameters:

guid The GUID instance to copy. It must not be null.

8.110.2.2 GUID_t (byte[] *value*)

Constructor.

Parameters:

value GUID value as a 16 byte array. It must not be null.

8.110.3 Member Data Documentation

8.110.3.1 final GUID_t GUID_UNKNOWN [static]

Unknown GUID.

8.110.3.2 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.virtu**
guid (p. 572) or **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.virtual_-**
guid (p. 505), RTI Connext will assign the virtual GUID automatically based on the RTPS or physical GUID.

8.110.3.3 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.

8.111 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 to be kept, when the `kind` *is* ***HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS*** *(*p. 1075*).*

- **RefilterQosPolicyKind refilter**

  <<**eXtension**>> (p. 270) *Specifies how a writer should handle previously written samples to a new reader.*

8.111.1 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. 538) or **com.rti.dds.subscription.DataReader** (p. 473). 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. 538) on behalf of existing **com.rti.dds.subscription.DataReader** (p. 473) entities. The behavior with regards to a **com.rti.dds.subscription.DataReader** (p. 473) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 65) 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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

See also:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1336)
com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)

8.111.2 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. 473) entities.

When a **com.rti.dds.publication.DataWriter** (p. 538) sends data, or a **com.rti.dds.subscription.DataReader** (p. 473) 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. 1336) by controlling whether RTI Connext guarantees that *all* of the sent data is received (**HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076)) or if only the last N data values sent are guaranteed to be received (**HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076))–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.DurabilityQosPolicy** (p. 765) to receive previously published data is also controlled 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. 1356).

If `kind` is **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075) (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. 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.

The default (and most common setting) for `depth` is 1, indicating that only the

most recent value should be delivered.

If `kind` is **HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076), 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. 102). If the limit is reached, then the behavior of RTI Connext will depend on the **RELIABILITY** (p. 101). If the Reliability `kind` is **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340), then the old values will be discarded. If Reliability `kind` is RELIABLE, then RTI Connext will block the **com.rti.dds.publication.DataWriter** (p. 538) until it can deliver the necessary old values to all subscribers.

If `refilter` is **com.rti.dds.infrastructure.RefilterQosPolicyKind.NONE_REFILTER_QOS** (p. 1334), then samples written before a DataReader is matched to a DataWriter are not refiltered by the DataWriter.

If `refilter` is **com.rti.dds.infrastructure.RefilterQosPolicyKind.ALL_REFILTER_QOS** (p. 1335), then all samples written before a DataReader is matched to a DataWriter are refiltered by the DataWriter when the DataReader is matched.

If `refilter` is **com.rti.dds.infrastructure.RefilterQosPolicyKind.ON_DEMAND_REFILTER_QOS** (p. 1335), then a DataWriter will only refilter samples that a DataReader requests.

8.111.3 Consistency

This QoS policy's `depth` must be consistent with the **RESOURCE_LIMITS** (p. 102) `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. 1356)

8.111.4 Member Data Documentation

8.111.4.1 HistoryQosPolicyKind kind

Specifies the kind of history to be kept.

[**default**] **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075)

8.111.4.2 int depth

Specifies the number of samples to be kept, when the `kind` is **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075).

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 102) policy. That is:

depth <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360)

When the `kind` is **HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076), the depth has no effect. Its implied value is `infinity` (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 102) policy).

[default] 1

[range] [1,100 million], <= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360)

8.111.4.3 RefilterQosPolicyKind refilter

<<***eXtension***>> *(p. 270)* Specifies how a writer should handle previously written samples to a new reader.

[default] **com.rti.dds.infrastructure.RefilterQosPolicyKind.NONE_REFILTER_QOS** (p. 1334)

8.112 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.*

8.112.1 Detailed Description

Kinds of history.

QoS:

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)

8.112.2 Member Data Documentation

8.112.2.1 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. 538).

On the subscribing side, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 's **take()** operation.

8.112.2.2 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. 538) 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. 473). These samples are kept until the application takes them from RTI Connext via the **take()** operation.

8.113 InconsistentTopicStatus Class Reference

StatusKind.INCONSISTENT_TOPIC_STATUS.

Inherits Status.

Public Attributes

- int **total_count**

  *Total cumulative count of the Topics discovered whose name matches the **com.rti.dds.topic.Topic** (p. 1545) to which this status is attached and whose type is inconsistent with that of that **com.rti.dds.topic.Topic** (p. 1545).*

- int **total_count_change**

  The incremental number of inconsistent topics discovered since the last time this status was read.

8.113.1 Detailed Description

StatusKind.INCONSISTENT_TOPIC_STATUS.

Entity:

com.rti.dds.topic.Topic (p. 1545)

Listener:

com.rti.dds.topic.TopicListener (p. 1564)

A remote **com.rti.dds.topic.Topic** (p. 1545) will be inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1545) if the type name of the two topics are different.

8.113.2 Member Data Documentation

8.113.2.1 int total_count

Total cumulative count of the Topics discovered whose name matches the **com.rti.dds.topic.Topic** (p. 1545) to which this status is attached and whose type is inconsistent with that of that **com.rti.dds.topic.Topic** (p. 1545).

8.113.2.2 int total_count_change

The incremental number of inconsistent topics discovered since the last time this status was read.

8.114 InetAddressSeq Class Reference

Declares IDL `sequence`< 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.

8.114.1 Detailed Description

Declares IDL `sequence`< java.net.InetAddress >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

java.net.InetAddress

8.114.2 Constructor & Destructor Documentation

8.114.2.1 InetAddressSeq ()

Construct a new empty sequence.

8.114.2.2 InetAddressSeq (int *initial_maximum*)

Construct a new empty sequence with the given maximum.

8.114.2.3 InetAddressSeq (Collection *addresses*)

Construct a new sequence containing all of the given addresses.

8.115 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 ***InstanceHandle_t.HANDLE_NIL*** *(*p. 1082*).*

- 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.

Static Public Attributes

- static final **InstanceHandle_t HANDLE_NIL**

  The NIL instance handle.

8.115.1 Detailed Description

Type definition for an instance handle.

Handle to identiy different instances of the same **com.rti.dds.topic.Topic** (p. 1545) of a certain type.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance
com.rti.dds.subscription.SampleInfo.instance_handle (p. 1410)

8.115.2 Constructor & Destructor Documentation

8.115.2.1 InstanceHandle_t ()

Construct a new instance handle equal to the nil handle.

See also:

HANDLE_NIL (p. 1082)

8.115.2.2 InstanceHandle_t (InstanceHandle_t *src*)

Construct a new instance handle equal to the given handle.

Exceptions:

NullPointerException if src is null

8.115.3 Member Function Documentation

8.115.3.1 boolean is_nil ()

Compare this handle to **InstanceHandle_t.HANDLE_NIL** (p. 1082).

Returns:

true if the given instance handle is equal to **InstanceHandle_t.HANDLE_NIL** (p. 1082) or false otherwise.

See also:

com.rti.dds.infrastructure.InstanceHandle_t.equals (p. 1082)

8.115.3.2 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. 271)* 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. 466).

8.115.3.3 boolean equals (Object *other*)

Compares this instance handle with another handle for equality.

Parameters:

other *<<in>>* *(p. 271)* The other handle to be compared with this handle. Cannot be null.

Returns:

true if the two handles have equal values, or false otherwise.

See also:

com.rti.dds.infrastructure.InstanceHandle_t.is_nil (p. 1081)

8.115.4 Member Data Documentation

8.115.4.1 final InstanceHandle_t HANDLE_NIL [static]

The NIL instance handle.

Special **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) value

See also:

com.rti.dds.infrastructure.InstanceHandle_t.is_nil (p. 1081)

8.116 InstanceHandleSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) > .

Inherits ArraySequence.

Public Member Functions

- **InstanceHandleSeq** ()

  Construct a new empty sequence for ***com.rti.dds.infrastructure.InstanceHandle_t*** *(p. 1080) elements.*

- **InstanceHandleSeq** (int initial_maximum)

  Construct a new empty sequence for ***com.rti.dds.infrastructure.InstanceHandle_t*** *(p. 1080) elements.*

- **InstanceHandleSeq** (Collection elements)

  Construct a new sequence containing the given ***com.rti.dds.infrastructure.InstanceHandle_t*** *(p. 1080) elements.*

- void **fill** (int size)

  Fill this sequence with the given number of instance handles.

8.116.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) > .

When reading into this sequence (as with **com.rti.dds.publication.DataWriter.get_matched_subscriptions** (p. 550) or **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 486)), 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.dds.topic.example.FooDataWriter.register_instance). 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. 1084).

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.InstanceHandle_t (p. 1080)
com.rti.dds.util.Sequence (p. 1432)

8.116.2 Constructor & Destructor Documentation

8.116.2.1 InstanceHandleSeq ()

Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) elements.

The maximum of the sequence will be set to a default value.

8.116.2.2 InstanceHandleSeq (int *initial_maximum*)

Construct a new empty sequence for **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) elements.

The maximum of the sequence will be set to the given value.

Parameters:

initial_maximum *<<in>>* *(p. 271)* Maximum length of sequence.

8.116.2.3 InstanceHandleSeq (Collection *elements*)

Construct a new sequence containing the given **com.rti.dds.infrastructure.InstanceHandle_t** (p. 1080) elements.

The maximum of the sequence will be set to the size of the given collection.

Parameters:

elements *<<in>>* *(p. 271)* Elements to construct a sequence with.

8.116.3 Member Function Documentation

8.116.3.1 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. *271)* Size of sequence to ensure.

Exceptions:

RETCODE_BAD_PARAMETER *(*p. ***1363****)* If size < 0

8.117 InstanceStateKind Class Reference

Indicates is the samples are from a live **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) objects are currently alive (according to the **LIVELINESS** (p. 78)) 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.

8.117.1 Detailed Description

Indicates is the samples are from a live **com.rti.dds.publication.DataWriter** (p. 538) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- InstanceStateKind.ALIVE_INTANCE_STATE indicates that (a) samples have been received for the instance, (b) there are live **com.rti.dds.publication.DataWriter** (p. 538) entities writing the instance, and (c) the instance has not been explicitly disposed (or else more samples have been received after it was disposed).

- **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) indicates the instance was explicitly disposed by a **com.rti.dds.publication.DataWriter** (p. 538) by means of the dispose operation.

- **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088) indicates the instance has been declared as not-alive by the **com.rti.dds.subscription.DataReader** (p. 473) because it detected that there are no live **com.rti.dds.publication.DataWriter** (p. 538) 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. 83) is set to OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS, then the instance state becomes **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) only if the **com.rti.dds.publication.DataWriter** (p. 538) that "owns" the instance explicitly disposes it. The instance state becomes InstanceStateKind.ALIVE_INTANCE_STATE again only if the **com.rti.dds.publication.DataWriter** (p. 538) that owns the instance writes it.

- If **OWNERSHIP** (p. 83) is set to OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS, then the instance state becomes **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) if any **com.rti.dds.publication.DataWriter** (p. 538) explicitly disposes the instance. The instance state becomes InstanceStateKind.ALIVE_INTANCE_STATE as soon as any **com.rti.dds.publication.DataWriter** (p. 538) writes the instance again.

The instance state available in the **com.rti.dds.subscription.SampleInfo** (p. 1404) 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.

8.117.2 Member Data Documentation

8.117.2.1 final int ALIVE_INSTANCE_STATE = 0x0001 << 0 [static]

Instance is currently in existence.

8.117.2.2 final int NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0001 << 1 [static]

Not alive disposed instance. The instance has been disposed by a DataWriter.

8.117.2.3 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. 538) objects are currently alive (according to the **LIVELINESS** (p. 78)) are writing the instance.

8.118 IntSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 1091) that returns a java.lang.Integer.*

- Object **set** (int index, Object element)

  *A wrapper for **setInt()** (p. 1092).*

- void **add** (int index, Object element)

  *A wrapper for **addInt(int, int)** (p. 1091).*

8.118.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < int >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

int
com.rti.dds.util.Sequence (p. 1432)

8.118.2 Constructor & Destructor Documentation

8.118.2.1 IntSeq ()

Constructs an empty sequence of integers with an initial maximum of zero.

8.118.2.2 IntSeq (int *initialMaximum*)

Constructs an empty sequence of integers with the given initial maximum.

8.118.2.3 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

8.118.3 Member Function Documentation

8.118.3.1 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.

8.118.3.2 boolean addAllInt (int[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.118.3.3 void addInt (int *element*)

Append the element to the end of the sequence.

8.118.3.4 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.

8.118.3.5 int getInt (int *index*)

Returns the integer at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.118.3.6 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.

8.118.3.7 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.

8.118.3.8 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.

8.118.3.9 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. 1094), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.118.3.10 Object get (int *index*) [virtual]

A wrapper for **getInt(int)** (p. 1091) that returns a java.lang.Integer.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.118.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setInt()** (p. 1092).

Exceptions:

ClassCastException if the element is not of type Integer.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.118.3.12 void add (int *index*, Object *element*) [virtual]

A wrapper for **addInt(int, int)** (p. 1091).

Exceptions:

ClassCastException if the element is not of type Integer.

See also:

java.util.List.add(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 378).

8.119 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 size)
  Constructor that specifies the allocated sizes.
- Object **copy_from** (Object src)
  Copy value of a data type from source.

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.Bytes *(*p. 417*) array value.*

8.119.1 Detailed Description

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

8.119.2 Constructor & Destructor Documentation

8.119.2.1 KeyedBytes ()

Default Constructor.

The default constructor initializes the newly created object with empty key, null value, zero length, and zero offset.

8.119.2.2 KeyedBytes (KeyedBytes *src*)

Copy constructor.

Parameters:

src *<<in>>* *(*p. 271*)* Object to copy from.

Exceptions:

NullPointerException if src is null.

8.119.2.3 KeyedBytes (int *size*)

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:

size *<<in>>* *(*p. 271*)* Size of the allocated bytes array.

Exceptions:

IllegalArgumentException if size is negative

8.119.3 Member Function Documentation

8.119.3.1 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. 271*)* 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. 466).

8.119.4 Member Data Documentation

8.119.4.1 String key

Instance key associated with the specified value.

8.119.4.2 int length

Number of bytes to serialize.

8.119.4.3 int offset

Offset from which to start serializing bytes .

The first position of the bytes array has offset 0.

8.119.4.4 byte [] value

com.rti.dds.type.builtin.Bytes (p. 417) array value.

8.120 KeyedBytesDataReader Class Reference

<<*interface*>> (p. 271) Instantiates `DataReader` < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

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. 473).

- 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. 473).

- void **read_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Accesses via ***com.rti.dds.type.builtin.KeyedBytesDataReader.read*** (p. 1101) *the samples that match the criteria specified in the* ***com.rti.dds.subscription.ReadCondition*** (p. 1326).

- void **take_w_condition** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to ***com.rti.dds.type.builtin.KeyedBytesDataReader.read_w_condition*** (p. 1101) *except it accesses samples via the* ***com.rti.dds.type.builtin.KeyedBytesDataReader.take*** (p. 1101) *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. 473).

- 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. 473).

- 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. 473).

- 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. 473).

- 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.KeyedBytesDataReader.read_instance*** (p. 1102) *the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition*** (p. 1326).

- 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.KeyedBytesDataReader.take_instance*** (p. 1102) *the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition*** (p. 1326).

- 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. 473).

- 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. 473).

- 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.KeyedBytesDataReader.read_next_instance*** (p. 1103) *the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition*** (p. 1326).

- 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.KeyedBytesDataReader.take_next_instance*** *(p. 1104) the samples that match the criteria specified in the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

- void **return_loan** (**KeyedBytesSeq** received_data, **SampleInfoSeq** info_seq)

  Indicates to the ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 473).*

- 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. 270) *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. 270) *Retrieve the instance* `handle` *that corresponds to an instance* `key`*.*

8.120.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataReader` < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

See also:

com.rti.dds.topic.example.FooDataReader
com.rti.dds.subscription.DataReader (p. 473)

8.120.2 Member Function Documentation

8.120.2.1 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read

8.120.2.2 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take

8.120.2.3 void read_w_condition (KeyedBytesSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via **com.rti.dds.type.builtin.KeyedBytesDataReader.read** (p. 1101) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition

8.120.2.4 void take_w_condition (KeyedBytesSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to **com.rti.dds.type.builtin.KeyedBytesDataReader.read_w_condition** (p. 1101) except it accesses samples via the **com.rti.dds.type.builtin.KeyedBytesDataReader.take** (p. 1101) operation.

See also:

com.rti.dds.topic.example.FooDataReader.take_w_condition

8.120.2.5 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_sample

8.120.2.6 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_sample

8.120.2.7 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_instance

8.120.2.8 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_instance

8.120.2.9 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.KeyedBytesDataReader.read_instance** (p. 1102) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_instance_w_condition

8.120.2.10 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.KeyedBytesDataReader.take_instance** (p. 1102) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.take_instance_w_condition

8.120.2.11 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance

8.120.2.12 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance

8.120.2.13 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.KeyedBytesDataReader.read_next_instance** (p. 1103) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition

8.120.2.14 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.KeyedBytesDataReader.take_next_instance** (p. 1104) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition

8.120.2.15 void return_loan (KeyedBytesSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.return_loan

8.120.2.16 void get_key_value (KeyedBytes *key_holder*, InstanceHandle_t *handle*)

Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.120.2.17 String get_key_value (InstanceHandle_t *handle*)

<<***eXtension***>> *(p. 270)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.120.2.18 InstanceHandle_t lookup_instance (KeyedBytes *key_holder*)

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also:

com.rti.dds.topic.example.FooDataReader.lookup_instance

8.120.2.19 InstanceHandle_t lookup_instance (String *key*)

<<***eXtension***>> *(p. 270)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

com.rti.dds.topic.example.FooDataReader.lookup_instance

8.121 KeyedBytesDataWriter Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

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. 270) *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.KeyedBytesDataWriter.register_instance** (p. 1109) 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. 270) *Performs the same functions as **KeyedBytesDataWriter.register_instance** (p. 1109) 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.KeyedBytesDataWriter.register_instance** (p. 1109).*

- void **unregister_instance** (String key, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Reverses the action of **KeyedBytesDataWriter.register_instance** (p. 1109).*

- 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.KeyedBytesDataWriter.unregister_instance** (p. 1110) 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. 270) *Performs the same function as* ***KeyedBytesDataWriter.unregister_instance*** *(p. 1110) 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.KeyedBytes*** *(p. 1095) data instance.*

- void **write** (String key, byte[] octets, int offset, int length, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Modifies the value of a* ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) data instance.*

- void **write** (String key, **ByteSeq** octets, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Modifies the value of a* ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) 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.KeyedBytesDataWriter.write*** *(p. 1111) 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 ***KeyedBytesDataWriter.write*** *(p. 1111) 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 ***KeyedBytesDataWriter.write*** *(p. 1111) 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. 270) *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.KeyedBytesDataWriter.dispose*** *(p. 1113) except that the application provides the value for the* `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

- void **dispose_w_timestamp** (String key, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

  <<**eXtension**>> (p. 270) *Performs the same functions as* ***KeyedBytesDataWriter.dispose*** *(p. 1113) except that the application provides the value for the* `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

- 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. 270) *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. 270) *Retrieve the instance* `handle` *that corresponds to an instance* `key`*.*

8.121.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

See also:

com.rti.dds.topic.example.FooDataWriter
com.rti.dds.publication.DataWriter (p. 538)

8.121.2 Member Function Documentation

8.121.2.1 InstanceHandle_t register_instance (KeyedBytes *instance_data*)

Informs RTI Connext that the application will be modifying a particular instance.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance

8.121.2.2 InstanceHandle_t register_instance (String *key*)

<<***eXtension***>> *(p. 270)* Informs RTI Connext that the application will be modifying a particular instance.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance

8.121.2.3 InstanceHandle_t register_instance_w_timestamp (KeyedBytes *instance_data*, Time_t *source_timestamp*)

Performs the same functions as **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109) except that the application provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp

8.121.2.4 InstanceHandle_t register_instance_w_timestamp (String *key*, Time_t *source_timestamp*)

<<***eXtension***>> *(p. 270)* Performs the same functions as **KeyedBytesDataWriter.register_instance** (p. 1109) except that the application provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp

8.121.2.5 void unregister_instance (KeyedBytes *instance_data*, InstanceHandle_t *handle*)

Reverses the action of **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance

8.121.2.6 void unregister_instance (String *key*, InstanceHandle_t *handle*)

<<eXtension>> *(p. 270)* Reverses the action of **KeyedBytesDataWriter.register_instance** (p. 1109).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance

8.121.2.7 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.KeyedBytesDataWriter.unregister_instance** (p. 1110) except that it also provides the value for the `source_timestamp`.

See also:

FooDataWriter.unregister_instance_w_timestamp

8.121.2.8 void unregister_instance_w_timestamp (String *key*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

<<eXtension>> *(p. 270)* Performs the same function as **KeyedBytesDataWriter.unregister_instance** (p. 1110) except that it also provides the value for the `source_timestamp`.

See also:

FooDataWriter.unregister_instance_w_timestamp

8.121.2.9 void write (KeyedBytes *instance_data*, InstanceHandle_t *handle*)

Modifies the value of a **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) data instance.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.121.2.10 void write (String *key*, byte[] *octets*, int *offset*, int *length*, InstanceHandle_t *handle*)

<<eXtension>> *(p. 270)* Modifies the value of a **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) data instance.

Parameters:

key *<<in>>* *(p. 271)* Instance key.

octets *<<in>>* *(p. 271)* Array of bytes to be published.

offset *<<in>>* *(p. 271)* Offset from which to start publishing.

length *<<in>>* *(p. 271)* Number of bytes to be published.

handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109), or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). See com.rti.dds.topic.example.FooDataWriter.write.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.121.2.11 void write (String *key*, ByteSeq *octets*, InstanceHandle_t *handle*)

<<eXtension>> *(p. 270)* Modifies the value of a **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) data instance.

Parameters:

key *<<in>>* *(p. 271)* Instance key.

octets *<<in>>* *(p. 271)* Sequence of bytes to be published.

handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109), or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). See com.rti.dds.topic.example.FooDataWriter.write.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.121.2.12 void write_w_timestamp (KeyedBytes *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as **com.rti.dds.type.builtin.KeyedBytesDataWriter.write** (p. 1111) except that it also provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.121.2.13 void write_w_timestamp (String *key*, byte[] *octets*, int *offset*, int *length*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as **KeyedBytesDataWriter.write** (p. 1111) except that it also provides the value for the `source_timestamp`.

Parameters:

key *<<in>>* *(p. 271)* Instance key.

octets *<<in>>* *(p. 271)* Array of bytes to be published.

offset *<<in>>* *(p. 271)* Offset from which to start publishing.

length *<<in>>* *(p. 271)* Number of bytes to be published.

handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109), or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). See com.rti.dds.topic.example.FooDataWriter.write.

source_timestamp *<<in>>* *(p. 271)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See com.rti.dds.topic.example.FooDataWriter.write_w_timestamp. Cannot be NULL.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.121.2.14 void write_w_timestamp (String *key*, ByteSeq *octets*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as **KeyedBytesDataWriter.write** (p. 1111) except that it also provides the value for the `source_timestamp`.

Parameters:

key *<<in>>* *(p. 271)* Instance key.

octets *<<in>>* *(p. 271)* Sequence of bytes to be published.

handle *<<in>>* *(p. 271)* Either the handle returned by a previous call to **com.rti.dds.type.builtin.KeyedBytesDataWriter.register_instance** (p. 1109), or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082). See com.rti.dds.topic.example.FooDataWriter.write.

source_timestamp *<<in>>* *(p. 271)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See com.rti.dds.topic.example.FooDataWriter.write_w_timestamp. Cannot be NULL.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.121.2.15 void dispose (KeyedBytes *instance_data*, InstanceHandle_t *instance_handle*)

Requests the middleware to delete the data.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.121.2.16 void dispose (String *key*, InstanceHandle_t *instance_handle*)

<<eXtension>> *(p. 270)* Requests the middleware to delete the data.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.121.2.17 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.KeyedBytesDataWriter.dispose** (p. 1113) except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp

8.121.2.18 void dispose_w_timestamp (String *key*, InstanceHandle_t *instance_handle*, Time_t *source_timestamp*)

*<<**eXtension**>> (p. 270)* Performs the same functions as **KeyedBytesDataWriter.dispose** (p. 1113) except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp

8.121.2.19 void get_key_value (KeyedBytes *key_holder*, InstanceHandle_t *handle*)

Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.121.2.20 String get_key_value (InstanceHandle_t *handle*)

*<<**eXtension**>> (p. 270)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.121.2.21 InstanceHandle_t lookup_instance (KeyedBytes *key_holder*)

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also:

com.rti.dds.topic.example.FooDataWriter.lookup_instance

8.121.2.22 InstanceHandle_t lookup_instance (String *key*)

<<***eXtension***>> *(p. 270)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

com.rti.dds.topic.example.FooDataWriter.lookup_instance

8.122 KeyedBytesSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

Inheritance diagram for KeyedBytesSeq::

Public Member Functions

- **KeyedBytesSeq** ()

  Constructs an empty sequence of ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) objects with an initial maximum of zero.*

- **KeyedBytesSeq** (int initialMaximum)

  Constructs an empty sequence of ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) objects with the given initial maximum.*

- **KeyedBytesSeq** (Collection elements)

  Constructs a new sequence containing the given ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) objects.*

- Object **copy_from** (Object src)

Package Attributes

- transient **Sequence** **_loanedInfoSequence** = null

8.122.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.type.builtin.KeyedBytes (p. 1095)

8.122.2 Constructor & Destructor Documentation

8.122.2.1 KeyedBytesSeq ()

Constructs an empty sequence of **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) objects with an initial maximum of zero.

8.122.2.2 KeyedBytesSeq (int *initialMaximum*)

Constructs an empty sequence of **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) objects with the given initial maximum.

8.122.2.3 KeyedBytesSeq (Collection *elements*)

Constructs a new sequence containing the given **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) objects.

Parameters:

elements the initial contents of this sequence.

Exceptions:

NullPointerException if the input collection is null

8.122.3 Member Function Documentation

8.122.3.1 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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.122.4 Member Data Documentation

8.122.4.1 transient Sequence _loanedInfoSequence = null [package]

When a memory loan has been taken out in the lower layers of NDDS, store a pointer to the native sequence here. That way, when we call finish(), we can give the memory back.

8.123 KeyedBytesTypeSupport Class Reference

<<*interface*>> *(p. 271)* **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) type support.

Inheritance diagram for KeyedBytesTypeSupport::

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.KeyedBytes*** *(p. 1095) data type.*

- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the ***com.rti.dds.type.builtin.KeyedBytes*** *(p. 1095) 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.KeyedBytes*** *(p. 1095) type.*

8.123.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) type support.

8.123.2 Member Function Documentation

8.123.2.1 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.KeyedBytes** (p. 1095) data type.

By default, The **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by

com.rti.dds.type.builtin.KeyedBytesTypeSupport.get_type_name (p. 1121). 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.KeyedBytesTypeSupport** (p. 1119) with a **com.rti.dds.domain.DomainParticipant** (p. 629) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 629) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to register the data type **com.rti.dds.type.builtin.Bytes** (p. 417) with. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1545). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_OUT_OF_RESOURCES.

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. 670)

8.123.2.2 static void unregister_type (DomainParticipant *participant*, String *type_name*) [static]

Allows an application to unregister the **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1545) 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. 1545) is associated with the type, the operation will fail with RETCODE_ERROR.

Postcondition:

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to unregister the data type **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) from. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) 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. 104), RETCODE_BAD_PARAMETER or RETCODE_ERROR

MT Safety:

SAFE.

See also:

com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type (p. 1119)

8.123.2.3 static String get_type_name () [static]

Get the default name for the **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) type.

Can be used for calling **com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type** (p. 1119) or creating **com.rti.dds.topic.Topic** (p. 1545).

Returns:

default name for the **com.rti.dds.type.builtin.KeyedBytes** (p. 1095) type.

See also:

com.rti.dds.type.builtin.KeyedBytesTypeSupport.register_type (p. 1119)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.124 KeyedString Class Reference

Keyed string built-in type.

Inheritance diagram for KeyedString::

Public Member Functions

- **KeyedString** ()

  Default Constructor.

- **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.

8.124.1 Detailed Description

Keyed string built-in type.

8.124.2 Constructor & Destructor Documentation

8.124.2.1 KeyedString ()

Default Constructor.

The default constructor initializes the newly created object with empty key and value.

8.124.2.2 KeyedString (KeyedString *src*)

Copy constructor.

Parameters:

src <<*in*>> *(*p. *271)* Object to copy from.

Exceptions:

NullPointerException if src is null.

8.124.3 Member Function Documentation

8.124.3.1 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. *271)* 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. 466).

8.124.4 Member Data Documentation

8.124.4.1 String key

Instance key associated with the specified value.

8.124.4.2 String value

String value.

8.125 KeyedStringDataReader Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataReader` < **com.rti.dds.type.builtin.KeyedString** (p. 1123) >.

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. 473).*

- 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. 473).*

- void **read_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Accesses via ***com.rti.dds.type.builtin.KeyedStringDataReader.read*** *(p. 1128) the samples that match the criteria specified in the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

- void **take_w_condition** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq, int max_samples, **ReadCondition** condition)

  Analogous to ***com.rti.dds.type.builtin.KeyedStringDataReader.read_w_condition*** *(p. 1128) except it accesses samples via the* ***com.rti.dds.type.builtin.KeyedStringDataReader.take*** *(p. 1128) 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. 473).*

- 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. 473).*

- 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. 473).*

- 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. 473).*

- 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.KeyedStringDataReader.read_instance** (p. 1129) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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.KeyedStringDataReader.take_instance** (p. 1129) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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. 473).*

- 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. 473).*

- 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.KeyedStringDataReader.read_next_instance** (p. 1130) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).*

- 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.KeyedStringDataReader.take_next_instance*** *(p. 1131) the samples that match the criteria specified in the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

- void **return_loan** (**KeyedStringSeq** received_data, **SampleInfoSeq** info_seq)

  Indicates to the ***com.rti.dds.subscription.DataReader*** *(p. 473) 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. 473).*

- 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. 270) *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. 270) *Retrieve the instance* `handle` *that corresponds to an instance* `key`*.*

8.125.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataReader` < **com.rti.dds.type.builtin.KeyedString** (p. 1123) >.

See also:

com.rti.dds.topic.example.FooDataReader
com.rti.dds.subscription.DataReader (p. 473)

8.125.2 Member Function Documentation

8.125.2.1 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read

8.125.2.2 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take

8.125.2.3 void read_w_condition (KeyedStringSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Accesses via **com.rti.dds.type.builtin.KeyedStringDataReader.read** (p. 1128) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition

8.125.2.4 void take_w_condition (KeyedStringSeq *received_data*, SampleInfoSeq *info_seq*, int *max_samples*, ReadCondition *condition*)

Analogous to **com.rti.dds.type.builtin.KeyedStringDataReader.read_w_condition** (p. 1128) except it accesses samples via the **com.rti.dds.type.builtin.KeyedStringDataReader.take** (p. 1128) operation.

See also:

com.rti.dds.topic.example.FooDataReader.take_w_condition

8.125.2.5 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_sample

8.125.2.6 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_sample

8.125.2.7 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_instance

8.125.2.8 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_instance

8.125.2.9 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.KeyedStringDataReader.read_instance** (p. 1129) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_instance_w_condition

8.125.2.10 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.KeyedStringDataReader.take_instance** (p. 1129) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.take_instance_w_condition

8.125.2.11 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance

8.125.2.12 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance

8.125.2.13 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.KeyedStringDataReader.read_next_instance** (p. 1130) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_instance_w_condition

8.125.2.14 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.KeyedStringDataReader.take_next_instance** (p. 1131) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_instance_w_condition

8.125.2.15 void return_loan (KeyedStringSeq *received_data*, SampleInfoSeq *info_seq*)

Indicates to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.return_loan

8.125.2.16 void get_key_value (KeyedString *key_holder*, InstanceHandle_t *handle*)

Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.125.2.17 String get_key_value (InstanceHandle_t *handle*)

<<***eXtension***>> *(p. 270)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataReader.get_key_value

8.125.2.18 InstanceHandle_t lookup_instance (KeyedString *key_holder*)

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also:

com.rti.dds.topic.example.FooDataReader.lookup_instance

8.125.2.19 InstanceHandle_t lookup_instance (String *key*)

<<***eXtension***>> *(p. 270)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

com.rti.dds.topic.example.FooDataReader.lookup_instance

8.126 KeyedStringDataWriter Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.KeyedString** (p. 1123) >.

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. 270) *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.KeyedStringDataWriter.register_instance*** *(p. 1135) 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. 270) *Performs the same functions as* ***KeyedStringDataWriter.register_instance*** *(p. 1135) 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.KeyedStringDataWriter.register_instance*** *(p. 1135).*

- void **unregister_instance** (String key, **InstanceHandle_t** handle)

  <<**eXtension**>> (p. 270) *Reverses the action of* ***KeyedStringDataWriter.register_instance*** *(p. 1135).*

- 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.KeyedStringDataWriter.unregister_instance*** *(p. 1136) 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. 270) *Performs the same function as* ***KeyedStringDataWriter.unregister_instance*** *(p. 1136) 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.KeyedString*** *(p. 1123) data instance.*

^ void **write** (String key, String str, **InstanceHandle_t** handle)

<<**eXtension**>> (p. 270) *Modifies the value of a* ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123) 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.KeyedStringDataWriter.write*** *(p. 1137) 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. 270) *Performs the same function as* ***KeyedStringDataWriter.write*** *(p. 1137) 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. 270) *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.KeyedStringDataWriter.dispose*** *(p. 1138) except that the application provides the value for the* `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

^ void **dispose_w_timestamp** (String key, **InstanceHandle_t** instance_handle, **Time_t** source_timestamp)

<<**eXtension**>> (p. 270) *Performs the same functions as* ***KeyedStringDataWriter.dispose*** *(p. 1138) except that the application provides the value for the* `source_timestamp` *that is made available to* ***com.rti.dds.subscription.DataReader*** *(p. 473) objects by means of the* `source_timestamp` *attribute inside the* ***com.rti.dds.subscription.SampleInfo*** *(p. 1404).*

- 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. 270) *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. 270) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

8.126.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataWriter` < **com.rti.dds.type.builtin.KeyedString** (p. 1123) >.

See also:

com.rti.dds.topic.example.FooDataWriter
com.rti.dds.publication.DataWriter (p. 538)

8.126.2 Member Function Documentation

8.126.2.1 InstanceHandle_t register_instance (KeyedString *instance_data*)

Informs RTI Connext that the application will be modifying a particular instance.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance

8.126.2.2 InstanceHandle_t register_instance (String *key*)

<<eXtension>> (p. 270) Informs RTI Connext that the application will be modifying a particular instance.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance

8.126.2.3 InstanceHandle_t register_instance_w_timestamp (KeyedString *instance_data*, Time_t *source_timestamp*)

Performs the same functions as **com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance** (p. 1135) except that the application provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp

8.126.2.4 InstanceHandle_t register_instance_w_timestamp (String *key*, Time_t *source_timestamp*)

<<eXtension>> (p. 270) Performs the same functions as **KeyedStringDataWriter.register_instance** (p. 1135) except that the application provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.register_instance_w_timestamp

8.126.2.5 void unregister_instance (KeyedString *instance_data*, InstanceHandle_t *handle*)

Reverses the action of **com.rti.dds.type.builtin.KeyedStringDataWriter.register_instance** (p. 1135).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance

8.126.2.6 void unregister_instance (String *key*, InstanceHandle_t *handle*)

<<*eXtension*>> *(p. 270)* Reverses the action of **KeyedStringDataWriter.register_instance** (p. 1135).

See also:

com.rti.dds.topic.example.FooDataWriter.unregister_instance

8.126.2.7 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.KeyedStringDataWriter.unregister_instance** (p. 1136) except that it also provides the value for the `source_timestamp`.

See also:

FooDataWriter.unregister_instance_w_timestamp

8.126.2.8 void unregister_instance_w_timestamp (String *key*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

<<*eXtension*>> *(p. 270)* Performs the same function as **KeyedStringDataWriter.unregister_instance** (p. 1136) except that it also provides the value for the `source_timestamp`.

See also:

FooDataWriter.unregister_instance_w_timestamp

8.126.2.9 void write (KeyedString *instance_data*, InstanceHandle_t *handle*)

Modifies the value of a **com.rti.dds.type.builtin.KeyedString** (p. 1123) data instance.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.126.2.10 void write (String *key*, String *str*, InstanceHandle_t *handle*)

<<*eXtension*>> *(p. 270)* Modifies the value of a **com.rti.dds.type.builtin.KeyedString** (p. 1123) data instance.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.126.2.11 void write_w_timestamp (KeyedString *instance_data*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

Performs the same function as **com.rti.dds.type.builtin.KeyedStringDataWriter.write** (p. 1137) except that it also provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.126.2.12 void write_w_timestamp (String *key*, String *str*, InstanceHandle_t *handle*, Time_t *source_timestamp*)

<<*eXtension*>> *(p. 270)* Performs the same function as **KeyedStringDataWriter.write** (p. 1137) except that it also provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.126.2.13 void dispose (KeyedString *instance_data*, InstanceHandle_t *instance_handle*)

Requests the middleware to delete the data.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.126.2.14 void dispose (String *key*, InstanceHandle_t *instance_handle*)

<<*eXtension*>> *(p. 270)* Requests the middleware to delete the data.

See also:

com.rti.dds.topic.example.FooDataWriter.dispose

8.126.2.15 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.KeyedStringDataWriter.dispose** (p. 1138) except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp

8.126.2.16 void dispose_w_timestamp (String *key*, InstanceHandle_t *instance_handle*, Time_t *source_timestamp*)

<<*eXtension*>> *(p. 270)* Performs the same functions as **KeyedStringDataWriter.dispose** (p. 1138) except that the application provides the value for the `source_timestamp` that is made available to **com.rti.dds.subscription.DataReader** (p. 473) objects by means of the `source_timestamp` attribute inside the **com.rti.dds.subscription.SampleInfo** (p. 1404).

See also:

com.rti.dds.topic.example.FooDataWriter.dispose_w_timestamp

8.126.2.17 void get_key_value (KeyedString *key_holder*, InstanceHandle_t *handle*)

Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.126.2.18 String get_key_value (InstanceHandle_t *handle*)

*<<**eXtension**>>* *(p. 270)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

com.rti.dds.topic.example.FooDataWriter.get_key_value

8.126.2.19 InstanceHandle_t lookup_instance (KeyedString *key_holder*)

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also:

com.rti.dds.topic.example.FooDataWriter.lookup_instance

8.126.2.20 InstanceHandle_t lookup_instance (String *key*)

*<<**eXtension**>>* *(p. 270)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

com.rti.dds.topic.example.FooDataWriter.lookup_instance

8.127 KeyedStringSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.KeyedString** (p. 1123) > .

Inheritance diagram for KeyedStringSeq::

Public Member Functions

- **KeyedStringSeq** ()

  Constructs an empty sequence of ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123) objects with an initial maximum of zero.*

- **KeyedStringSeq** (int initialMaximum)

  Constructs an empty sequence of ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123) objects with the given initial maximum.*

- **KeyedStringSeq** (Collection elements)

  Constructs a new sequence containing the given ***com.rti.dds.type.builtin.KeyedString*** *(p. 1123) objects.*

- Object **copy_from** (Object src)

Package Attributes

- transient **Sequence** **_loanedInfoSequence** = null

8.127.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **com.rti.dds.type.builtin.KeyedString** (p. 1123) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.type.builtin.KeyedString (p. 1123)

8.127.2 Constructor & Destructor Documentation

8.127.2.1 KeyedStringSeq ()

Constructs an empty sequence of **com.rti.dds.type.builtin.KeyedString** (p. 1123) objects with an initial maximum of zero.

8.127.2.2 KeyedStringSeq (int *initialMaximum*)

Constructs an empty sequence of **com.rti.dds.type.builtin.KeyedString** (p. 1123) objects with the given initial maximum.

8.127.2.3 KeyedStringSeq (Collection *elements*)

Constructs a new sequence containing the given **com.rti.dds.type.builtin.KeyedString** (p. 1123) objects.

Parameters:

elements the initial contents of this sequence.

Exceptions:

NullPointerException if the input collection is null

8.127.3 Member Function Documentation

8.127.3.1 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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.127.4 Member Data Documentation

8.127.4.1 transient Sequence _loanedInfoSequence = null [package]

When a memory loan has been taken out in the lower layers of NDDS, store a pointer to the native sequence here. That way, when we call finish(), we can give the memory back.

8.128 KeyedStringTypeSupport Class Reference

<<*interface*>> *(*p. *271)* Keyed string type support.

Inheritance diagram for KeyedStringTypeSupport::

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.KeyedString*** *(*p. *1123) data type.*

- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the ***com.rti.dds.type.builtin.KeyedString*** *(*p. *1123) 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.KeyedString*** *(*p. *1123) type.*

8.128.1 Detailed Description

<<*interface*>> *(*p. *271)* Keyed string type support.

8.128.2 Member Function Documentation

8.128.2.1 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.KeyedString** (p. 1123) data type.

By default, The **com.rti.dds.type.builtin.KeyedString** (p. 1123) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by

com.rti.dds.type.builtin.KeyedStringTypeSupport.get_type_name (p. 1146). 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.KeyedStringTypeSupport** (p. 1144) with a **com.rti.dds.domain.DomainParticipant** (p. 629) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 629) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>>* *(p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to register the data type **com.rti.dds.type.builtin.KeyedString** (p. 1123) with. Cannot be null.

type_name *<<in>>* *(p. 271)* the type name under with the data type **com.rti.dds.type.builtin.KeyedString** (p. 1123) is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1545). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_OUT_OF_RESOURCES.

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. 670)

8.128.2.2 static void unregister_type (DomainParticipant *participant*, String *type_name*) [static]

Allows an application to unregister the **com.rti.dds.type.builtin.KeyedString** (p. 1123) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **com.rti.dds.type.builtin.KeyedString** (p. 1123) type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1545) 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. 1545) is associated with the type, the operation will fail with RETCODE_ERROR.

Postcondition:

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to unregister the data type **com.rti.dds.type.builtin.KeyedString** (p. 1123) from. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type **com.rti.dds.type.builtin.KeyedString** (p. 1123) 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. 104), RETCODE_BAD_PARAMETER or RETCODE_ERROR

MT Safety:

SAFE.

See also:

com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type (p. 1144)

8.128.2.3 static String get_type_name () [static]

Get the default name for the **com.rti.dds.type.builtin.KeyedString** (p. 1123) type.

Can be used for calling **com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type** (p. 1144) or creating **com.rti.dds.topic.Topic** (p. 1545).

Returns:

default name for the **com.rti.dds.type.builtin.KeyedString** (p. 1123) type.

See also:

com.rti.dds.type.builtin.KeyedStringTypeSupport.register_type (p. 1144)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.129 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.

8.129.1 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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **YES** (p. 98)

See also:

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)
com.rti.dds.publication.FlowController (p. 942)

8.129.2 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 **PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1312) **com.rti.dds.publication.DataWriter** (p. 538) instances associated with a FlowControllerSchedulingPolicy.EDF_FLOW_CONTROLLER_SCHED_POLICY **com.rti.dds.publication.FlowController** (p. 942) only. Together with the time of write, **com.rti.dds.infrastructure.LatencyBudgetQosPolicy.duration** (p. 1149) 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. 387).

8.129.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* <= *requested duration* evaluates to 'TRUE'.

8.129.4 Member Data Documentation

8.129.4.1 final Duration_t duration

Duration of the maximum acceptable delay.

[**default**] 0 (meaning minimize the delay)

8.130 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 char **release**

  The release letter of a single RTI Connext library.

- final int **build**

  The build number of a single RTI Connext library.

8.130.1 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.

8.130.2 Member Data Documentation

8.130.2.1 final int major

The major version of a single RTI Connext library.

8.130.2.2 final int minor

The minor version of a single RTI Connext library.

8.130.2.3 final char release

The release letter of a single RTI Connext library.

8.130.2.4 final int build

The build number of a single RTI Connext library.

8.131 LifespanQosPolicy Class Reference

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 538) is considered valid.

Inheritance diagram for LifespanQosPolicy::

Public Attributes

- final **Duration_t duration**

  Maximum duration for the data's validity.

8.131.1 Detailed Description

Specifies how long the data written by the **com.rti.dds.publication.DataWriter** (p. 538) is considered valid.

Each data sample written by the **com.rti.dds.publication.DataWriter** (p. 538) 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. 473) 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. 538)'s cache is computed by adding the duration specified by this QoS policy to the sample's source timestamp. The expiration time of each sample from the **com.rti.dds.subscription.DataReader** (p. 473)'s cache is computed by adding the duration to the reception timestamp.

See also:

com.rti.dds.topic.example.FooDataWriter.write
com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

Entity:

com.rti.dds.topic.Topic (p. 1545), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

8.131.2 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** (p. 350) (see **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071)), 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. 1410)
com.rti.dds.subscription.SampleInfo.reception_timestamp (p. 1412)

8.131.3 Member Data Documentation

8.131.3.1 final Duration_t duration

Maximum duration for the data's validity.

[**default**] com.rti.dds.infrastructure.Duration_t.INFINITE

[**range**] [1 nanosec, 1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.132 Listener Interface Reference

<<*interface*>> *(p. 271)* Abstract base class for all **Listener** (p. 1154) interfaces.

Inheritance diagram for Listener::

8.132.1 Detailed Description

<<*interface*>> *(p. 271)* Abstract base class for all **Listener** (p. 1154) interfaces.

Entity:

com.rti.dds.infrastructure.Entity (p. 912)

QoS:

QoS Policies (p. 90)

Status:

Status Kinds (p. 106)

All the supported kinds of concrete **com.rti.dds.infrastructure.Listener** (p. 1154) interfaces (one per concrete **com.rti.dds.infrastructure.Entity** (p. 912) type) derive from this root and add methods whose prototype depends on the concrete **Listener** (p. 1154).

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. 1154) 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. 912).

8.132.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 106)) 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. 106).

- on_<communication_status> is available on the relevant **com.rti.dds.infrastructure.Entity** (p. 912) as well as those that embed it, as expressed in the following figure:

- 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. 912) 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. 629)) and to set dedicated behaviors only where needed.

8.132.3 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. 1505) with a parameter of the related **com.rti.dds.subscription.Subscriber** (p. 1478);

- 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. 503) on all the related **com.rti.dds.subscription.DataReaderListener** (p. 501) objects, with a parameter of the related **com.rti.dds.subscription.DataReader** (p. 473).

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. 473) objects by calling **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1491) on the related **com.rti.dds.subscription.Subscriber** (p. 1478) and then get the data by calling com.rti.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take on the returned **com.rti.dds.subscription.DataReader** (p. 473) objects), or it wants to treat each **com.rti.dds.subscription.DataReader** (p. 473) independently and it may choose the second option (and then get the data by calling com.rti.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take on the related **com.rti.dds.subscription.DataReader** (p. 473)).

Note that if **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1505) is called, RTI Connext will *not* try to call **com.rti.dds.subscription.DataReaderListener.on_data_available** (p. 503). However, an application can force a call to the **com.rti.dds.subscription.DataReader** (p. 473) objects that have data by calling **com.rti.dds.subscription.Subscriber.notify_datareaders** (p. 1493).

8.132.4 Operations Allowed in Listener Callbacks

The operations that are allowed in **com.rti.dds.infrastructure.Listener** (p. 1154) callbacks depend on the **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy** (p. 933) QoS policy of the **com.rti.dds.infrastructure.Entity** (p. 912) to which the **com.rti.dds.infrastructure.Listener** (p. 1154) is attached – or in the case of a **com.rti.dds.publication.DataWriter** (p. 538) of **com.rti.dds.subscription.DataReader** (p. 473) listener, on the **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy** (p. 933) QoS of the parent **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478). For instance, the **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy** (p. 933) settings of a **com.rti.dds.subscription.Subscriber** (p. 1478) will determine which operations are allowed within the callbacks of the listeners associated with all the DataReaders created through that **com.rti.dds.subscription.Subscriber** (p. 1478).

Note: these restrictions do not apply to builtin **topic** (p. 350) listener callbacks.

Regardless of whether **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy.use_shared_exclusive_area** (p. 934) is set to true or false, the following operations are *not* allowed:

- Within any listener callback, deleting the entity to which the **com.rti.dds.infrastructure.Listener** (p. 1154) is attached

- Within a **com.rti.dds.topic.Topic** (p. 1545) listener callback, any operations on any subscribers, readers, publishers or writers

An attempt to call a disallowed method from within a callback will result in **RETCODE_ILLEGAL_OPERATION** (p. 1365).

If **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy.use_shared_exclusive_area** (p. 934) is set to false, the setting which allows more concurrency among RTI Connext threads, the following are *not* allowed:

- Within any listener callback, creating any entity
- Within any listener callback, deleting any entity
- Within any listener callback, enabling any entity
- Within any listener callback, setting the QoS of any entities
- Within a **com.rti.dds.subscription.DataReader** (p. 473) or **com.rti.dds.subscription.Subscriber** (p. 1478) listener callback, invoking any operation on any other **com.rti.dds.subscription.Subscriber** (p. 1478) or on any **com.rti.dds.subscription.DataReader** (p. 473) belonging to another **com.rti.dds.subscription.Subscriber** (p. 1478).
- Within a **com.rti.dds.subscription.DataReader** (p. 473) or **com.rti.dds.subscription.Subscriber** (p. 1478) listener callback, invoking any operation on any **com.rti.dds.publication.Publisher** (p. 1277) (or on any **com.rti.dds.publication.DataWriter** (p. 538) belonging to such a **com.rti.dds.publication.Publisher** (p. 1277)) that has **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy.use_shared_exclusive_area** (p. 934) set to true.
- Within a **com.rti.dds.publication.DataWriter** (p. 538) of **com.rti.dds.publication.Publisher** (p. 1277) listener callback, invoking any operation on another Publisher or on a **com.rti.dds.publication.DataWriter** (p. 538) belonging to another **com.rti.dds.publication.Publisher** (p. 1277).
- Within a **com.rti.dds.publication.DataWriter** (p. 538) of **com.rti.dds.publication.Publisher** (p. 1277) listener callback, invoking any operation on any **com.rti.dds.subscription.Subscriber** (p. 1478) or **com.rti.dds.subscription.DataReader** (p. 473).

An attempt to call a disallowed method from within a callback will result in **RETCODE_ILLEGAL_OPERATION** (p. 1365).

The above limitations can be lifted by setting **com.rti.dds.infrastructure.ExclusiveAreaQosPolicy.use_shared_exclusive_area** (p. 934) to true on the **com.rti.dds.publication.Publisher**

(p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478) (or on the **com.rti.dds.publication.Publisher** (p. 1277)/ **com.rti.dds.subscription.Subscriber** (p. 1478) of the **com.rti.dds.publication.DataWriter** (p. 538)/com.**rti.dds.subscription.DataReader** (p. 473)) to which the listener is attached. However, the application will pay the cost of reduced concurrency between the affected publishers and subscribers.

See also:

EXCLUSIVE_AREA (p. 72)
Status Kinds (p. 106)
com.rti.dds.infrastructure.WaitSet (p. 1695),
com.rti.dds.infrastructure.Condition (p. 451)

8.133 LivelinessChangedStatus Class Reference

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. 538) entities that write the* ***com.rti.dds.topic.Topic*** *(p. 1545) the* ***com.rti.dds.subscription.DataReader*** *(p. 473) reads.*

- int **not_alive_count**

  The total count of currently not_alive ***com.rti.dds.publication.DataWriter*** *(p. 538) entities that write the* ***com.rti.dds.topic.Topic*** *(p. 1545) the* ***com.rti.dds.subscription.DataReader*** *(p. 473) 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**

  An instance handle to the last remote writer to change its liveliness.

8.133.1 Detailed Description

StatusKind.LIVELINESS_CHANGED_STATUS.

The **com.rti.dds.subscription.DataReaderListener.on_liveliness_changed** (p. 503) callback may be invoked for the following reasons:

- Liveliness is truly lost - a sample has not been received within the timeframe specified in **com.rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1167)
- Liveliness is recovered after being lost.
- A new matching entity has been discovered.
- A QoS has changed such that a pair of matching entities are no longer matching (such as a change to the **com.rti.dds.infrastructure.PartitionQosPolicy** (p. 1233)). In this case, RTI Connext will no longer keep track of the entities' liveliness. Furthermore:
  - If liveliness was maintained: **com.rti.dds.subscription.LivelinessChangedStatus.alive_count** (p. 1161) will decrease and **com.rti.dds.subscription.LivelinessChangedStatus.not_alive_count** (p. 1161) will remain the same.
  - If liveliness had been lost: **com.rti.dds.subscription.LivelinessChangedStatus.alive_count** (p. 1161) will remain the same and **com.rti.dds.subscription.LivelinessChangedStatus.not_alive_count** (p. 1161) will decrease.

8.133.2 Constructor & Destructor Documentation

8.133.2.1 LivelinessChangedStatus ()

The no-argument constructor for this status object.

8.133.2.2 LivelinessChangedStatus (LivelinessChangedStatus *src*)

A copy constructor.

Exceptions:

NullPointerException if the given status is null.

8.133.3 Member Data Documentation

8.133.3.1 int alive_count

The total count of currently alive **com.rti.dds.publication.DataWriter** (p. 538) entities that write the **com.rti.dds.topic.Topic** (p. 1545) the **com.rti.dds.subscription.DataReader** (p. 473) reads.

8.133.3.2 int not_alive_count

The total count of currently not_alive **com.rti.dds.publication.DataWriter** (p. 538) entities that write the **com.rti.dds.topic.Topic** (p. 1545) the **com.rti.dds.subscription.DataReader** (p. 473) reads.

8.133.3.3 int alive_count_change

The change in the alive_count since the last time the listener was called or the status was read.

8.133.3.4 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.

8.133.3.5 final InstanceHandle_t last_publication_handle

An instance handle to the last remote writer to change its liveliness.

8.134 LivelinessLostStatus Class Reference

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. 538) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.*

- int **total_count_change**

  The incremental changees in total_count since the last time the listener was called or the status was read.

8.134.1 Detailed Description

StatusKind.LIVELINESS_LOST_STATUS.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

The liveliness that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164) was not respected; thus **com.rti.dds.subscription.DataReader** (p. 473) entities will consider the **com.rti.dds.publication.DataWriter** (p. 538) as no longer "alive/active".

8.134.2 Member Data Documentation

8.134.2.1 int total_count

Total cumulative number of times that a previously-alive **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) simply remains not alive for another liveliness period.

8.134.2.2 int total_count_change

The incremental changees in total_count since the last time the listener was called or the status was read.

8.135 LivelinessQosPolicy Class Reference

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 473) entities to detect when **com.rti.dds.publication.DataWriter** (p. 538) entities become disconnected or "dead.".

Inheritance diagram for LivelinessQosPolicy::

Public Attributes

- ˆ **LivelinessQosPolicyKind kind**

  The kind of liveliness desired.

- ˆ final **Duration_t lease_duration**

  *The duration within which a **com.rti.dds.infrastructure.Entity** (p. 912) must be asserted, or else it is assumed to be not alive.*

8.135.1 Detailed Description

Specifies and configures the mechanism that allows **com.rti.dds.subscription.DataReader** (p. 473) entities to detect when **com.rti.dds.publication.DataWriter** (p. 538) entities become disconnected or "dead.".

Liveliness must be asserted at least once every `lease_duration` otherwise RTI Connext will assume the corresponding **com.rti.dds.infrastructure.Entity** (p. 912) or is no longer alive.

The liveliness status of a **com.rti.dds.infrastructure.Entity** (p. 912) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 83) policy. The application is also informed via **com.rti.dds.infrastructure.Listener** (p. 1154) when an **com.rti.dds.infrastructure.Entity** (p. 912) is no longer alive.

A **com.rti.dds.subscription.DataReader** (p. 473) requests that liveliness of writers is maintained by the requested means and loss of liveliness is detected with delay not to exceed the `lease_duration`.

A **com.rti.dds.publication.DataWriter** (p. 538) commits to signalling its liveliness using the stated means at intervals not to exceed the `lease_duration`.

Listeners are used to notify a **com.rti.dds.subscription.DataReader** (p. 473) of loss of liveliness and **com.rti.dds.publication.DataWriter** (p. 538) of vi-

olations 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. 538) 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 run time 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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.LIVELINESS_LOST_STATUS (p. 1461), **com.rti.dds.publication.LivelinessLostStatus** (p. 1162); **StatusKind.LIVELINESS_CHANGED_STATUS** (p. 1461), **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1159); **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459), **StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.135.2 Usage

This policy controls the mechanism and parameters used by RTI Connext to ensure that particular entities on the network are still alive. The liveliness can also affect the ownership of a particular instance, as determined by the **OWNERSHIP** (p. 83) 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 **LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS** (p. 1169) 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. 629) is running and the link

connecting it to remote participants remains connected, the entities within the **com.rti.dds.domain.DomainParticipant** (p. 629) will be considered alive. This requires the lowest overhead.

The manual settings (**LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1169), **LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1169)) 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. 912) is still alive. The action can be explicit by calling the **com.rti.dds.publication.DataWriter.assert_liveliness** (p. 554) operation or implicit by writing some data.

The two possible manual settings control the granularity at which the application must assert liveliness.

- The setting **LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1169) requires only that one **com.rti.dds.infrastructure.Entity** (p. 912) within a participant is asserted to be alive to deduce all other **com.rti.dds.infrastructure.Entity** (p. 912) objects within the same **com.rti.dds.domain.DomainParticipant** (p. 629) are also alive.

- The setting **LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1169) requires that at least one instance within the **com.rti.dds.publication.DataWriter** (p. 538) is asserted.

Changes in **LIVELINESS** (p. 78) must be detected by the Service with a time-granularity greater or equal to the `lease_duration`. This ensures that the value of the **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1159) is updated at least once during each `lease_duration` and the related Listeners and **com.rti.dds.infrastructure.WaitSet** (p. 1695) s are notified within a `lease_duration` from the time the **LIVELINESS** (p. 78) changed.

8.135.3 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. 1168) kind are considered ordered such that: **LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS**

(p. 1169) < **LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1169) < **LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1169).

^ the inequality *offered* `lease_duration` <= *requested* `lease_duration` evaluates to true.

See also:

RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 1218)

8.135.4 Member Data Documentation

8.135.4.1 LivelinessQosPolicyKind kind

The kind of liveliness desired.

[**default**] **LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS** (p. 1169)

8.135.4.2 final Duration_t lease_duration

The duration within which a **com.rti.dds.infrastructure.Entity** (p. 912) must be asserted, or else it is assumed to be not alive.

[**default**] com.rti.dds.infrastructure.Duration_t.INFINITE

[**range**] [0,1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.136 LivelinessQosPolicyKind Class Reference

Kinds of liveliness.

Inheritance diagram for LivelinessQosPolicyKind::

Static Public Attributes

- static final **LivelinessQosPolicyKind AUTOMATIC_LIVELINESS_QOS**

  [**default**] *The **infrastructure** (p. 323) will automatically signal liveliness for the **com.rti.dds.publication.DataWriter** (p. 538) (s) at least as often as required by the* `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. 538) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 629) (or the **com.rti.dds.domain.DomainParticipant** (p. 629) itself) has asserted its liveliness, then the other Entities belonging to that same **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 538) if the application has asserted liveliness of that **com.rti.dds.publication.DataWriter** (p. 538) itself.*

8.136.1 Detailed Description

Kinds of liveliness.

QoS:

com.rti.dds.infrastructure.LivelinessQosPolicy (p. 1164)

8.136.2 Member Data Documentation

8.136.2.1 final LivelinessQosPolicyKind AUTOMATIC_LIVELINESS_QOS [static]

[**default**] The **infrastructure** (p. 323) will automatically signal liveliness for the **com.rti.dds.publication.DataWriter** (p. 538) (s) at least as often as required by the `lease_duration`.

A **com.rti.dds.publication.DataWriter** (p. 538) with this setting does not need to take any specific action in order to be considered 'alive.' The **com.rti.dds.publication.DataWriter** (p. 538) 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.

8.136.2.2 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. 538) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 629) (or the **com.rti.dds.domain.DomainParticipant** (p. 629) itself) has asserted its liveliness, then the other Entities belonging to that same **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 690), or by calling **com.rti.dds.publication.DataWriter.assert_liveliness** (p. 554), or com.rti.dds.topic.example.FooDataWriter.write on any **com.rti.dds.publication.DataWriter** (p. 538) belonging to the **com.rti.dds.domain.DomainParticipant** (p. 629).

8.136.2.3 final LivelinessQosPolicyKind MANUAL_BY_TOPIC_LIVELINESS_QOS [static]

RTI Connext will only assume liveliness of the **com.rti.dds.publication.DataWriter** (p. 538) if the application has asserted liveliness of that **com.rti.dds.publication.DataWriter** (p. 538) itself.

The user application takes responsibility to signal liveliness to RTI Connext using the **com.rti.dds.publication.DataWriter.assert_liveliness** (p. 554) method, or by writing some data.

8.137 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.

8.137.1 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. 490)(java.util.List, **com.rti.dds.subscription.SampleInfoSeq** (p. 1414), int, int, int, int)
com.rti.dds.subscription.DataReader.take_untyped (p. 490)(java.util.List, **com.rti.dds.subscription.SampleInfoSeq** (p. 1414), int, int, int, int)

8.137.2 Constructor & Destructor Documentation

8.137.2.1 LoanableSequence (Class *elementType*)

Construct a new sequence for elements of the given type.

8.137.2.2 LoanableSequence (Class *elementType*, int *maximum*)

Construct a new sequence for elements of the given type.

8.137.2.3 LoanableSequence (Class *elementType*, Collection *elements*)

Construct a new sequence for elements of the given type.

8.137.3 Member Function Documentation

8.137.3.1 final boolean hasOwnership ()

Return the value of the owned flag.

Returns:

true if sequence owns the underlying buffer, or false if it has an outstanding loan.

8.137.3.2 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. 383), or explicitly by calling **Sequence.setMaximum** (p. 1433).

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.137.3.3 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. 383), the sequence's size is increased implicitly.

Postcondition:

length == MINIMUM(original length, new_max)

Parameters:

new_max Must be >= 0.

Returns:

true on success, false if the preconditions are not met. In that case the sequence is not modified.

Reimplemented from **AbstractSequence** (p. 382).

8.137.3.4 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)

8.137.3.5 Object get (int *index*)

Returns the element at the specified position in this sequence.

See also:

java.util.List.get(int)

8.137.3.6 int size ()

Returns the length of the sequence.

See also:

java.util.List.get(int)

8.138 Locator_t Class Reference

*<<**eXtension**>>* *(p. 270)* 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. 1176) 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.

ˆ static final int **KIND_SHMEM**

A locator for an address acessed via shared memory.

ˆ static final int **KIND_UDPv6**

A locator for a UDPv6 address.

ˆ static final int **KIND_RESERVED**

Locator of this kind is reserved.

ˆ static final int **ADDRESS_LENGTH_MAX** = 16

Declares length of address field in locator.

8.138.1 Detailed Description

<<***eXtension***>> *(*p. 270*)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

8.138.2 Constructor & Destructor Documentation

8.138.2.1 Locator_t ()

Constructor.

8.138.3 Member Data Documentation

8.138.3.1 final int KIND_INVALID [static]

Locator of this kind is invalid.

8.138.3.2 final int PORT_INVALID [static]

An invalid port.

8.138.3.3 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.

8.138.3.4 final Locator_t INVALID [static]

Initial value:

```
new Locator_t(
       KIND_INVALID, PORT_INVALID, ADDRESS_INVALID)
```

An invalid locator.

8.138.3.5 final int KIND_UDPv4 [static]

A locator for a UDPv4 address.

8.138.3.6 final int KIND_SHMEM [static]

A locator for an address acessed via shared memory.

8.138.3.7 final int KIND_UDPv6 [static]

A locator for a UDPv6 address.

8.138.3.8 final int KIND_RESERVED [static]

Locator of this kind is reserved.

8.138.3.9 final int ADDRESS_LENGTH_MAX = 16 [static]

Declares length of address field in locator.

8.138.3.10 int kind

The kind of locator.

If the **Locator_t** (p. 1174) kind is **com.rti.dds.infrastructure.Locator_t.KIND_UDPv4** (p. 1176), the address contains an IPv4 address. In this

case, the leading 12 octets of the **com.rti.dds.infrastructure.Locator_t.address** (p. 1177) must be zero. The last 4 octets of **com.rti.dds.infrastructure.Locator_t.address** (p. 1177) are used to store the IPv4 address.

If the **Locator_t** (p. 1174) kind is **com.rti.dds.infrastructure.Locator_t.KIND_UDPv6** (p. 1176), 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. 1177) field.

8.138.3.11 int port

the port number

8.138.3.12 final byte [] address = new byte[ADDRESS_LENGTH_MAX]

A **com.rti.dds.infrastructure.Locator_t.ADDRESS_LENGTH_MAX** (p. 1176) octet field to hold the IP address.

8.139 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 four ***com.rti.dds.infrastructure.Locator_t*** *(p. 1174), 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.

8.139.1 Detailed Description

Specifies the configuration of an individual channel within a MultiChannel DataWriter.

QoS:

com.rti.dds.infrastructure.LocatorFilterQosPolicy (p. 1181)

8.139.2 Constructor & Destructor Documentation

8.139.2.1 LocatorFilter_t ()

Constructor.

8.139.2.2 LocatorFilter_t (LocatorFilter_t *src*)

Constructor.

Parameters:

src *<<**in**>>* *(*p. 271*)* Locator used to initialized the new locator.

8.139.2.3 LocatorFilter_t (LocatorSeq *locators*, String *filter_expression*)

Constructor.

Parameters:

locators *<<**in**>>* *(*p. 271*)* Locators.

filter_expression *<<**in**>>* *(*p. 271*)* Filter expression.

8.139.3 Member Data Documentation

8.139.3.1 LocatorSeq locators

Initial value:

```
new LocatorSeq()
```

Sequence containing from one to four **com.rti.dds.infrastructure.Locator_t** (p. 1174), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.

[**default**] Empty sequence.

8.139.3.2 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. 1182)

See also:

Queries and Filters Syntax (p. 278)

[default] NULL (invalid value)

8.140 LocatorFilterQosPolicy Class Reference

The QoS policy used to report the configuration of a MultiChannel DataWriter as part of builtin.PublicationBuiltinTopicData.

Inheritance diagram for LocatorFilterQosPolicy::

Public Attributes

- final **LocatorFilterSeq locator_filters**

  A sequence of ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1178). Each* ***com.rti.dds.infrastructure.LocatorFilter_t*** *(p. 1178) 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.

8.140.1 Detailed Description

The QoS policy used to report the configuration of a MultiChannel DataWriter as part of builtin.PublicationBuiltinTopicData.

Entity:

builtin.PublicationBuiltinTopicData

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.140.2 Member Data Documentation

8.140.2.1 final LocatorFilterSeq locator_filters

A sequence of **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1178). Each **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1178) reports the configuration of a single channel of a MultiChannel DataWriter.

A sequence length of zero indicates the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) is not in use.

[default] Empty sequence.

8.140.2.2 String filter_name

Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported: DomainParticipant.SQLFILTER_NAME and DomainParticipant.STRINGMATCHFILTER_NAME.

[default] DomainParticipant.STRINGMATCHFILTER_NAME

8.141 LocatorFilterSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1178) `>`.

Inherits ArraySequence.

8.141.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1178) `>`.

A sequence of **com.rti.dds.infrastructure.LocatorFilter_t** (p. 1178) used to report the channels' properties. If the length of the sequence is zero, the **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205) is not in use.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.LocatorFilter_t (p. 1178)

8.142 LocatorSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.Locator_t** (p. 1174) >.

Inherits ArraySequence.

8.142.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.Locator_t** (p. 1174) >.

See also:

com.rti.dds.infrastructure.Locator_t (p. 1174)

8.143 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 the discovery process are in this 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.

8.143.1 Detailed Description

Categories of logged messages.

The **com.rti.ndds.config.Logger.get_verbosity_by_category** (p. 1188) and **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1189) can be used to specify different verbosities for different categories of messages.

8.143.2 Member Data Documentation

8.143.2.1 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.

8.143.2.2 final LogCategory NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION [static]

Log messages pertaining to data serialization and deserialization and network traffic are in this category.

8.143.2.3 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.

8.143.2.4 final LogCategory NDDS_CONFIG_LOG_CATEGORY_ENTITIES [static]

Log messages pertaining to local and remote entities and to the discovery process are in this category.

8.143.2.5 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.

8.144 Logger Class Reference

<<*interface*>> *(p. 271)* 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.

- **LogPrintFormat get_print_format** ()

  Get the current message format 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.

Static Public Member Functions

- static **Logger get_instance** ()

  Get the singleton instance of this type.

8.144.1 Detailed Description

<<*interface*>> *(p. 271)* The singleton type used to configure RTI Connext logging.

8.144.2 Member Function Documentation

8.144.2.1 static Logger get_instance () [static]

Get the singleton instance of this type.

8.144.2.2 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. 1188) is never called is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1196).

If **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1189) has been used to set different verbosities for different categories of messages, this method will return the maximum verbosity of all categories.

8.144.2.3 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. 1188) and **com.rti.ndds.config.Logger.set_verbosity_by_category** (p. 1189) are never called is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1196).

8.144.2.4 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. 1196) or below. (The default verbosity if you never set it is **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1196).)

8.144.2.5 void set_verbosity_by_category (LogCategory *category*, LogVerbosity *verbosity*)

Set the verbosity at which RTI Connext will log diagnostic information in the given category.

8.144.2.6 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. 1189), this method will return NULL. In this case, logged output will on most platforms go to standard out as if through printf.

8.144.2.7 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).

8.144.2.8 LogPrintFormat get_print_format ()

Get the current message format that RTI Connext is using to log diagnostic information.

If **com.rti.ndds.config.Logger.set_print_format** (p. 1189) is never called, the default format is **com.rti.ndds.config.LogPrintFormat.NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1193).

8.144.2.9 boolean set_print_format (LogPrintFormat *print_format*)

Set the message format that RTI Connext will use to log diagnostic information.

8.145 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.

8.145.1 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. 708)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **Changeable** (p. 98)

8.145.2 Member Data Documentation

8.145.2.1 LogVerbosity verbosity

The verbosities at which RTI Connext diagnostic information is logged.

[**default**] **com.rti.ndds.config.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1196)

8.145.2.2 LogCategory category

Categories of logged messages.

[**default**] Logging will be enabled for all the categories.

8.145.2.3 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. 1193).

8.145.2.4 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.

[**default**] NULL

8.146 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**

  Print message, method name, and activity context (default).

- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED**

  Print message, method name, activity context, and timestamp.

- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE**

  Print message with all available context information (includes thread identifier, activity context).

- 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 field that may be useful for internal debug.

- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL**

  Print only message number and method name.

- static final **LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL**

  Print all available fields.

8.146.1 Detailed Description

The format used to output RTI Connext diagnostic information.

8.146.2 Member Data Documentation

8.146.2.1 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT [static]

Print message, method name, and activity context (default).

8.146.2.2 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED [static]

Print message, method name, activity context, and timestamp.

8.146.2.3 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE [static]

Print message with all available context information (includes thread identifier, activity context).

8.146.2.4 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED [static]

Print message with all available context information, and timestamp.

8.146.2.5 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG [static]

Print a set of field that may be useful for internal debug.

8.146.2.6 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL [static]

Print only message number and method name.

8.146.2.7 final LogPrintFormat NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL [static]

Print all available fields.

8.147 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 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.

8.147.1 Detailed Description

The verbosities at which RTI Connext diagnostic information is logged.

8.147.2 Member Data Documentation

8.147.2.1 final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_SILENT [static]

No further output will be logged.

8.147.2.2 final LogVerbosity NDDS_CONFIG_LOG_VERBOSITY_ERROR [static]

Only 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.

8.147.2.3 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.

8.147.2.4 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.

8.147.2.5 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.

8.147.2.6 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.

8.148 LongDoubleSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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.

8.148.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < com.rti.dds.infrastructure.LongDouble >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.LongDouble
com.rti.dds.util.Sequence (p. 1432)

8.148.2 Constructor & Destructor Documentation

8.148.2.1 LongDoubleSeq ()

Constructs an empty sequence of long doubles with an initial maximum of zero.

8.148.2.2 LongDoubleSeq (int *initialMaximum*)

Constructs an empty sequence of long doubles with the given initial maximum.

8.148.2.3 LongDoubleSeq (double[] *doubles*)

Constructs a new sequence containing the given long doubles.

Parameters:

doubles the initial contents of this sequence

8.149 LongSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 1201) that return a java.lang.Long.*

- Object **set** (int index, Object element)

  *A wrapper for **setLong()** (p. 1202).*

- void **add** (int index, Object element)

  A wrapper for addLong(int, int).

8.149.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < long >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

long
com.rti.dds.util.Sequence (p. 1432)

8.149.2 Constructor & Destructor Documentation

8.149.2.1 LongSeq ()

Constructs an empty sequence of long integers with an initial maximum of zero.

8.149.2.2 LongSeq (int *initialMaximum*)

Constructs an empty sequence of long integers with the given initial maximum.

8.149.2.3 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

8.149.3 Member Function Documentation

8.149.3.1 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.

8.149.3.2 boolean addAllLong (long[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.149.3.3 void addLong (long *element*)

Append the element to the end of the sequence.

8.149.3.4 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.

8.149.3.5 long getLong (int *index*)

Returns the long at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.149.3.6 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.

8.149.3.7 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.

8.149.3.8 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.

8.149.3.9 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. 1204), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.149.3.10 Object get (int *index*) [virtual]

A wrapper for **getLong(int)** (p. 1201) that return a java.lang.Long.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.149.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setLong()** (p. 1202).

Exceptions:

ClassCastException if the element is not of type Long.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.149.3.12 void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.150 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. 441) 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 Multi-Channel DataWriter.

8.150.1 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. 538) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.150.2 Usage

By using this QoS, a **com.rti.dds.publication.DataWriter** (p. 538) 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.dds.topic.example.FooDataWriter.write, 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.

8.150.3 Member Data Documentation

8.150.3.1 final ChannelSettingsSeq channels

A sequence of **com.rti.dds.infrastructure.ChannelSettings_t** (p. 441) 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. 1205) is not in use.

The sequence length cannot be greater than **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.channel_seq_max_length** (p. 756).

[default] Empty sequence.

8.150.3.2 String filter_name

Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

The following builtin filters are supported: DomainParticipant.SQLFILTER_NAME and DomainParticipant.STRINGMATCHFILTER_NAME.

[**default**] DomainParticipant.STRINGMATCHFILTER_NAME

8.151 MultiTopic Interface Reference

[**Not supported (optional)**] <<***interface***>> *(p. 271)* A specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) 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. 1208).*

- void **get_expression_parameters** (**StringSeq** parameters)

  Get the expression parameters.

- void **set_expression_parameters** (**StringSeq** parameters)

  Set the `expression_parameters`.

8.151.1 Detailed Description

[**Not supported (optional)**] <<***interface***>> *(p. 271)* A specialization of **com.rti.dds.topic.TopicDescription** (p. 1561) that allows subscriptions that combine/filter/rearrange data coming from several topics.

com.rti.dds.topic.MultiTopic (p. 1208) allows a more sophisticated **subscription** (p. 343) 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. 473) entities associated with a **com.rti.dds.topic.MultiTopic** (p. 1208) are alerted of data modifications by the usual **com.rti.dds.infrastructure.Listener** (p. 1154) or **com.rti.dds.infrastructure.WaitSet** (p. 1695) / **com.rti.dds.infrastructure.Condition** (p. 451) mechanisms whenever modifications occur to the data associated with any of the topics relevant to the **com.rti.dds.topic.MultiTopic** (p. 1208).

Note that the source for data may not be restricted to a single **topic** (p. 350).

com.rti.dds.subscription.DataReader (p. 473) entities associated with a **com.rti.dds.topic.MultiTopic** (p. 1208) may access instances that are "constructed" at the **com.rti.dds.subscription.DataReader** (p. 473) side from the instances written by multiple **com.rti.dds.publication.DataWriter** (p. 538) entities. The **com.rti.dds.topic.MultiTopic** (p. 1208) access instance will begin to exist as soon as all the constituting **com.rti.dds.topic.Topic** (p. 1545) 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. 1208) instance is `ViewStateKind.NEW_VIEW_STATE` if at least one of the constituting instances has `view_state` = ViewStateKind.NEW_VIEW_STATE. Otherwise, it will be ViewStateKind.NOT_NEW_VIEW_STATE.

- The `instance_state` of the **com.rti.dds.topic.MultiTopic** (p. 1208) instance is InstanceStateKind.ALIVE_INTANCE_STATE if the `instance_state` of all the constituting **com.rti.dds.topic.Topic** (p. 1545) instances is InstanceStateKind.ALIVE_INTANCE_STATE. It is InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE if at least one of the constituting **com.rti.dds.topic.Topic** (p. 1545) instances is InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE. Otherwise, it is InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

Queries and Filters Syntax (p. 278) describes the syntax of `subscription_expression` and `expression_parameters`.

8.151.2 Member Function Documentation

8.151.2.1 String get_subscription_expression ()

Get the expression for this **com.rti.dds.topic.MultiTopic** (p. 1208).

The expressions syntax is described in the DDS specification. It is specified when the **com.rti.dds.topic.MultiTopic** (p. 1208) is created.

Returns:

`subscription_expression` of the **com.rti.dds.topic.MultiTopic** (p. 1208).

8.151.2.2 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. 1210), or if **com.rti.dds.topic.MultiTopic.set_expression_parameters** (p. 1210) was never called, the `parameters` specified when the **com.rti.dds.topic.MultiTopic** (p. 1208) was created.

Parameters:

parameters *<<**inout**>> (p. 271)* Fill in this sequence with the expression parameters. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.151.2.3 void set_expression_parameters (StringSeq *parameters*)

Set the `expression_parameters`.

Changes the `expression_parameters` associated with the **com.rti.dds.topic.MultiTopic** (p. 1208).

Parameters:

parameters *<<**in**>> (p. 271)* the filter expression parameters

Returns:

One of the **Standard Return Codes** (p. 104).

8.152 ObjectHolder Class Reference

<<*eXtension*>> *(p. 270)* Holder of object instance

Public Attributes

- ˆ Object **value** = null

  Instance of Object embedded in ***ObjectHolder*** *(p. 1211).*

8.152.1 Detailed Description

<<*eXtension*>> *(p. 270)* Holder of object instance

Holder of object instance. Can be used as an output parameter in a method for non-primitive types.

8.152.2 Member Data Documentation

8.152.2.1 Object value = null

Instance of Object embedded in **ObjectHolder** (p. 1211).

8.153 OfferedDeadlineMissedStatus Class Reference

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. 538) 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. 538) for which an offered deadline was missed.*

8.153.1 Detailed Description

StatusKind.OFFERED_DEADLINE_MISSED_STATUS.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

The deadline that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) was not respected for a specific instance.

8.153.2 Member Data Documentation

8.153.2.1 int total_count

Total cumulative count of the number of times the **com.rti.dds.publication.DataWriter** (p. 538) failed to write within its

offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

8.153.2.2 int total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

8.153.2.3 final InstanceHandle_t last_instance_handle

Handle to the last instance in the **com.rti.dds.publication.DataWriter** (p. 538) for which an offered deadline was missed.

8.154 OfferedIncompatibleQosStatus Class Reference

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. 538) discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545), common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).*

- 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. 1318) 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. 538) discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).*

8.154.1 Detailed Description

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

The qos policy value was incompatible with what was requested.

8.154.2 Member Data Documentation

8.154.2.1 int total_count

Total cumulative number of times the concerned **com.rti.dds.publication.DataWriter** (p. 538) discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545), common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

8.154.2.2 int total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

8.154.2.3 QosPolicyId_t last_policy_id

The **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1318) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.154.2.4 final QosPolicyCountSeq policies

A list containing for each policy the total number of times that the concerned **com.rti.dds.publication.DataWriter** (p. 538) discovered a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition with a requested QoS that is incompatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

8.155 OwnershipQosPolicy Class Reference

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 538) (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.

8.155.1 Detailed Description

Specifies whether it is allowed for multiple **com.rti.dds.publication.DataWriter** (p. 538) (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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

See also:

OWNERSHIP_STRENGTH (p. 84)

8.155.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 84), this QoS policy specifies if **com.rti.dds.subscription.DataReader** (p. 473) entities can re-

ceive updates to the same instance (identified by its key) from multiple **com.rti.dds.publication.DataWriter** (p. 538) entities at the same time.

There are two kinds of ownership, selected by the setting of the `kind:` SHARED and EXCLUSIVE.

8.155.2.1 SHARED ownership

OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS (p. 1223) 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. 1545) will be able to access modifications from all **com.rti.dds.publication.DataWriter** (p. 538) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 113) or **HISTORY** (p. 75) policy). In any case, there is no "filtering" of modifications made based on the identity of the **com.rti.dds.publication.DataWriter** (p. 538) that causes the modification.

8.155.2.2 EXCLUSIVE ownership

OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS (p. 1224) indicates that each instance of a data type can only be modified by one **com.rti.dds.publication.DataWriter** (p. 538). In other words, at any point in time, a single **com.rti.dds.publication.DataWriter** (p. 538) owns each instance and is the only one whose modifications will be visible to the **com.rti.dds.subscription.DataReader** (p. 473) objects. The owner is determined by selecting the **com.rti.dds.publication.DataWriter** (p. 538) with the highest value of the **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy.value** (p. 1226) that is currently alive, as defined by the **LIVELINESS** (p. 78) policy, and has not violated its **DEADLINE** (p. 50) contract with regards to the data instance.

Ownership can therefore change as a result of:

- a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) that owns the instance, and
- a change in the liveliness of the **com.rti.dds.publication.DataWriter** (p. 538) that owns the instance.
- a deadline with regards to the instance that is missed by the **com.rti.dds.publication.DataWriter** (p. 538) 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. 473). Each **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) objects for that **com.rti.dds.topic.Topic** (p. 1545) have a consistent picture of who owns each instance.

It is also not a requirement that the **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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. 538) 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. 538) 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. 473) objects will make the same choice of the particular **com.rti.dds.publication.DataWriter** (p. 538) 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.DataWriter** (p. 538) with higher same strength, or a new **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) as long as they affect instances whose values have not been set by the higher-strength **com.rti.dds.publication.DataWriter** (p. 538).

8.155.3 Compatibility

The value of the **com.rti.dds.infrastructure.OwnershipQosPolicyKind** (p. 1223) offered must exactly match the one requested or else they are considered incompatible.

8.155.4 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.dds.topic.example.FooDataWriter.unregister_instance and com.rti.dds.topic.example.FooDataWriter.dispose.

8.155.4.1 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. 538) entities are "capable" of writing the same instance. In this situation, the **com.rti.dds.publication.DataWriter** (p. 538) 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 **OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1224) and arbitrate themselves by means of the **com.rti.dds.infrastructure.OwnershipStrengthQosPolicy** (p. 1225). Regardless of the scheme, the desired behavior from the **com.rti.dds.subscription.DataReader** (p. 473) point of view is that **com.rti.dds.subscription.DataReader** (p. 473) normally receives data from the primary unless the "primary" writer stops writing, in which case the **com.rti.dds.subscription.DataReader** (p. 473) starts to receive data from the secondary **com.rti.dds.publication.DataWriter** (p. 538).

This approach requires some mechanism to detect that a **com.rti.dds.publication.DataWriter** (p. 538) (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.dds.topic.example.FooDataWriter.write.

Arbitrating from a **com.rti.dds.publication.DataWriter** (p. 538) to one of a higher strength is simple and the decision can be taken autonomously by the **com.rti.dds.subscription.DataReader** (p. 473). Switching ownership from a higher strength **com.rti.dds.publication.DataWriter** (p. 538) to one of a lower strength **com.rti.dds.publication.DataWriter** (p. 538) requires that the **com.rti.dds.subscription.DataReader** (p. 473) can make a determination that the stronger **com.rti.dds.publication.DataWriter** (p. 538) is "no longer writing the instance".

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. 538) simply states its offered **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) (maximum interval between updates) and the **com.rti.dds.subscription.DataReader** (p. 473) automatically monitors that the **com.rti.dds.publication.DataWriter** (p. 538) indeed updates the instance at least once per **com.rti.dds.infrastructure.DeadlineQosPolicy.period** (p. 606). If the deadline is missed, the **com.rti.dds.subscription.DataReader** (p. 473) considers the **com.rti.dds.publication.DataWriter** (p. 538) "not alive" and automatically gives ownership to the next highest-strength **com.rti.dds.publication.DataWriter** (p. 538) that *is* alive.

Case where data is not periodically updated The case where the **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) is "alive" and for the **com.rti.dds.publication.DataWriter** (p. 538) to be alive it must fulfill its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164) 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 **LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1169). The **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) process periodically

writes data or calls **com.rti.dds.domain.DomainParticipant.assert_liveliness** (p. 690). However, this scenario requires that the **com.rti.dds.subscription.DataReader** (p. 473) knows what instances are being "written" by the **com.rti.dds.publication.DataWriter** (p. 538). That is the only way that the **com.rti.dds.subscription.DataReader** (p. 473) deduces the ownership of specific instances from the fact that the **com.rti.dds.publication.DataWriter** (p. 538) is still "alive". Hence the need for the **com.rti.dds.publication.DataWriter** (p. 538) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **com.rti.dds.publication.DataWriter** (p. 538) 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. 473).

8.155.4.2 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. 604) 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.LivelinessQosPolicy** (p. 1164) in combination with register_instance/unregister_instance to detect whether the "connectivity" has been lost, and the notification is provided by means of InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

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).

8.155.4.3 Semantic Difference between unregister_instance and dispose

com.rti.dds.topic.example.FooDataWriter.dispose is semantically different from com.rti.dds.topic.example.FooDataWriter.unregister_instance. com.rti.dds.topic.example.FooDataWriter.dispose 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.dds.topic.example.FooDataWriter.unregister_instance indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **com.rti.dds.publication.DataWriter** (p. 538) 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. 1545) with **OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1224), 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. 538) 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. 1545) with **OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1224), 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.

8.155.5 Member Data Documentation

8.155.5.1 OwnershipQosPolicyKind kind

The kind of ownership.

[default] **OwnershipQosPolicyKind.SHARED_OWNERSHIP_QOS** (p. 1223)

8.156 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. 538)**, but the owner of an instance can change dynamically.*

8.156.1 Detailed Description

Kinds of ownership.

QoS:

com.rti.dds.infrastructure.OwnershipQosPolicy (p. 1216)

8.156.2 Member Data Documentation

8.156.2.1 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. 83) policy is not specified or supported.

8.156.2.2 final OwnershipQosPolicyKind EXCLUSIVE_OWNERSHIP_QOS [static]

Indicates each instance can only be owned by one **com.rti.dds.publication.DataWriter** (p. 538), but the owner of an instance can change dynamically.

The selection of the owner is controlled by the setting of the **OWNERSHIP_STRENGTH** (p. 84) policy. The owner is always set to be the highest-strength **com.rti.dds.publication.DataWriter** (p. 538) object among the ones currently active (as determined by the **LIVELINESS** (p. 78)).

8.157 OwnershipStrengthQosPolicy Class Reference

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 538) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1545) + key).

Inheritance diagram for OwnershipStrengthQosPolicy::

Public Attributes

- int **value**

  The strength value used to arbitrate among multiple writers.

8.157.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **com.rti.dds.publication.DataWriter** (p. 538) objects that attempt to modify the same instance of a data type (identified by **com.rti.dds.topic.Topic** (p. 1545) + key).

This policy only applies if the **OWNERSHIP** (p. 83) policy is of kind **OwnershipQosPolicyKind.EXCLUSIVE_OWNERSHIP_QOS** (p. 1224).

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

The value of the **OWNERSHIP_STRENGTH** (p. 84) 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. 473).

See also:

EXCLUSIVE ownership (p. 1217)

8.157.2 Member Data Documentation

8.157.2.1 int value

The strength value used to arbitrate among multiple writers.

[default] 0

[range] [0, 1 million]

8.158 ParticipantBuiltinTopicData Class Reference

Entry created when a **DomainParticipant** (p. 629) 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. 629).*

- final **PropertyQosPolicy property**

  <<**eXtension**>> (p. 270) *Name value pair properties to be stored with **domain** (p. 317) participant*

- final **ProtocolVersion_t rtps_protocol_version**

  <<**eXtension**>> (p. 270) *Version number of the RTPS wire protocol used.*

- final **VendorId_t rtps_vendor_id**

  <<**eXtension**>> (p. 270) *ID of vendor implementing the RTPS wire protocol.*

- int **dds_builtin_endpoints**

  <<**eXtension**>> (p. 270) *Bitmap of **builtin** (p. 319) endpoints supported by the participant.*

- final **LocatorSeq default_unicast_locators**

  <<**eXtension**>> (p. 270) *Unicast locators used when individual entities do not specify unicast locators.*

- final **ProductVersion_t product_version**

  <<**eXtension**>> (p. 270) *This is a vendor specific parameter. It gives the current version for rti-dds.*

- final **EntityNameQosPolicy participant_name**

  <<**eXtension**>> (p. 270) *The participant name and role name.*

8.158.1 Detailed Description

Entry created when a **DomainParticipant** (p. 629) object is discovered.

Data associated with the built-in **topic** (p. 350) **ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME** (p. 1232). It contains QoS policies and additional information that apply to the remote **com.rti.dds.domain.DomainParticipant** (p. 629).

See also:

ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 1232)
builtin.ParticipantBuiltinTopicDataDataReader (p. 1230)

8.158.2 Member Data Documentation

8.158.2.1 final BuiltinTopicKey_t key

DCPS key to distinguish entries.

8.158.2.2 final UserDataQosPolicy user_data

Policy of the corresponding **DomainParticipant** (p. 629).

8.158.2.3 final PropertyQosPolicy property

<<***eXtension***>> *(p. 270)* Name value pair properties to be stored with **domain** (p. 317) participant

8.158.2.4 final ProtocolVersion_t rtps_protocol_version

<<***eXtension***>> *(p. 270)* Version number of the RTPS wire protocol used.

8.158.2.5 final VendorId_t rtps_vendor_id

<<***eXtension***>> *(p. 270)* ID of vendor implementing the RTPS wire protocol.

8.158.2.6 int dds_builtin_endpoints

<<***eXtension***>> *(p. 270)* Bitmap of **builtin** (p. 319) endpoints supported by the participant.

Each bit indicates a **builtin** (p. 319) endpoint that may be available on the participant for use in discovery.

8.158.2.7 final LocatorSeq default_unicast_locators

<<*eXtension*>> *(p. 270)* Unicast locators used when individual entities do not specify unicast locators.

8.158.2.8 final ProductVersion_t product_version

<<*eXtension*>> *(p. 270)* This is a vendor specific parameter. It gives the current version for rti-dds.

8.158.2.9 final EntityNameQosPolicy participant_name

<<*eXtension*>> *(p. 270)* The participant name and role name.

This parameter contains the name and the role name of the discovered participant.

8.159 ParticipantBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

Inherits DataReaderImpl.

8.159.1 Detailed Description

Instantiates `DataReader` < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

com.rti.dds.subscription.DataReader (p. 473) of **topic** (p. 350) **ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME** (p. 1232) used for accessing **builtin.ParticipantBuiltinTopicData** (p. 1227) of the remote **com.rti.dds.domain.DomainParticipant** (p. 629).

Instantiates:

<<***generic***>> *(p. 271)* com.rti.dds.topic.example.FooDataReader

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)
ParticipantBuiltinTopicDataTypeSupport.PARTICIPANT_TOPIC_NAME (p. 1232)

8.160 ParticipantBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

Inherits AbstractBuiltinTopicDataSeq.

8.160.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)

8.161 ParticipantBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

Inheritance diagram for ParticipantBuiltinTopicDataTypeSupport::

Static Public Attributes

- static final String **PARTICIPANT_TOPIC_NAME** = DDS_PARTICIPANT_TOPIC_NAME()

  Participant ***topic*** *(p. 350) name.*

8.161.1 Detailed Description

Instantiates `TypeSupport` < **builtin.ParticipantBuiltinTopicData** (p. 1227) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)

8.161.2 Member Data Documentation

8.161.2.1 final String PARTICIPANT_TOPIC_NAME = DDS_PARTICIPANT_TOPIC_NAME() `[static]`

Participant **topic** (p. 350) name.

Topic name of **builtin.ParticipantBuiltinTopicDataDataReader** (p. 1230)

See also:

builtin.ParticipantBuiltinTopicData (p. 1227)
builtin.ParticipantBuiltinTopicDataDataReader (p. 1230)

8.162 PartitionQosPolicy Class Reference

Set of strings that introduces a logical partition among the topics visible by a **com.rti.dds.publication.Publisher** (p. 1277) and a **com.rti.dds.subscription.Subscriber** (p. 1478).

Inheritance diagram for PartitionQosPolicy::

Public Attributes

- final **StringSeq name**

  A list of partition names.

8.162.1 Detailed Description

Set of strings that introduces a logical partition among the topics visible by a **com.rti.dds.publication.Publisher** (p. 1277) and a **com.rti.dds.subscription.Subscriber** (p. 1478).

This QoS policy is used to set string identifiers that are used for matching DataReaders and DataWriters for the same Topic.

A **com.rti.dds.publication.DataWriter** (p. 538) within a **com.rti.dds.publication.Publisher** (p. 1277) only communicates with a **com.rti.dds.subscription.DataReader** (p. 473) in a **com.rti.dds.subscription.Subscriber** (p. 1478) if (in addition to matching the **com.rti.dds.topic.Topic** (p. 1545) and having compatible QoS) the **com.rti.dds.publication.Publisher** (p. 1277) and **com.rti.dds.subscription.Subscriber** (p. 1478) have a common partition name string.

Entity:

com.rti.dds.publication.Publisher (p. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **YES** (p. 98)

8.162.2 Usage

This policy allows the introduction of a logical partition concept inside the 'physical' partition induced by a ***domain*** *(p. 317)*.

Usually DataReaders and DataWriters are matched only by their **topic** (p. 350) (so that data are only sent by DataWriters to DataReaders for the same **topic** (p. 350)). The Partition QoS policy allows you to add one or more strings, "partitions", to a Publisher and/or Subscriber. If partitions are added, then a DataWriter and DataReader for the same **topic** (p. 350) are only considered matched if their Publishers and Subscribers have partitions in common (intersecting partitions).

Since the set of partitions for a publisher or subscriber can be dynamically changed, the Partition QoS policy is useful to control which DataWriters can send data to which DataReaders and vice versa – even if all of the DataWriters and DataReaders are for the same **topic** (p. 350). This facility is useful for creating temporary separation groups among entities that would otherwise be connected to and exchange data each other.

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 a Publisher intersect with the partitions or wildcard patterns of a Subscriber, their DataWriters and DataReaders of the same **topic** (p. 350) are able to match; otherwise they are not.

These partition name patterns are regular expressions as defined by the POSIX fnmatch API (1003.2-1992 section B.6). Either **com.rti.dds.publication.Publisher** (p. 1277) or **com.rti.dds.subscription.Subscriber** (p. 1478) 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 both at publisher as well as subscriber side, RTI Connext will not try to match two regular expressions (between publishers and subscribers).

Each publisher and subscriber must belong to at least one logical partition. A regular expression is not considered to be a logical partition. If a publisher or subscriber has not specify 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. 912) 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** (p. 317) it does not belong to.
- Second, a **com.rti.dds.infrastructure.Entity** (p. 912) can only belong to one **domain** (p. 317) whereas a **com.rti.dds.infrastructure.Entity** (p. 912) can be in multiple partitions.
- Finally, as far as RTI Connext is concerned, each unique data instance is identified by the tuple **(DomainID, com.rti.dds.topic.Topic (p. 1545), key)**. Therefore two **com.rti.dds.infrastructure.Entity** (p. 912) 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.

8.162.3 Member Data Documentation

8.162.3.1 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 **RETCODE_INCONSISTENT_POLICY** (p. 1367) when setting a **com.rti.dds.publication.Publisher** (p. 1277)'s or **com.rti.dds.subscription.Subscriber** (p. 1478)'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. 1233) is specified on a **domain** (p. 317) basis in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.max_partitions** (p. 754). 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. 1233) is specified on a **domain** (p. 317) basis in **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.max_partition_cumulative_characters** (p. 754).

[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

8.163 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*** *(p. 343) end. Controls whether ordered access is supported within the scope* `access_scope`.

8.163.1 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. 1277), **com.rti.dds.subscription.Subscriber** (p. 1478)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.163.2 Usage

A **com.rti.dds.subscription.DataReader** (p. 473) will usually receive data in the order that it was sent by a **com.rti.dds.publication.DataWriter** (p. 538), and the data is presented to the **com.rti.dds.subscription.DataReader** (p. 473) as soon as the application receives the next expected value. However, sometimes, you may want a set of data for the same **topic** (p. 350) to be presented to the **com.rti.dds.subscription.DataReader** (p. 473) only after *all* of the elements of the set have been received. Or you may want the data to be presented in a different order than that in which it was received. 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 allows you to specify different *scopes* of presentation: within a **topic** (p. 350), across instances of a single **topic** (p. 350), and even across multiple topics used by different writers of a publisher. 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. 1296) and **com.rti.dds.publication.Publisher.end_coherent_changes** (p. 1297).

- `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` controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to **PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS** (p. 1243) (the default), the use of **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1296) and **com.rti.dds.publication.Publisher.end_coherent_changes** (p. 1297) has no effect on how the subscriber can access the data, because with the scope limited to each instance, changes to separate instances are considered independent and thus cannot be grouped into a coherent set.

- If `access_scope` is set to **PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS** (p. 1243), then coherent changes (indicated by their enclosure within calls to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1296) and **com.rti.dds.publication.Publisher.end_coherent_changes** (p. 1297)) will be made available as such to each remote **com.rti.dds.subscription.DataReader** (p. 473) independently. That is, changes made to instances within each individual **com.rti.dds.publication.DataWriter** (p. 538) will be available as coherent with respect to other changes to instances in that same **com.rti.dds.publication.DataWriter** (p. 538), but will not be grouped with changes made to instances belonging to a different **com.rti.dds.publication.DataWriter** (p. 538).

- If access_scope is set to **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243), then coherent changes made to instances through a **com.rti.dds.publication.DataWriter** (p. 538) attached to a common **com.rti.dds.publication.Publisher** (p. 1277) are made available as a unit to remote subscribers. *(RTI does not currently support this access scope.)*

If `ordered_access` is set, then the `access_scope` controls the maximum extent for which order will be preserved by RTI Connext.

- If `access_scope` is set to **PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS** (p. 1243) (the lowest level), then changes to each instance are considered unordered relative to changes to any other instance. That means that changes (creations, deletions, modifications) made to two instances are not necessarily seen in the order they occur. This is the case even if it is the same application thread making the changes using the same **com.rti.dds.publication.DataWriter** (p. 538).

- If `access_scope` is set to **PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS** (p. 1243),

changes (creations, deletions, modifications) made by a single **com.rti.dds.publication.DataWriter** (p. 538) are made available to subscribers in the same order they occur. Changes made to instances though different **com.rti.dds.publication.DataWriter** (p. 538) entities are not necessarily seen in the order they occur. This is the case, even if the changes are made by a single application thread using **com.rti.dds.publication.DataWriter** (p. 538) objects attached to the same **com.rti.dds.publication.Publisher** (p. 1277).

- Finally, if access_scope is set to **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243), changes made to instances via **com.rti.dds.publication.DataWriter** (p. 538) entities attached to the same **com.rti.dds.publication.Publisher** (p. 1277) object are made available to subscribers on the same order they occur.

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. 1478) **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. 473) objects.

For **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243) the subscribing application must use the APIs **com.rti.dds.subscription.Subscriber.begin_access** (p. 1499), **com.rti.dds.subscription.Subscriber.end_access** (p. 1500) and **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1491) to access the changes in the proper order.

8.163.3 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 **PresentationQosPolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS** (p. 1243). For the purposes of this inequality, the values of `access_scope` are considered ordered such that **PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS** (p. 1243) < **PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS** (p. 1243) < **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243).

- requested `coherent_access` is false, or else both offered and requested `coherent_access` are true.
- requested `ordered_access` is false, or else both offered and requested `ordered_access` are true.

8.163.4 Member Data Documentation

8.163.4.1 PresentationQosPolicyAccessScopeKind access_scope

Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] **PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS** (p. 1243)

8.163.4.2 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.

[default] false

8.163.4.3 boolean ordered_access

Specifies support for *ordered* access to the samples received at the **subscription** (p. 343) 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] false

8.164 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 to all instances within the same **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)), but not across instances in different **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)).*

- static final **PresentationQosPolicyAccessScopeKind GROUP_PRESENTATION_QOS**

  *Scope spans to all instances belonging to **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) entities within the same **com.rti.dds.publication.Publisher** (p. 1277) (or **com.rti.dds.subscription.Subscriber** (p. 1478)).*

- static final **PresentationQosPolicyAccessScopeKind HIGHEST_OFFERED_PRESENTATION_QOS**

  *This value only applies to a **com.rti.dds.subscription.Subscriber** (p. 1478). The **com.rti.dds.subscription.Subscriber** (p. 1478) will use the access scope specified by each remote **com.rti.dds.publication.Publisher** (p. 1277).*

8.164.1 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. 1237)

8.164.2 Member Data Documentation

8.164.2.1 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.

8.164.2.2 final PresentationQosPolicyAccessScopeKind TOPIC_PRESENTATION_QOS [static]

Scope spans to all instances within the same **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)), but not across instances in different **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)).

8.164.2.3 final PresentationQosPolicyAccessScopeKind GROUP_PRESENTATION_QOS [static]

Scope spans to all instances belonging to **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) entities within the same **com.rti.dds.publication.Publisher** (p. 1277) (or **com.rti.dds.subscription.Subscriber** (p. 1478)).

8.164.2.4 final PresentationQosPolicyAccessScopeKind HIGHEST_OFFERED_PRESENTATION_QOS [static]

This value only applies to a **com.rti.dds.subscription.Subscriber** (p. 1478). The **com.rti.dds.subscription.Subscriber** (p. 1478) will use the access scope specified by each remote **com.rti.dds.publication.Publisher** (p. 1277).

8.165 PRIVATE_MEMBER Class Reference

Constant used to indicate that a value type member is private.

Static Public Attributes

- ^ static final short **VALUE**

8.165.1 Detailed Description

Constant used to indicate that a value type member is private.

See also:

PUBLIC_MEMBER (p. 1263)

8.165.2 Member Data Documentation

8.165.2.1 final short VALUE [static]

Constant value.

8.166 ProductVersion_t Class Reference

<<*eXtension*>> *(p. 270)* Type used to represent the current version of RTI Connext.

Inherits Struct.

Public Member Functions

- **ProductVersion_t** ()

  Constructor.

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.

8.166.1 Detailed Description

<<*eXtension*>> *(p. 270)* Type used to represent the current version of RTI Connext.

8.166.2 Constructor & Destructor Documentation

8.166.2.1 ProductVersion_t ()

Constructor.

8.166.3 Member Data Documentation

8.166.3.1 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.

8.166.3.2 char major

Major product version.

8.166.3.3 char minor

Minor product version.

8.166.3.4 char release

Release letter for product version.

8.166.3.5 char revision

Revision number of product.

8.167 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. 227) 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. 227).*

- boolean **ignore_resource_profile**

  Ignores the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.4d/xml.

8.167.1 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. 708)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **Changeable** (p. 98)

8.167.2 Member Data Documentation

8.167.2.1 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).

8.167.2.2 final StringSeq url_profile

Sequence of **URL groups** (p. 227) 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).

8.167.2.3 boolean ignore_user_profile

Ignores the file USER_QOS_PROFILES.xml in the current working directory.

When this field is set to true, the QoS profiles contained in the file USER_QOS_PROFILES.xml in the current working directory will be ignored.

[**default**] false

8.167.2.4 boolean ignore_environment_profile

Ignores the value of the **NDDS_QOS_PROFILES environment variable** (p. 227).

When this field is set to true, the value of the environment variable NDDS_QOS_PROFILES will be ignored.

[**default**] false

8.167.2.5 boolean ignore_resource_profile

Ignores the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.4d/xml.

When this field is set to true, the QoS profiles contained in the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.5f/xml will be ignored.

[**default**] false

8.168 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.

Public Attributes

- String **name**

  Property name.

- String **value**

  Property value.

- boolean **propagate**

  Indicates if the property must be propagated on discovery.

8.168.1 Detailed Description

Properties are name/value pairs objects.

8.168.2 Constructor & Destructor Documentation

8.168.2.1 Property_t ()

Constructor.

8.168.2.2 Property_t (Property_t *src*)

Constructor.

Parameters:

src <<*in*>> *(*p. *271)* Property used to initialized the new property.

8.168.2.3 Property_t (String *name*, String *value*, boolean *propagate*)

Constructor.

Parameters:

name <<*in*>> *(*p. *271)* Property name.

value <<*in*>> *(*p. *271)* Property value.

propagate <<*in*>> *(*p. *271)* Parameter used to indicates whether or not the property must be propagated.

8.168.3 Member Data Documentation

8.168.3.1 String name

Property name.

8.168.3.2 String value

Property value.

8.168.3.3 boolean propagate

Indicates if the property must be propagated on discovery.

8.169 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.

8.169.1 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. 629)
com.rti.dds.subscription.DataReader (p. 473)
com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A;
Changeable (p. 98) = **YES** (p. 98)

See also:

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.169.2 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. 473), **com.rti.dds.publication.DataWriter** (p. 538), or **com.rti.dds.domain.DomainParticipant** (p. 629). This is similar to the **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680), 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** (p. 350)).

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 221)
- Durable Reader State, see **Configuring Durable Reader State** (p. 221)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1655), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1667), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1441)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 213)
- **Clock Selection** (p. 141)

In addition, you may add your own name/value pairs to the Property QoS policy of an **Entity** (p. 912). Via this QoS policy, you can direct RTI Connext to propagate these name/value pairs with the discovery information for the **Entity** (p. 912). Applications that discover the **Entity** (p. 912) can then access the user-specific name/value pairs in the discovery information of the remote **Entity** (p. 912). This allows you to add meta-information about an **Entity** (p. 912) for application-specific use, for example, authentication/authorization certificates (which can also be done using the **com.rti.dds.infrastructure.UserDataQosPolicy** (p. 1680) or **com.rti.dds.infrastructure.GroupDataQosPolicy** (p. 1064)).

8.169.2.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports (such as RTI Secure WAN Transport)
- 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. 912) 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 **com.rti.dds.infrastructure.PropertyQosPolicyHelper** (p. 1255) class on the **PROPERTY** (p. 88) page.

8.169.3 Member Data Documentation

8.169.3.1 final PropertySeq value

Initial value:

```
new PropertySeq()
```

Sequence of properties.

[**default**] An empty list.

8.170 PropertyQosPolicyHelper Class Reference

Policy Helpers which 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.

8.170.1 Detailed Description

Policy Helpers which facilitate management of the properties in the input policy.

8.170.2 Member Function Documentation

8.170.2.1 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. 271)* Input policy.

Returns:

Number of properties.

8.170.2.2 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. 271)* Input policy.

name *<<in>>* *(p. 271)* Property name.

value *<<in>>* *(p. 271)* Property value.

propagate *<<in>>* *(p. 271)* Indicates if the property will be propagated on discovery.

Returns:

One of the **Standard Return Codes** (p. 104) or **RETCODE_OUT_OF_RESOURCES** (p. 1370).

8.170.2.3 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 **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

Precondition:

policy, name and value cannot be null.
The property is not in the policy.

Parameters:

policy *<<in>>* *(p. 271)* Input policy.

name *<<in>>* *(p. 271)* Property name.

value *<<in>>* *(p. 271)* Property value.

propagate *<<in>>* *(p. 271)* Indicates if the property will be propagated on discovery.

Returns:

One of the **Standard Return Codes** (p. 104) or **RETCODE_OUT_OF_RESOURCES** (p. 1370) or **RETCODE_PRECONDITION_NOT_MET** (p. 1371)

8.170.2.4 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. 271)* Input policy.

name *<<in>>* *(p. 271)* Property name.

Returns:

On success, the function returns the first property with the given name. Otherwise, the function returns NULL.

8.170.2.5 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 **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

Precondition:

policy and name cannot be null.
The property is in the policy.

Parameters:

policy *<<in>> (p. 271)* Input policy.

name *<<in>> (p. 271)* Property name.

Returns:

One of the **Standard Return Codes** (p. 104) or **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

8.170.2.6 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 **RETCODE_OUT_OF_RESOURCES** (p. 1370).

Precondition:

policy, properties and name_prefix cannot be null.

Parameters:

policy *<<in>> (p. 271)* Input policy.

properties *<<inout>> (p. 271)* A **com.rti.dds.infrastructure.PropertySeq** (p. 1259) object where the set or list of properties will be returned.

name_prefix Name prefix.

Returns:

One of the **Standard Return Codes** (p. 104) or **RETCODE_OUT_OF_RESOURCES** (p. 1370).

8.171 PropertySeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.Property_t** (p. 1250) >.

Inherits ArraySequence.

8.171.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.Property_t** (p. 1250) >.

See also:

com.rti.dds.infrastructure.Property_t (p. 1250)

8.172 ProtocolVersion_t Class Reference

<<eXtension>> (p. 270) 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 1.2.

8.172.1 Detailed Description

<<*eXtension*>> *(p. 270)* Type used to represent the version of the RTPS protocol.

8.172.2 Constructor & Destructor Documentation

8.172.2.1 ProtocolVersion_t ()

Constructor.

8.172.3 Member Data Documentation

8.172.3.1 final ProtocolVersion_t PROTOCOLVERSION_1_0 [static]

Initial value:

```
new ProtocolVersion_t((byte)1, (byte)0)
```

The protocol version 1.0.

8.172.3.2 final ProtocolVersion_t PROTOCOLVERSION_1_1 [static]

Initial value:

```
new ProtocolVersion_t((byte)1, (byte)1)
```

The protocol version 1.1.

8.172.3.3 final ProtocolVersion_t PROTOCOLVERSION_1_2 [static]

Initial value:

```
new ProtocolVersion_t((byte)1, (byte)2)
```

The protocol version 1.2.

8.172.3.4 final ProtocolVersion_t PROTOCOLVERSION_2_0 [static]

Initial value:

```
new ProtocolVersion_t((byte)2, (byte)0)
```

The protocol version 2.0.

8.172.3.5 final ProtocolVersion_t PROTOCOLVERSION_2_1 [static]

Initial value:

```
new ProtocolVersion_t((byte)2, (byte)1)
```

The protocol version 2.1.

8.172.3.6 final ProtocolVersion_t PROTOCOLVERSION [static]

Initial value:

```
new ProtocolVersion_t((byte)2, (byte)1)
```

The most recent protocol version. Currently 1.2.

8.172.3.7 byte major

Major protocol version number.

8.172.3.8 byte minor

Minor protocol version number.

8.173 PUBLIC_MEMBER Class Reference

Constant used to indicate that a value type member is public.

Static Public Attributes

- ^ static final short **VALUE**

8.173.1 Detailed Description

Constant used to indicate that a value type member is public.

See also:

PRIVATE_MEMBER (p. 1244)

8.173.2 Member Data Documentation

8.173.2.1 final short VALUE [static]

Constant value.

8.174 PublicationBuiltinTopicData Class Reference

Entry created when a **com.rti.dds.publication.DataWriter** (p. 538) is discovered in association with its **Publisher** (p. 1277).

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. 538) belongs.*

- String **topic_name**

  *Name of the related **com.rti.dds.topic.Topic** (p. 1545).*

- String **type_name**

  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).*

- final **DurabilityQosPolicy durability**

  *durability policy of the corresponding **DataWriter** (p. 538)*

- final **DurabilityServiceQosPolicy durability_service**

  *durability_service policy of the corresponding **DataWriter** (p. 538)*

- final **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataWriter** (p. 538).*

- final **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataWriter** (p. 538).*

- final **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataWriter** (p. 538).*

- final **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataWriter** (p. 538).*

- final **LifespanQosPolicy lifespan**

  *Policy of the corresponding **DataWriter** (p. 538).*

^ final **UserDataQosPolicy user_data**

*Policy of the corresponding **DataWriter** (p. 538).*

^ final **OwnershipQosPolicy ownership**

*Policy of the corresponding **DataWriter** (p. 538).*

^ final **OwnershipStrengthQosPolicy ownership_strength**

*Policy of the corresponding **DataWriter** (p. 538).*

^ final **DestinationOrderQosPolicy destination_order**

*Policy of the corresponding **DataWriter** (p. 538).*

^ final **PresentationQosPolicy presentation**

*Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.*

^ final **PartitionQosPolicy partition**

*Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.*

^ final **TopicDataQosPolicy topic_data**

Policy of the related Topic.

^ final **GroupDataQosPolicy group_data**

*Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.*

^ **TypeCode type_code**

<<**eXtension**>> (p. 270) *Type code information of the corresponding Topic*

^ final **BuiltinTopicKey_t publisher_key**

<<**eXtension**>> (p. 270) *DCPS key of the publisher to which the **DataWriter** (p. 538) belongs*

^ final **PropertyQosPolicy property**

<<**eXtension**>> (p. 270) *Properties of the corresponding **DataWriter** (p. 538).*

^ final **LocatorSeq unicast_locators**

<<**eXtension**>> (p. 270) *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. 270) *Virtual GUID associated to the* ***DataWriter*** *(p. 538).*

- final **ProtocolVersion_t rtps_protocol_version**

  <<**eXtension**>> (p. 270) *Version number of the RTPS wire protocol used.*

- final **VendorId_t rtps_vendor_id**

  <<**eXtension**>> (p. 270) *ID of vendor implementing the RTPS wire protocol.*

- final **ProductVersion_t product_version**

  <<**eXtension**>> (p. 270) *This is a vendor specific parameter. It gives the current version for rti-dds.*

- final **LocatorFilterQosPolicy locator_filter**

  <<**eXtension**>> (p. 270) *Policy of the corresponding* ***DataWriter*** *(p. 538)*

- boolean **disable_positive_acks**

  <<**eXtension**>> (p. 270) *This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.*

- final **EntityNameQosPolicy publication_name**

  <<**eXtension**>> (p. 270) *The* ***publication*** *(p. 338) name and role name.*

8.174.1 Detailed Description

Entry created when a **com.rti.dds.publication.DataWriter** (p. 538) is discovered in association with its **Publisher** (p. 1277).

Data associated with the built-in **topic** (p. 350) **PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME** (p. 1273). It contains QoS policies and additional information that apply to the remote **com.rti.dds.publication.DataWriter** (p. 538) the related **com.rti.dds.publication.Publisher** (p. 1277).

See also:

PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME (p. 1273)
builtin.PublicationBuiltinTopicDataDataReader (p. 1271)

8.174.2 Member Data Documentation

8.174.2.1 final BuiltinTopicKey_t key

DCPS key to distinguish entries.

8.174.2.2 final BuiltinTopicKey_t participant_key

DCPS key of the participant to which the **DataWriter** (p. 538) belongs.

8.174.2.3 String topic_name

Name of the related **com.rti.dds.topic.Topic** (p. 1545).
The length of this string is limited to 255 characters.

8.174.2.4 String type_name

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).
The length of this string is limited to 255 characters.

8.174.2.5 final DurabilityQosPolicy durability

durability policy of the corresponding **DataWriter** (p. 538)

8.174.2.6 final DurabilityServiceQosPolicy durability_service

durability_service policy of the corresponding **DataWriter** (p. 538)

8.174.2.7 final DeadlineQosPolicy deadline

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.8 final LatencyBudgetQosPolicy latency_budget

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.9 final LivelinessQosPolicy liveliness

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.10 final ReliabilityQosPolicy reliability

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.11 final LifespanQosPolicy lifespan

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.12 final UserDataQosPolicy user_data

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.13 final OwnershipQosPolicy ownership

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.14 final OwnershipStrengthQosPolicy ownership_strength

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.15 final DestinationOrderQosPolicy destination_order

Policy of the corresponding **DataWriter** (p. 538).

8.174.2.16 final PresentationQosPolicy presentation

Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.

8.174.2.17 final PartitionQosPolicy partition

Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.

8.174.2.18 final TopicDataQosPolicy topic_data

Policy of the related Topic.

8.174.2.19 final GroupDataQosPolicy group_data

Policy of the **Publisher** (p. 1277) to which the **DataWriter** (p. 538) belongs.

8.174.2.20 TypeCode type_code

<<*eXtension*>> *(p. 270)* Type code information of the corresponding Topic

8.174.2.21 final BuiltinTopicKey_t publisher_key

<<*eXtension*>> *(p. 270)* DCPS key of the publisher to which the **DataWriter** (p. 538) belongs

8.174.2.22 final PropertyQosPolicy property

<<*eXtension*>> *(p. 270)* Properties of the corresponding **DataWriter** (p. 538).

8.174.2.23 final LocatorSeq unicast_locators

<<*eXtension*>> *(p. 270)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.174.2.24 final GUID_t virtual_guid

<<*eXtension*>> *(p. 270)* Virtual GUID associated to the **DataWriter** (p. 538).

See also:

com.rti.dds.infrastructure.GUID_t (p. 1069)

8.174.2.25 final ProtocolVersion_t rtps_protocol_version

<<*eXtension*>> *(p. 270)* Version number of the RTPS wire protocol used.

8.174.2.26 final VendorId_t rtps_vendor_id

<<*eXtension*>> *(p. 270)* ID of vendor implementing the RTPS wire protocol.

8.174.2.27 final ProductVersion_t product_version

<<*eXtension*>> *(p. 270)* This is a vendor specific parameter. It gives the current version for rti-dds.

8.174.2.28 final LocatorFilterQosPolicy locator_filter

<<*eXtension*>> *(p. 270)* Policy of the corresponding **DataWriter** (p. 538)

Related to **com.rti.dds.infrastructure.MultiChannelQosPolicy** (p. 1205).

8.174.2.29 boolean disable_positive_acks

<<*eXtension*>> *(p. 270)* This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.

8.174.2.30 final EntityNameQosPolicy publication_name

<<*eXtension*>> *(p. 270)* The **publication** (p. 338) name and role name.

This member contains the name and the role name of the discovered **publication** (p. 338).

8.175 PublicationBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

Inherits DataReaderImpl.

8.175.1 Detailed Description

Instantiates `DataReader` < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

com.rti.dds.subscription.DataReader (p. 473) of **topic** (p. 350) **PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME** (p. 1273) used for accessing **builtin.PublicationBuiltinTopicData** (p. 1264) of the remote **com.rti.dds.publication.DataWriter** (p. 538) and the associated **com.rti.dds.publication.Publisher** (p. 1277).

Instantiates:

<<***generic***>> *(p. 271)* com.rti.dds.topic.example.FooDataReader

See also:

builtin.PublicationBuiltinTopicData (p. 1264)
PublicationBuiltinTopicDataTypeSupport.PUBLICATION_TOPIC_NAME (p. 1273)

8.176 PublicationBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

Inherits AbstractBuiltinTopicDataSeq.

8.176.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

Instantiates:

<<***generic***>> *(*p. *271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

builtin.PublicationBuiltinTopicData (p. 1264)

8.177 PublicationBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

Inheritance diagram for PublicationBuiltinTopicDataTypeSupport::

Static Public Attributes

- ^ static final String **PUBLICATION_TOPIC_NAME** = DDS_PUBLICATION_TOPIC_NAME()

  Publication ***topic*** *(p. 350) name.*

8.177.1 Detailed Description

Instantiates `TypeSupport` < **builtin.PublicationBuiltinTopicData** (p. 1264) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

See also:

builtin.PublicationBuiltinTopicData (p. 1264)

8.177.2 Member Data Documentation

8.177.2.1 final String PUBLICATION_TOPIC_NAME = DDS_PUBLICATION_TOPIC_NAME() [static]

Publication **topic** (p. 350) name.

Topic name of **builtin.PublicationBuiltinTopicDataDataReader** (p. 1271)

See also:

builtin.PublicationBuiltinTopicData (p. 1264)
builtin.PublicationBuiltinTopicDataDataReader (p. 1271)

8.178 PublicationMatchedStatus Class Reference

StatusKind.PUBLICATION_MATCHED_STATUS.

Inherits Status.

Public Attributes

- int **total_count**

  *The total cumulative number of times the concerned **com.rti.dds.publication.DataWriter** (p. 538) discovered a "match" with a **com.rti.dds.subscription.DataReader** (p. 473).*

- 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 readers with which the **com.rti.dds.publication.DataWriter** (p. 538) is matched.*

- int **current_count_peak**

  <<**eXtension**>> (p. 270) *The highest value that current_count has reached until now.*

- 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_subscription_handle**

  *A handle to the last **com.rti.dds.subscription.DataReader** (p. 473) that caused the the **com.rti.dds.publication.DataWriter** (p. 538)'s status to change.*

8.178.1 Detailed Description

StatusKind.PUBLICATION_MATCHED_STATUS.

A "match" happens when the **com.rti.dds.publication.DataWriter** (p. 538) finds a **com.rti.dds.subscription.DataReader** (p. 473) for the same **com.rti.dds.topic.Topic** (p. 1545) and common partition

with a requested QoS that is compatible with that offered by the **com.rti.dds.publication.DataWriter** (p. 538).

This status is also changed (and the listener, if any, called) when a match is ended. A local **com.rti.dds.publication.DataWriter** (p. 538) will become "unmatched" from a remote **com.rti.dds.subscription.DataReader** (p. 473) when that **com.rti.dds.subscription.DataReader** (p. 473) goes away for any reason.

8.178.2 Member Data Documentation

8.178.2.1 int total_count

The total cumulative number of times the concerned **com.rti.dds.publication.DataWriter** (p. 538) discovered a "match" with a **com.rti.dds.subscription.DataReader** (p. 473).

This number increases whenever a new match is discovered. It does not change when an existing match goes away.

8.178.2.2 int total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

8.178.2.3 int current_count

The current number of readers with which the **com.rti.dds.publication.DataWriter** (p. 538) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away.

8.178.2.4 int current_count_peak

<<***eXtension***>> *(p. 270)* The highest value that current_count has reached until now.

8.178.2.5 int current_count_change

The change in current_count since the last time the listener was called or the status was read.

8.178.2.6 final InstanceHandle_t last_subscription_handle

A handle to the last **com.rti.dds.subscription.DataReader** (p. 473) that caused the the **com.rti.dds.publication.DataWriter** (p. 538)'s status to change.

8.179 Publisher Interface Reference

<<***interface***>> *(p. 271)* 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. 588) values into the provided **com.rti.dds.publication.DataWriterQos** (p. 588) instance.*

- void **set_default_datawriter_qos** (**DataWriterQos** qos)

  *Sets the default **com.rti.dds.publication.DataWriterQos** (p. 588) values for this publisher.*

- void **set_default_datawriter_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default **com.rti.dds.publication.DataWriterQos** (p. 588) 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. 538) that will be attached and belong to the **com.rti.dds.publication.Publisher** (p. 1277).*

- **DataWriter create_datawriter_with_profile** (**Topic** topic, String library_name, String profile_name, **DataWriterListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a **com.rti.dds.publication.DataWriter** (p. 538) object using the **com.rti.dds.publication.DataWriterQos** (p. 588) associated with the input XML QoS profile.*

- void **delete_datawriter** (**DataWriter** a_datawriter)

  *Deletes a **com.rti.dds.publication.DataWriter** (p. 538) that belongs to the **com.rti.dds.publication.Publisher** (p. 1277).*

- **DataWriter lookup_datawriter** (String topic_name)

  *Retrieves the **com.rti.dds.publication.DataWriter** (p. 538) for a specific **com.rti.dds.topic.Topic** (p. 1545).*

- void **set_qos** (**PublisherQos** qos)

  Sets the publisher QoS.

- void **set_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *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. 270) *Gets the default XML library associated with a* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- void **set_default_library** (String library_name)

  <<**eXtension**>> (p. 270) *Sets the default XML library for a* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- String **get_default_profile** ()

  <<**eXtension**>> (p. 270) *Gets the default XML profile associated with a* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- void **set_default_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Sets the default XML profile for a* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- String **get_default_profile_library** ()

  <<**eXtension**>> (p. 270) *Gets the library where the default XML QoS profile is contained for a* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- 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. 538) objects belonging to the* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- 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. 1294).*

- void **begin_coherent_changes** ()

  Indicates that the application will begin a coherent set of modifications using ***com.rti.dds.publication.DataWriter*** *(p. 538) objects attached to the* ***com.rti.dds.publication.Publisher*** *(p. 1277).*

- void **end_coherent_changes** ()

  Terminates the coherent set initiated by the matching call to ***com.rti.dds.publication.Publisher.begin_coherent_changes*** *(p. 1296).*

- void **copy_from_topic_qos** (**DataWriterQos** a_datawriter_qos, **TopicQos** a_topic_qos)

  Copies the policies in the ***com.rti.dds.topic.TopicQos*** *(p. 1566) to the corresponding policies in the* ***com.rti.dds.publication.DataWriterQos*** *(p. 588).*

- void **get_all_datawriters** (DataWriterSeq writers)

  Retrieve all the DataWriters created from this ***Publisher*** *(p. 1277).*

- **DomainParticipant get_participant** ()

  Returns the ***com.rti.dds.domain.DomainParticipant*** *(p. 629) to which the* ***com.rti.dds.publication.Publisher*** *(p. 1277) 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. 1277).*

- void **wait_for_acknowledgments** (**Duration_t** max_wait)

  Blocks the calling thread until all data written by reliable ***com.rti.dds.publication.DataWriter*** *(p. 538) entities is acknowledged, or until timeout expires.*

- void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

  <<**eXtension**>> (p. 270) *Blocks the calling thread until asynchronous sending is complete.*

Static Public Attributes

- static final **DataWriterQos DATAWRITER_QOS_DEFAULT**

Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 538) with default QoS.*

- static final **DataWriterQos DATAWRITER_QOS_USE_TOPIC_QOS** = new **DataWriterQos**()

  Special value for creating ***com.rti.dds.publication.DataWriter*** *(p. 538) with a combination of the default* ***com.rti.dds.publication.DataWriterQos*** *(p. 588) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1566).*

8.179.1 Detailed Description

<<***interface***>> *(p. 271)* A publisher is the object responsible for the actual dissemination of publications.

QoS:

com.rti.dds.publication.PublisherQos (p. 1303)

Listener:

com.rti.dds.publication.PublisherListener (p. 1302)

A publisher acts on the behalf of one or several **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) 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. 1277) and the **com.rti.dds.publication.DataWriter** (p. 538).

The following operations may be called even if the **com.rti.dds.publication.Publisher** (p. 1277) is not enabled. Other operations will fail with the value RETCODE_NOT_ENABLED if called on a disabled **com.rti.dds.publication.Publisher** (p. 1277):

- The base-class operations **com.rti.dds.publication.Publisher.set_qos** (p. 1289), **com.rti.dds.publication.Publisher.set_qos_with_profile** (p. 1290), **com.rti.dds.publication.Publisher.get_qos** (p. 1291), **com.rti.dds.publication.Publisher.set_listener** (p. 1294), **com.rti.dds.publication.Publisher.get_listener** (p. 1294), **com.rti.dds.infrastructure.Entity.enable** (p. 915), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 917), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917)

ˆ **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284), **com.rti.dds.publication.Publisher.create_datawriter_with_profile** (p. 1286), **com.rti.dds.publication.Publisher.delete_datawriter** (p. 1287), **com.rti.dds.publication.Publisher.delete_contained_entities** (p. 1299), **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282), **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1283), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1281), **com.rti.dds.publication.Publisher.wait_for_acknowledgments** (p. 1299), **com.rti.dds.publication.Publisher.set_default_library** (p. 1291), **com.rti.dds.publication.Publisher.set_default_profile** (p. 1292),

See also:

Operations Allowed in Listener Callbacks (p. 1156)

8.179.2 Member Function Documentation

8.179.2.1 void get_default_datawriter_qos (DataWriterQos *qos*)

Copies the default **com.rti.dds.publication.DataWriterQos** (p. 588) values into the provided **com.rti.dds.publication.DataWriterQos** (p. 588) instance.

The retreived `qos` will match the set of values specified on the last successful call to **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282) or **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1283), or else, if the call was never made, the default values from is owning **com.rti.dds.domain.DomainParticipant** (p. 629).

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. 1277) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282).

Parameters:

qos *<<inout>>* *(p. 271)* **com.rti.dds.publication.DataWriterQos** (p. 588) to be filled-up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

Publisher.DATAWRITER_QOS_DEFAULT (p. 177)
com.rti.dds.publication.Publisher.create_datawriter (p. 1284)

8.179.2.2 void set_default_datawriter_qos (DataWriterQos *qos*)

Sets the default **com.rti.dds.publication.DataWriterQos** (p. 588) values for this publisher.

This call causes the default values inherited from the owning **com.rti.dds.domain.DomainParticipant** (p. 629) to be overridden.

This default value will be used for newly created **com.rti.dds.publication.DataWriter** (p. 538) if **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) is specified as the `qos` parameter when **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.publication.Publisher** (p. 1277) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1281) or calling **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284) with **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) as the `qos` parameter.

Parameters:

qos <<***in***>> *(p. 271)* Default qos to be set. The special value Subscriber.DATAREADER_QOS_DEFAULT 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. 1282) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

8.179.2.3 void set_default_datawriter_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> *(p. 270)* Set the default **com.rti.dds.publication.DataWriterQos** (p. 588) 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. 538) if **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) is specified as the `qos` parameter when **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284) is called.

Precondition:

The **com.rti.dds.publication.DataWriterQos** (p. 588) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.publication.Publisher** (p. 1277) while another thread may be simultaneously calling **com.rti.dds.publication.Publisher.set_default_datawriter_qos** (p. 1282), **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1281) or calling **com.rti.dds.publication.Publisher.create_datawriter** (p. 1284) with **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) as the `qos` parameter.

Parameters:

library_name *<<in>>* *(p. 271)* 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. 1291)).

profile_name *<<in>>* *(p. 271)* 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. 1292)).

If the input profile cannot be found, the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

Publisher.DATAWRITER_QOS_DEFAULT (p. 177)
com.rti.dds.publication.Publisher.create_datawriter_with_profile (p. 1286)

8.179.2.4 DataWriter create_datawriter (Topic *topic*, DataWriterQos *qos*, DataWriterListener *listener*, int *mask*)

Creates a **com.rti.dds.publication.DataWriter** (p. 538) that will be attached and belong to the **com.rti.dds.publication.Publisher** (p. 1277).

For each application-defined type, `Foo`, there is an implied, auto-generated class `com.rti.dds.topic.example.FooDataWriter` that extends **com.rti.dds.publication.DataWriter** (p. 538) and contains the operations to write data of type `Foo`.

Note that a common application pattern to construct the QoS for the **com.rti.dds.publication.DataWriter** (p. 538) is to:

- Retrieve the QoS policies on the associated **com.rti.dds.topic.Topic** (p. 1545) by means of the **com.rti.dds.topic.Topic.get_qos** (p. 1548) operation.
- Retrieve the default **com.rti.dds.publication.DataWriter** (p. 538) qos by means of the **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1281) operation.
- Combine those two QoS policies (for **example** (p. 342), using **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1297)) and selectively modify policies as desired.

When a **com.rti.dds.publication.DataWriter** (p. 538) is created, only those transports already registered are available to the **com.rti.dds.publication.DataWriter** (p. 538). See **Built-in Transport Plugins** (p. 216) for details on when a **builtin** (p. 341) transport is registered.

Precondition:

If publisher is enabled, **topic** (p. 350) must have been enabled. Otherwise, this operation will fail and no **com.rti.dds.publication.DataWriter** (p. 538) will be created.
The given **com.rti.dds.topic.Topic** (p. 1545) 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 **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) 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. 1282).

Parameters:

topic *(p. **350**)* *<<in>>* *(p. 271)* The **com.rti.dds.topic.Topic** (p. 1545) that the **com.rti.dds.publication.DataWriter** (p. 538) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 271)* QoS to be used for creating the new **com.rti.dds.publication.DataWriter** (p. 538). The special value **Publisher.DATAWRITER_QOS_DEFAULT** (p. 177) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) should be created with the default **com.rti.dds.publication.DataWriterQos** (p. 588) set in the **com.rti.dds.publication.Publisher** (p. 1277). The special value **Publisher.DATAWRITER_QOS_USE_TOPIC_QOS** (p. 178) can be used to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) should be created with the combination of the default **com.rti.dds.publication.DataWriterQos** (p. 588) set on the **com.rti.dds.publication.Publisher** (p. 1277) and the **com.rti.dds.topic.TopicQos** (p. 1566) of the **com.rti.dds.topic.Topic** (p. 1545). Cannot be NULL.

listener *<<in>>* *(p. 271)* The listener of the **com.rti.dds.publication.DataWriter** (p. 538).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.publication.DataWriter** (p. 538) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataWriter
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS
Publisher.DATAWRITER_QOS_DEFAULT (p. 177)
Publisher.DATAWRITER_QOS_USE_TOPIC_QOS (p. 178)
com.rti.dds.publication.Publisher.create_datawriter_with_profile (p. 1286)
com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1281)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1297)
com.rti.dds.publication.DataWriter.set_listener (p. 545)

8.179.2.5 DataWriter create_datawriter_with_profile (Topic *topic*, String *library_name*, String *profile_name*, DataWriterListener *listener*, int *mask*)

*<<**eXtension**>>* *(p. 270)* Creates a **com.rti.dds.publication.DataWriter** (p. 538) object using the **com.rti.dds.publication.DataWriterQos** (p. 588) associated with the input XML QoS profile.

The **com.rti.dds.publication.DataWriter** (p. 538) will be attached and belong to the **com.rti.dds.publication.Publisher** (p. 1277).

For each application-defined type, `Foo`, there is an implied, auto-generated class `com.rti.dds.topic.example.FooDataWriter` that extends **com.rti.dds.publication.DataWriter** (p. 538) and contains the operations to write data of type `Foo`.

When a **com.rti.dds.publication.DataWriter** (p. 538) is created, only those transports already registered are available to the **com.rti.dds.publication.DataWriter** (p. 538). See **Built-in Transport Plugins** (p. 216) for details on when a **builtin** (p. 341) transport is registered.

Precondition:

If publisher is enabled, **topic** (p. 350) must have been enabled. Otherwise, this operation will fail and no **com.rti.dds.publication.DataWriter** (p. 538) will be created.
The given **com.rti.dds.topic.Topic** (p. 1545) 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 *(p. **350**)* *<<in>>* *(p. 271)* The **com.rti.dds.topic.Topic** (p. 1545) that the **com.rti.dds.publication.DataWriter** (p. 538) will be associated with. Cannot be NULL.

library_name *<<in>>* *(p. 271)* 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. 1291)).

profile_name *<<in>>* *(p. 271)* 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. 1292)).

listener *<<in>>* *(p. 271)* The listener of the **com.rti.dds.publication.DataWriter** (p. 538).

mask *<<in>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.publication.DataWriter** (p. 538) of a derived class specific to the data type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataWriter
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.publication.DataWriterQos (p. 588) for rules on consistency among QoS
com.rti.dds.publication.Publisher.create_datawriter (p. 1284)
com.rti.dds.publication.Publisher.get_default_datawriter_qos (p. 1281)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.publication.Publisher.copy_from_topic_qos (p. 1297)
com.rti.dds.publication.DataWriter.set_listener (p. 545)

8.179.2.6 void delete_datawriter (DataWriter *a_datawriter*)

Deletes a **com.rti.dds.publication.DataWriter** (p. 538) that belongs to the **com.rti.dds.publication.Publisher** (p. 1277).

The deletion of the **com.rti.dds.publication.DataWriter** (p. 538) will automatically unregister all instances. Depending on the settings of the **WRITER_DATA_LIFECYCLE** (p. 134) QosPolicy, the deletion of the **com.rti.dds.publication.DataWriter** (p. 538) may also dispose all instances.

8.179.3 Special Instructions if Using 'Timestamp' APIs and BY_SOURCE_TIMESTAMP Destination Ordering:

If the DataWriter's **com.rti.dds.infrastructure.DestinationOrderQosPolicy.kind** (p. 609) is DestinationOrderQosPolicyKind.BY_SOURCE_TIMESTAMP_-DESTINATIONORDER_QOS, calls to **delete_datawriter()** (p. 1287) may fail if your application has previously used the 'with timestamp' APIs (write_w_timestamp(), register_instance_w_timestamp(), unregister_instance_-w_timestamp(), or dispose_w_timestamp()) with a timestamp larger (later) than the time at which **delete_datawriter()** (p. 1287) is called. To prevent **delete_datawriter()** (p. 1287) from failing in this situation, either:

- Change the **WRITER_DATA_LIFECYCLE** (p. 134) QosPolicy so that RTI Connext will not autodispose unregistered instances (set **com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy.autodispose_-unregistered_instances** (p. 1723) to false.) or
- Explicitly call unregister_instance_w_timestamp() for all instances modified with the *_w_timestamp() APIs before calling **delete_datawriter()** (p. 1287).

Precondition:

If the **com.rti.dds.publication.DataWriter** (p. 538) does not belong to the **com.rti.dds.publication.Publisher** (p. 1277), the operation will fail with RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.publication.DataWriter** (p. 538) will not be called after this method completes successfully.

Parameters:

a_datawriter *<<in>>* *(p. 271)* The **com.rti.dds.publication.DataWriter** (p. 538) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_-PRECONDITION_NOT_MET.

8.179.3.1 DataWriter lookup_datawriter (String *topic_name*)

Retrieves the **com.rti.dds.publication.DataWriter** (p. 538) for a specific **com.rti.dds.topic.Topic** (p. 1545).

This returned **com.rti.dds.publication.DataWriter** (p. 538) is either enabled or disabled.

Parameters:

topic_name *<<in>>* *(p. 271)* Name of the **com.rti.dds.topic.Topic** (p. 1545) associated with the **com.rti.dds.publication.DataWriter** (p. 538) that is to be looked up. Cannot be NULL.

Returns:

A **com.rti.dds.publication.DataWriter** (p. 538) that belongs to the **com.rti.dds.publication.Publisher** (p. 1277) attached to the **com.rti.dds.topic.Topic** (p. 1545) with `topic_name`. If no such **com.rti.dds.publication.DataWriter** (p. 538) exists, this operation returns NULL.

If more than one **com.rti.dds.publication.DataWriter** (p. 538) is attached to the **com.rti.dds.publication.Publisher** (p. 1277) 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. 538) in one thread while another thread is simultaneously creating or destroying that **com.rti.dds.publication.DataWriter** (p. 538).

8.179.3.2 void set_qos (PublisherQos *qos*)

Sets the publisher QoS.

This operation modifies the QoS of the **com.rti.dds.publication.Publisher** (p. 1277).

The **com.rti.dds.publication.PublisherQos.group_data** (p. 1304), **com.rti.dds.publication.PublisherQos.partition** (p. 1304) and **com.rti.dds.publication.PublisherQos.entity_factory** (p. 1304) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 271)* **com.rti.dds.publication.PublisherQos** (p. 1303) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.publication.Publisher** (p. 1277) is enabled. The special value **DomainParticipant.PUBLISHER_QOS_DEFAULT** (p. 149) can be used to

indicate that the QoS of the **com.rti.dds.publication.Publisher** (p. 1277) should be changed to match the current default **com.rti.dds.publication.PublisherQos** (p. 1303) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.publication.PublisherQos (p. 1303) for rules on consistency among QoS
set_qos (abstract) (p. 913)
Operations Allowed in Listener Callbacks (p. 1156)

8.179.3.3 void set_qos_with_profile (String *library_name*, String *profile_name*)

<<***eXtension***>> *(p. 270)* 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. 1277).

The **com.rti.dds.publication.PublisherQos.group_data** (p. 1304), **com.rti.dds.publication.PublisherQos.partition** (p. 1304) and **com.rti.dds.publication.PublisherQos.entity_factory** (p. 1304) can be changed. The other policies are immutable.

Parameters:

library_name <<***in***>> *(p. 271)* 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. 1291)).

profile_name <<***in***>> *(p. 271)* 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. 1292)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.publication.PublisherQos (p. 1303) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1156)

8.179.3.4 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. 271)* **com.rti.dds.publication.PublisherQos** (p. 1303) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.179.3.5 String get_default_library ()

<<eXtension>> *(p. 270)* Gets the default XML library associated with a **com.rti.dds.publication.Publisher** (p. 1277).

Returns:

The default library or null if the default library was not set.

See also:

com.rti.dds.publication.Publisher.set_default_library (p. 1291)

8.179.3.6 void set_default_library (String *library_name*)

<<eXtension>> *(p. 270)* Sets the default XML library for a **com.rti.dds.publication.Publisher** (p. 1277).

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'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. 1277) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 629) (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 679)).

Parameters:

library_name *<<**in**>>* *(p. 271)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.publication.Publisher.get_default_library (p. 1291)

8.179.3.7 String get_default_profile ()

*<<**eXtension**>>* *(p. 270)* Gets the default XML profile associated with a **com.rti.dds.publication.Publisher** (p. 1277).

Returns:

The default profile or null if the default profile was not set.

See also:

com.rti.dds.publication.Publisher.set_default_profile (p. 1292)

8.179.3.8 void set_default_profile (String *library_name*, String *profile_name*)

*<<**eXtension**>>* *(p. 270)* Sets the default XML profile for a **com.rti.dds.publication.Publisher** (p. 1277).

This method specifies the profile that will be used as the default the next time a default **Publisher** (p. 1277) profile is needed during a call to one of this Publisher's operations. When calling a **com.rti.dds.publication.Publisher** (p. 1277) 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. 1277) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 629) (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 680)).

This method does not set the default QoS for **com.rti.dds.publication.DataWriter** (p. 538) objects created by the **com.rti.dds.publication.Publisher** (p. 1277); for this functionality, use **com.rti.dds.publication.Publisher.set_default_datawriter_qos_with_profile** (p. 1283) (you may pass in NULL after having called **set_default_profile()** (p. 1292)).

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. 645).

Parameters:

library_name *<<in>> (p. 271)* The library name containing the profile.

profile_name *<<in>> (p. 271)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.publication.Publisher.get_default_profile (p. 1292)
com.rti.dds.publication.Publisher.get_default_profile_library (p. 1293)

8.179.3.9 String get_default_profile_library ()

<<eXtension>> (p. 270) Gets the library where the default XML QoS profile is contained for a **com.rti.dds.publication.Publisher** (p. 1277).

The default profile library is automatically set when **com.rti.dds.publication.Publisher.set_default_profile** (p. 1292) is called.

This library can be different than the **com.rti.dds.publication.Publisher** (p. 1277) default library (see **com.rti.dds.publication.Publisher.get_default_library** (p. 1291)).

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. 1292)

8.179.3.10 void set_listener (PublisherListener *l*, int *mask*)

Sets the publisher listener.

Parameters:

l *<<in>>* *(p. 271)* **com.rti.dds.publication.PublisherListener** (p. 1302) to set to.

mask *<<in>>* *(p. 271)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.publication.PublisherListener** (p. 1302).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.179.3.11 PublisherListener get_listener ()

Get the publisher listener.

Returns:

com.rti.dds.publication.PublisherListener (p. 1302) of the **com.rti.dds.publication.Publisher** (p. 1277).

See also:

get_listener (abstract) (p. 915)

8.179.3.12 void suspend_publications ()

Indicates to RTI Connext that the application is about to make multiple modifications using **com.rti.dds.publication.DataWriter** (p. 538) objects belonging to the **com.rti.dds.publication.Publisher** (p. 1277).

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. 1295) indicating that the set of modifications has completed.

If the **com.rti.dds.publication.Publisher** (p. 1277) is deleted before **com.rti.dds.publication.Publisher.resume_publications** (p. 1295) 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. 942), PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 538) instances allow the user even finer control of traffic shaping and sample coalescing.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.publication.FlowController (p. 942)
com.rti.dds.publication.FlowController.trigger_flow (p. 945)
FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME (p. 184)
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)

8.179.3.13 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. 1294).

This is a **hint** to RTI Connext that can be used for **example** (p. 342), to batch all the modifications made since the **com.rti.dds.publication.Publisher.suspend_publications** (p. 1294).

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. 942), PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS **com.rti.dds.publication.DataWriter** (p. 538) 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. 1295) must match a previous call to

com.rti.dds.publication.Publisher.suspend_publications (p. 1294). Otherwise the operation will fail with RETCODE_PRECONDITION_NOT_MET.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

See also:

com.rti.dds.publication.FlowController (p. 942)
com.rti.dds.publication.FlowController.trigger_flow (p. 945)
FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME (p. 184)
com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)

8.179.3.14 void begin_coherent_changes ()

Indicates that the application will begin a coherent set of modifications using **com.rti.dds.publication.DataWriter** (p. 538) objects attached to the **com.rti.dds.publication.Publisher** (p. 1277).

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** (p. 342), the set of partitions used by the **com.rti.dds.publication.Publisher** (p. 1277) or one of its **com.rti.dds.subscription.Subscriber** (p. 1478) s may change, a late-joining **com.rti.dds.subscription.DataReader** (p. 473) 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. 1297).

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** (p. 342), 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).

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

8.179.3.15 void end_coherent_changes ()

Terminates the coherent set initiated by the matching call to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1296).

Precondition:

If there is no matching call to **com.rti.dds.publication.Publisher.begin_coherent_changes** (p. 1296) the operation will fail with RETCODE_PRECONDITION_NOT_MET.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

8.179.3.16 void copy_from_topic_qos (DataWriterQos *a_datawriter_qos*, TopicQos *a_topic_qos*)

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1566) to the corresponding policies in the **com.rti.dds.publication.DataWriterQos** (p. 588).

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1566) to the corresponding policies in the **com.rti.dds.publication.DataWriterQos** (p. 588) (replacing values in the **com.rti.dds.publication.DataWriterQos** (p. 588), if present).

This is a "convenience" operation most useful in combination with the operations **com.rti.dds.publication.Publisher.get_default_datawriter_qos** (p. 1281) and **com.rti.dds.topic.Topic.get_qos** (p. 1548). The operation **com.rti.dds.publication.Publisher.copy_from_topic_qos** (p. 1297) can be used to merge the **com.rti.dds.publication.DataWriter** (p. 538) default QoS policies with the corresponding ones on the **com.rti.dds.topic.Topic** (p. 1545). The resulting QoS can then be used to create a new **com.rti.dds.publication.DataWriter** (p. 538), or set its QoS.

This operation does not check the resulting **com.rti.dds.publication.DataWriterQos** (p. 588) for consistency. This

is because the 'merged' **com.rti.dds.publication.DataWriterQos** (p. 588) 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. 538).

Parameters:

a_datawriter_qos *<<inout>>* *(p. 271)* **com.rti.dds.publication.DataWriterQos** (p. 588) to be filled-up. Cannot be NULL.

a_topic_qos *<<in>>* *(p. 271)* **com.rti.dds.topic.TopicQos** (p. 1566) to be merged with **com.rti.dds.publication.DataWriterQos** (p. 588). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.179.3.17 void get_all_datawriters (DataWriterSeq *writers*)

Retrieve all the DataWriters created from this **Publisher** (p. 1277).

Parameters:

writers *<<inout>>* *(p. 271)* Sequence where the DataWriters will be added

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.179.3.18 DomainParticipant get_participant ()

Returns the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.publication.Publisher** (p. 1277) belongs.

Returns:

the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.publication.Publisher** (p. 1277) belongs.

8.179.3.19 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. 1277).

Deletes all contained **com.rti.dds.publication.DataWriter** (p. 538) objects. Once **com.rti.dds.publication.Publisher.delete_contained_entities** (p. 1299) completes successfully, the application may delete the **com.rti.dds.publication.Publisher** (p. 1277), knowing that it has no contained **com.rti.dds.publication.DataWriter** (p. 538) objects.

The operation will fail with RETCODE_PRECONDITION_NOT_MET if any of the contained entities is in a state where it cannot be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET.

8.179.3.20 void wait_for_acknowledgments (Duration_t *max_wait*)

Blocks the calling thread until all data written by reliable **com.rti.dds.publication.DataWriter** (p. 538) entities is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **com.rti.dds.publication.DataWriter** (p. 538) entities is acknowledged by all matched reliable **com.rti.dds.subscription.DataReader** (p. 473) entities, or else 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; a return value of TIMEOUT indicates that max_wait elapsed before all the data was acknowledged.

If none of the **com.rti.dds.publication.DataWriter** (p. 538) instances have **com.rti.dds.infrastructure.ReliabilityQosPolicy** (p. 1336) kind set to RELIABLE, the operation will complete successfully.

Parameters:

max_wait *<<in>>* *(p. 271)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 776) .

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NOT_ENABLED, RETCODE_TIMEOUT

8.179.3.21 void wait_for_asynchronous_publishing (Duration_t *max_wait*)

<<***eXtension***>> *(*p. 270*)* 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. 538) entities is sent and acknowledged (if reliable) by all matched **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) is complete in addition to what **com.rti.dds.publication.Publisher.wait_for_acknowledgments** (p. 1299) provides.

If none of the **com.rti.dds.publication.DataWriter** (p. 538) instances have **com.rti.dds.infrastructure.PublishModeQosPolicy.kind** (p. 1310) set to PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS, the operation will complete immediately, with RETCODE_OK.

Parameters:

max_wait <<***in***>> *(*p. 271*)* Specifies maximum time to wait for acknowledgements **com.rti.dds.infrastructure.Duration_t** (p. 776).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_NOT_ENABLED, RETCODE_TIMEOUT

8.180 PublisherAdapter Class Reference

<<***eXtension***>> *(p. 270)* 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::

8.180.1 Detailed Description

<<***eXtension***>> *(p. 270)* 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.

8.181 PublisherListener Interface Reference

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for **com.rti.dds.publication.Publisher** (p. 1277) status.

Inheritance diagram for PublisherListener::

8.181.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for **com.rti.dds.publication.Publisher** (p. 1277) status.

Entity:

com.rti.dds.publication.Publisher (p. 1277)

Status:

StatusKind.LIVELINESS_LOST_STATUS, **com.rti.dds.publication.LivelinessLostStatus** (p. 1162);
StatusKind.OFFERED_DEADLINE_MISSED_STATUS, **com.rti.dds.publication.OfferedDeadlineMissedStatus** (p. 1212);
StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS, **com.rti.dds.publication.OfferedIncompatibleQosStatus** (p. 1214);
StatusKind.PUBLICATION_MATCHED_STATUS, **com.rti.dds.publication.PublicationMatchedStatus** (p. 1274);
StatusKind.RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1342);
StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS, **com.rti.dds.publication.ReliableWriterCacheChangedStatus** (p. 1345)

See also:

com.rti.dds.infrastructure.Listener (p. 1154)
Status Kinds (p. 106)
Operations Allowed in Listener Callbacks (p. 1156)

8.182 PublisherQos Class Reference

QoS policies supported by a **com.rti.dds.publication.Publisher** (p. 1277) entity.

Inheritance diagram for PublisherQos::

Public Attributes

- final **PresentationQosPolicy presentation**

  Presentation policy, ***PRESENTATION*** *(p. 86).*

- final **PartitionQosPolicy partition**

  Partition policy, ***PARTITION*** *(p. 85).*

- final **GroupDataQosPolicy group_data**

  Group data policy, ***GROUP_DATA*** *(p. 73).*

- final **EntityFactoryQosPolicy entity_factory**

  Entity factory policy, ***ENTITY_FACTORY*** *(p. 69).*

- final **AsynchronousPublisherQosPolicy asynchronous_publisher**

  <<**eXtension**>> (p. 270) *Asynchronous publishing settings for the* ***com.rti.dds.publication.Publisher*** *(p. 1277) and all entities that are created by it.*

- final **ExclusiveAreaQosPolicy exclusive_area**

  <<**eXtension**>> (p. 270) *Exclusive area for the* ***com.rti.dds.publication.Publisher*** *(p. 1277) and all entities that are created by it.*

8.182.1 Detailed Description

QoS policies supported by a **com.rti.dds.publication.Publisher** (p. 1277) entity.

You must set certain members in a consistent manner:

length of com.rti.dds.publication.PublisherQos.group_data.value <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .publisher_group_data_max_length

length of com.rti.dds.publication.PublisherQos.partition.name <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .max_partitions

combined number of characters (including terminating 0) in com.rti.dds.publication.PublisherQos.partition.name <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .max_partition_cumulative_characters

If any of the above are not true, **com.rti.dds.publication.Publisher.set_qos** (p. 1289) and **com.rti.dds.publication.Publisher.set_qos_with_profile** (p. 1290) will fail with RETCODE_INCONSISTENT_POLICY and **com.rti.dds.domain.DomainParticipant.create_publisher** (p. 656) will return NULL.

8.182.2 Member Data Documentation

8.182.2.1 final PresentationQosPolicy presentation

Presentation policy, **PRESENTATION** (p. 86).

8.182.2.2 final PartitionQosPolicy partition

Partition policy, **PARTITION** (p. 85).

8.182.2.3 final GroupDataQosPolicy group_data

Group data policy, **GROUP_DATA** (p. 73).

8.182.2.4 final EntityFactoryQosPolicy entity_factory

Entity factory policy, **ENTITY_FACTORY** (p. 69).

8.182.2.5 final AsynchronousPublisherQosPolicy asynchronous_publisher

<<***eXtension***>> *(p. 270)* Asynchronous publishing settings for the **com.rti.dds.publication.Publisher** (p. 1277) and all entities that are created by it.

8.182.2.6 final ExclusiveAreaQosPolicy exclusive_area

<<*eXtension*>> *(p. 270)* Exclusive area for the **com.rti.dds.publication.Publisher** (p. 1277) and all entities that are created by it.

8.183 PublisherSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.publication.Publisher** (p. 1277) > .

Inherits AbstractNativeSequence.

Public Member Functions

- **PublisherSeq** (Collection publishers)
- int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.183.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.publication.Publisher** (p. 1277) > .

See also:

com.rti.dds.util.Sequence (p. 1432)

8.183.2 Constructor & Destructor Documentation

8.183.2.1 PublisherSeq (Collection *publishers*)

Exceptions:

NullPointerException if the given collection is null

8.183.3 Member Function Documentation

8.183.3.1 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. 383), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.184 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.

8.184.1 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. 538) determines whether data is written synchronously in the context of the user thread when calling com.rti.dds.topic.example.FooDataWriter.write or asynchronously in the context of a separate thread internal to the middleware.

Each **com.rti.dds.publication.Publisher** (p. 1277) spawns a single asynchronous publishing thread (**com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy.thread** (p. 389)) to serve all its asynchronous **com.rti.dds.publication.DataWriter** (p. 538) instances.

See also:

com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)
com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)
com.rti.dds.publication.FlowController (p. 942)

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.184.2 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. 942), 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. 538). 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.dds.topic.example.FooDataWriter.write 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 Transport.Property_t.gather_send_buffer_count_max (each sample requires at least 2-4 gather-send buffers). Performance can be improved by increasing Transport.Property_t.gather_send_buffer_count_max. 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. 942)). The number of samples that will be queued is determined by the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071). When using **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075), only the most recent **com.rti.dds.infrastructure.HistoryQosPolicy.depth**

(p. 1074) 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.

8.184.3 Member Data Documentation

8.184.3.1 PublishModeQosPolicyKind kind

Publishing mode.

[default] PublishModeQosPolicyKind.SYNCHRONOUS_PUBLISH_MODE_QOS (p. 1311)

8.184.3.2 String flow_controller_name

Name of the associated flow controller.

NULL value or zero-length string refers to FlowController.DEFAULT_FLOW_CONTROLLER_NAME.

See also:

com.rti.dds.domain.DomainParticipant.create_flowcontroller (p. 654)
FlowController.DEFAULT_FLOW_CONTROLLER_NAME
FlowController.FIXED_RATE_FLOW_CONTROLLER_NAME
FlowController.ON_DEMAND_FLOW_CONTROLLER_NAME

[default] FlowController.DEFAULT_FLOW_CONTROLLER_NAME

8.185 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.

8.185.1 Detailed Description

Kinds of publishing mode.

QoS:

com.rti.dds.infrastructure.PublishModeQosPolicy (p. 1308)

8.185.2 Member Data Documentation

8.185.2.1 final PublishModeQosPolicyKind SYNCHRONOUS_PUBLISH_MODE_QOS [static]

Indicates to send data synchronously.

If **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) is true, data is sent immediately in the context of com.rti.dds.topic.example.FooDataWriter.write.

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. 573)

[**default**] for **com.rti.dds.publication.DataWriter** (p. 538)

8.185.2.2 final PublishModeQosPolicyKind ASYNCHRONOUS_PUBLISH_MODE_QOS [static]

Indicates to send data asynchronously.

Configures the **com.rti.dds.publication.DataWriter** (p. 538) to delegate the task of data transmission to a separate publishing thread. The com.rti.dds.topic.example.FooDataWriter.write call does not send the data, but instead schedules the data to be sent later by its associated **com.rti.dds.publication.Publisher** (p. 1277).

Each **com.rti.dds.publication.Publisher** (p. 1277) uses its dedicated publishing thread (**com.rti.dds.publication.PublisherQos.asynchronous_publisher** (p. 1304)) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated **com.rti.dds.publication.FlowController** (p. 942) determines when the publishing thread is allowed to send the data.

com.rti.dds.publication.DataWriter.wait_for_asynchronous_publishing (p. 553) and **com.rti.dds.publication.Publisher.wait_for_asynchronous_publishing** (p. 1300) enable you to determine when the data has actually been sent.

Note: **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.push_on_write** (p. 573) must be TRUE for Asynchronous DataWriters. Otherwise, samples will never be sent.

See also:

com.rti.dds.publication.FlowController (p. 942)
com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)
com.rti.dds.publication.DataWriter.wait_for_asynchronous_publishing (p. 553)
com.rti.dds.publication.Publisher.wait_for_asynchronous_publishing (p. 1300)
Transport.Property_t.gather_send_buffer_count_max

8.186 Qos Class Reference

An abstract base class for all QoS types.

Inheritance diagram for Qos::

Public Member Functions

- ^ final boolean **equals** (Object other)

8.186.1 Detailed Description

An abstract base class for all QoS types.

8.186.2 Member Function Documentation

8.186.2.1 final boolean equals (Object *other*)

Parameters:

other <<***in***>> *(*p. *271**)* 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

8.187 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.

8.187.1 Detailed Description

The base class for all QoS policies.

8.187.2 Member Data Documentation

8.187.2.1 final QosPolicyId_t id

The ID of this QoS policy.

This attribute is provided for more efficient comparisons of policy types that comparing strings.

8.187.2.2 final String policy_name

The name of this QoS policy.

8.188 QosPolicyCount Class Reference

Type to hold a counter for a **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1318).

Inherits Struct.

Public Member Functions

- **QosPolicyCount** (**QosPolicyCount** src)

  Copy constructor.

Public Attributes

- **QosPolicyId_t policy_id**

  The `QosPolicy` (p. 1314) *ID.*

- int **count**

  a counter

8.188.1 Detailed Description

Type to hold a counter for a **com.rti.dds.infrastructure.QosPolicyId_t** (p. 1318).

8.188.2 Constructor & Destructor Documentation

8.188.2.1 QosPolicyCount (QosPolicyCount *src*)

Copy constructor.

Parameters:

src **QosPolicyCount** (p. 1315) to copy from.

Exceptions:

NullPointerException if the source object is null.

8.188.3 Member Data Documentation

8.188.3.1 QosPolicyId_t policy_id

The QosPolicy (p. 1314) ID.

8.188.3.2 int count

a counter

8.189 QosPolicyCountSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.infrastructure.QosPolicyCount** (p. 1315) >.

Inherits ArraySequence.

8.189.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.infrastructure.QosPolicyCount** (p. 1315) >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.QosPolicyCount (p. 1315)

8.190 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. 1680*).*

- static final **QosPolicyId_t DURABILITY_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.DurabilityQosPolicy*** *(*p. 765*).*

- static final **QosPolicyId_t PRESENTATION_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.PresentationQosPolicy*** *(*p. 1237*).*

- static final **QosPolicyId_t DEADLINE_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.DeadlineQosPolicy*** *(*p. 604*).*

- static final **QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.LatencyBudgetQosPolicy*** *(*p. 1148*).*

- static final **QosPolicyId_t OWNERSHIP_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.OwnershipQosPolicy*** *(*p. 1216*).*

- static final **QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.OwnershipStrengthQosPolicy*** *(*p. 1225*).*

- static final **QosPolicyId_t LIVELINESS_QOS_POLICY_ID**

  Identifier for ***com.rti.dds.infrastructure.LivelinessQosPolicy*** *(*p. 1164*).*

ˆ static final **QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.TimeBasedFilterQosPolicy*** *(*p. 1541*).*

ˆ static final **QosPolicyId_t PARTITION_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.PartitionQosPolicy*** *(*p. 1233*).*

ˆ static final **QosPolicyId_t RELIABILITY_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.ReliabilityQosPolicy*** *(*p. 1336*).*

ˆ static final **QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.DestinationOrderQosPolicy*** *(*p. 607*).*

ˆ static final **QosPolicyId_t HISTORY_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.HistoryQosPolicy*** *(*p. 1071*).*

ˆ static final **QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.ResourceLimitsQosPolicy*** *(*p. 1356*).*

ˆ static final **QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.EntityFactoryQosPolicy*** *(*p. 919*).*

ˆ static final **QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.WriterDataLifecycleQosPolicy*** *(*p. 1722*).*

ˆ static final **QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.ReaderDataLifecycleQosPolicy*** *(*p. 1328*).*

ˆ static final **QosPolicyId_t TOPICDATA_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.TopicDataQosPolicy*** *(*p. 1559*).*

^ static final **QosPolicyId_t GROUPDATA_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.GroupDataQosPolicy*** *(p. 1064).*

^ static final **QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.TransportPriorityQosPolicy*** *(p. 1598).*

^ static final **QosPolicyId_t LIFESPAN_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.LifespanQosPolicy*** *(p. 1152).*

^ static final **QosPolicyId_t DURABILITY_SERVICE_QOS_POLICY_ID**

Identifier for ***com.rti.dds.infrastructure.DurabilityServiceQosPolicy*** *(p. 773).*

^ static final **QosPolicyId_t WIREPROTOCOL_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.WireProtocolQosPolicy*** *(p. 1709)*

^ static final **QosPolicyId_t DISCOVERY_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DiscoveryQosPolicy*** *(p. 624)*

^ static final **QosPolicyId_t DATAREADERRESOURCELIMITS_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy*** *(p. 524)*

^ static final **QosPolicyId_t DATA_WRITER_RESOURCE_LIMITS_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy*** *(p. 598)*

^ static final **QosPolicyId_t DATAREADERPROTOCOL_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataReaderProtocolQosPolicy*** *(p. 504)*

^ static final **QosPolicyId_t DATAWRITERPROTOCOL_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DataWriterProtocolQosPolicy*** *(p. 571)*

ˆ static final **QosPolicyId_t DOMAINPARTICIPANTRESOURCE-LIMITS_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy*** *(p. 741)*

ˆ static final **QosPolicyId_t EVENT_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.EventQosPolicy*** *(p. 930)*

ˆ static final **QosPolicyId_t DATABASE_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DatabaseQosPolicy*** *(p. 468)*

ˆ static final **QosPolicyId_t RECEIVERPOOL_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.ReceiverPoolQosPolicy*** *(p. 1331)*

ˆ static final **QosPolicyId_t DISCOVERYCONFIG_QOS_POLICY_-ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.DiscoveryConfigQosPolicy*** *(p. 615)*

ˆ static final **QosPolicyId_t EXCLUSIVEAREA_QOS_POLICY_-ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.ExclusiveAreaQosPolicy*** *(p. 933)*

ˆ static final **QosPolicyId_t USEROBJECT_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* *com.rti.dds.infrastructure.UserObjectQosPolicy*

ˆ static final **QosPolicyId_t SYSTEMRESOURCELIMITS_QOS_-POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.SystemResourceLimitsQosPolicy*** *(p. 1524)*

ˆ static final **QosPolicyId_t TRANSPORTSELECTION_QOS_-POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TransportSelectionQosPolicy*** *(p. 1600)*

^ static final **QosPolicyId_t TRANSPORTUNICAST_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TransportUnicastQosPolicy*** *(p. 1605)*

^ static final **QosPolicyId_t TRANSPORTMULTICAST_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TransportMulticastQosPolicy*** *(p. 1590)*

^ static final **QosPolicyId_t TRANSPORTBUILTIN_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TransportBuiltinQosPolicy*** *(p. 1580)*

^ static final **QosPolicyId_t PUBLISHMODE_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.PublishModeQosPolicy*** *(p. 1308)*

^ static final **QosPolicyId_t ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy*** *(p. 387)*

^ static final **QosPolicyId_t TYPESUPPORT_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** *(p. 1652)*

^ static final **QosPolicyId_t ENTITYNAME_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.TypeSupportQosPolicy*** *(p. 1652)*

^ static final **QosPolicyId_t BATCH_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.BatchQosPolicy*** *(p. 401)*

^ static final **QosPolicyId_t PROFILE_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.ProfileQosPolicy*** *(p. 1247)*

^ static final **QosPolicyId_t LOCATORFILTER_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.LocatorFilterQosPolicy*** *(p. 1181)*

ˆ static final **QosPolicyId_t MULTICHANNEL_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.MultiChannelQosPolicy*** *(p. 1205)*

ˆ static final **QosPolicyId_t AVAILABILITY_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.AvailabilityQosPolicy*** *(p. 392)*

ˆ static final **QosPolicyId_t LOGGING_QOS_POLICY_ID**

<<eXtension>> (p. 270) *Identifier for* ***com.rti.dds.infrastructure.LoggingQosPolicy*** *(p. 1190)*

8.190.1 Detailed Description

Type to identify `QosPolicies`.

8.190.2 Member Data Documentation

8.190.2.1 final QosPolicyId_t INVALID_QOS_POLICY_ID [static]

Identifier for an invalid QoS policy.

8.190.2.2 final QosPolicyId_t USEROBJECT_QOS_POLICY_ID [static]

<<***eXtension***>> *(p. 270)* Identifier for com.rti.dds.infrastructure.UserObjectQosPolicy

8.191 QueryCondition Interface Reference

<<*interface*>> *(p. 271)* These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1326) 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.

8.191.1 Detailed Description

<<*interface*>> *(p. 271)* These are specialised **com.rti.dds.subscription.ReadCondition** (p. 1326) 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. 1326) 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. 1325) 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. 278) describes the syntax of `query_expression` and `query_parameters`.

8.191.2 Member Function Documentation

8.191.2.1 String get_query_expression ()

Retrieves the query expression.

8.191.2.2 void get_query_parameters (StringSeq *query_parameters*)

Retrieves the query parameters.

Parameters:

query_parameters <<*inout*>> *(*p. 271*)* the query parameters are returned here.

8.191.2.3 void set_query_parameters (StringSeq *query_parameters*)

Sets the query parameters.

Parameters:

query_parameters <<*in*>> *(*p. 271*)* the new query parameters

8.192 ReadCondition Interface Reference

<<*interface*>> *(p. 271)* Conditions specifically dedicated to read operations and attached to one **com.rti.dds.subscription.DataReader** (p. 473).

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.*

- **DataReader get_datareader** ()

  Returns the ***com.rti.dds.subscription.DataReader*** *(p. 473) associated with the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

8.192.1 Detailed Description

<<*interface*>> *(p. 271)* Conditions specifically dedicated to read operations and attached to one **com.rti.dds.subscription.DataReader** (p. 473).

com.rti.dds.subscription.ReadCondition (p. 1326) 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.dds.topic.example.FooDataReader.read and com.rti.dds.topic.example.FooDataReader.take 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. 1326) may be attached to the same **com.rti.dds.subscription.DataReader** (p. 473).

Note: If you are using a **ReadCondition** (p. 1326) simply to detect the presence of new data, consider using a **com.rti.dds.infrastructure.StatusCondition**

(p. 1452) with the DATA_AVAILABLE_STATUS instead, which will perform better in this situation.

8.192.2 Member Function Documentation

8.192.2.1 int get_sample_state_mask ()

Retrieves the set of `sample_states` for the condition.

8.192.2.2 int get_view_state_mask ()

Retrieves the set of `view_states` for the condition.

8.192.2.3 int get_instance_state_mask ()

Retrieves the set of `instance_states` for the condition.

8.192.2.4 DataReader get_datareader ()

Returns the **com.rti.dds.subscription.DataReader** (p. 473) associated with the **com.rti.dds.subscription.ReadCondition** (p. 1326).

There is exactly one **com.rti.dds.subscription.DataReader** (p. 473) associated with each **com.rti.dds.subscription.ReadCondition** (p. 1326).

Returns:

com.rti.dds.subscription.DataReader (p. 473) associated with the **com.rti.dds.subscription.ReadCondition** (p. 1326).

8.193 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**

  Maximum duration for which the ***com.rti.dds.subscription.DataReader*** *(p. 473) will maintain information regarding an instance once its* `instance_state` *becomes InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.*

- final **Duration_t autopurge_disposed_samples_delay**

  Maximum duration for which the ***com.rti.dds.subscription.DataReader*** *(p. 473) will maintain samples for an instance once its* `instance_state` *becomes InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE.*

8.193.1 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. 473) must retain information regarding instances that have the instance_state InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **com.rti.dds.subscription.DataReader** (p. 473) 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. 1071) and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356).

The **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473) will have taken for that instance will have an `instance_state` of either InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE or InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 99), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **com.rti.dds.subscription.DataReader** (p. 473) from reclaiming the resources and they would remain in the **com.rti.dds.subscription.DataReader** (p. 473) indefinitely.

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. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

8.193.2 Member Data Documentation

8.193.2.1 final Duration_t autopurge_nowriter_samples_delay

Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 473) will maintain information regarding an instance once its `instance_state` becomes InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

After this time elapses, the **com.rti.dds.subscription.DataReader** (p. 473) will purge all internal information regarding the instance, any "untaken" samples will also be lost.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE

[range] [1 nanosec, 1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.193.2.2 final Duration_t autopurge_disposed_samples_delay

Maximum duration for which the **com.rti.dds.subscription.DataReader** (p. 473) will maintain samples for an instance once its `instance_state` becomes InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE.

After this time elapses, the **com.rti.dds.subscription.DataReader** (p. 473) will purge all samples for the instance.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE

[range] [1 nanosec, 1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.194 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.
- int **buffer_alignment**
  The receive buffer alignment.

8.194.1 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. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

See also:

Controlling CPU Core Affinity for RTI Threads (p. 1534)

8.194.2 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. 473), set a unique port for the **com.rti.dds.infrastructure.TransportUnicastQosPolicy** (p. 1605) or **com.rti.dds.infrastructure.TransportMulticastQosPolicy** (p. 1590) for the **com.rti.dds.subscription.DataReader** (p. 473).

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. 629) 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.

8.194.3 Member Data Documentation

8.194.3.1 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 Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 29

For INTEGRITY: 100

For VxWorks: 71

For all others: OS default priority.

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For INTEGRITY: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[**default**] mask **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_FLOATING_POINT** (p. 1536) | **com.rti.dds.infrastructure.ThreadSettingsKind.THREAD_SETTINGS_STDIO** (p. 1536)

8.194.3.2 int buffer_size

The receive buffer size.

The receive buffer is used by the receive thread to store the raw data that arrives over the transport.

In many applications, users will change the configuration of the built-in transport Transport.Property_t.message_size_max to increase the size of the largest data packet that can be sent or received through the transport. Typically, users will change the UDPv4 transport plugin's Transport.Property_t.message_size_max to 65536 (64 K), which is the largest packet that can be sent/received via UDP.

Ihe ReceiverPoolQosPolicy's `buffer_size` should be set to be the same value as the maximum Transport.Property_t.message_size_max across *all* of the transports being used.

If you are using the default configuration of the built-in transports, you should not need to change this buffer size.

In addition, if your application *only* uses transports that support zero-copy, then you do not need to modify the value of buffer_size, even if the Transport.Property_t.message_size_max of the transport is changed. Transports that support zero-copy do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in a buffer allocated by the transport itself. The only built-in transport that supports zero-copy is the UDPv4 transport on VxWorks platforms.

[**default**] 9216

[**range**] [1, 1 GB]

8.194.3.3 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.

8.195 RefilterQosPolicyKind Class Reference

<<*eXtension*>> *(p. 270)* Kinds of Refiltering

Inheritance diagram for RefilterQosPolicyKind::

Static Public Attributes

- static final **RefilterQosPolicyKind NONE_REFILTER_QOS**

  [**default**] *Do not filter existing samples for a new reader*

- static final **RefilterQosPolicyKind ALL_REFILTER_QOS**

  Filter all existing samples for a new reader.

- static final **RefilterQosPolicyKind ON_DEMAND_REFILTER_QOS**

  Filter existing samples only when they are requested by the reader.

8.195.1 Detailed Description

<<*eXtension*>> *(p. 270)* Kinds of Refiltering

QoS:

com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)

8.195.2 Member Data Documentation

8.195.2.1 final RefilterQosPolicyKind NONE_REFILTER_QOS [static]

[**default**] Do not filter existing samples for a new reader

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue for the new reader. This option configures the writer to not filter any existing samples for the reader and the reader will do the filtering.

8.195.2.2 final RefilterQosPolicyKind ALL_REFILTER_QOS [static]

Filter all existing samples for a new reader.

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue. This option configures the writer to filter all existing samples for the reader when a new reader is matched to the writer.

8.195.2.3 final RefilterQosPolicyKind ON_DEMAND_REFILTER_QOS [static]

Filter existing samples only when they are requested by the reader.

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue. This option configures the writer to filter only existing samples that are requested by the reader.

8.196 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 writer may block on a write() call.

8.196.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Connext.

Entity:

com.rti.dds.topic.Topic (p. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1459), **StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1459)

Properties:

RxO (p. 97) = YES
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.196.2 Usage

This policy indicates the level of reliability requested by a **com.rti.dds.subscription.DataReader** (p. 473) or offered by a **com.rti.dds.publication.DataWriter** (p. 538).

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, **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340) 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** (p. 350) 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 lost 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. 1071), **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 571), **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504), and **com.rti.dds.infrastructure.ResourceLimitsQosPolicy** (p. 1356).

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, **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340) < **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341). A **com.rti.dds.publication.DataWriter** (p. 538) offering one level is implicitly offering all levels below.

Note: To send *large* data reliably, you will also need to set **PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1312). *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. 102) policy. In case the reliability kind is set to **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341) the write operation on the **com.rti.dds.publication.DataWriter** (p. 538) 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. 102) to be exceeded. Under these circumstances, the **RELIABILITY** (p. 101) `max_blocking_time` configures the maximum duration the write operation may block.

If the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) is set to **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341), data samples originating from a single **com.rti.dds.publication.DataWriter** (p. 538) cannot be made available to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 538) history before it is accessible by the **com.rti.dds.subscription.DataReader** (p. 473).

If the **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) is set to **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340), 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. 473) history in the same order they originated in the **com.rti.dds.publication.DataWriter** (p. 538). In other words, the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1071)
com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1356)

8.196.3 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. 1339) are considered ordered such that **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340) < **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341).

8.196.4 Member Data Documentation

8.196.4.1 ReliabilityQosPolicyKind kind

Kind of reliability.

[**default**] **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340) for **com.rti.dds.subscription.DataReader** (p. 473) and **com.rti.dds.topic.Topic** (p. 1545), **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341) for **com.rti.dds.publication.DataWriter** (p. 538)

8.196.4.2 final Duration_t max_blocking_time

The maximum time a writer may block on a write() call.

This setting applies only to the case where **com.rti.dds.infrastructure.ReliabilityQosPolicy.kind** (p. 1339) = **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341). com.rti.dds.topic.example.FooDataWriter.write is allowed to block if the **com.rti.dds.publication.DataWriter** (p. 538) does not have space to store the value written. Only applies to **com.rti.dds.publication.DataWriter** (p. 538).

[**default**] 100 milliseconds

[**range**] [0,1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

See also:

com.rti.dds.infrastructure.ResourceLimitsQosPolicy (p. 1356)

8.197 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.

8.197.1 Detailed Description

Kinds of reliability.

QoS:

com.rti.dds.infrastructure.ReliabilityQosPolicy (p. 1336)

8.197.2 Member Data Documentation

8.197.2.1 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. 473) and **com.rti.dds.topic.Topic** (p. 1545)

8.197.2.2 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. 538)) RTI Connext guarantees that all samples in the **com.rti.dds.publication.DataWriter** (p. 538) history will eventually be delivered to all the **com.rti.dds.subscription.DataReader** (p. 473) objects (subject to timeouts that indicate loss of communication with a particular **com.rti.dds.subscription.Subscriber** (p. 1478)).

Outside steady state the **HISTORY** (p. 75) and **RESOURCE_LIMITS** (p. 102) 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. 538)

8.198 ReliableReaderActivityChangedStatus Class Reference

<<*eXtension*>> *(p. 270)* 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.

8.198.1 Detailed Description

<<*eXtension*>> *(p. 270)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

This status is the reciprocal status to the **com.rti.dds.subscription.LivelinessChangedStatus** (p. 1159) on the reader. It is different than the **com.rti.dds.publication.LivelinessLostStatus** (p. 1162) 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.

8.198.2 Constructor & Destructor Documentation

8.198.2.1 ReliableReaderActivityChangedStatus ()

Construct a new status object with default contents.

8.198.2.2 ReliableReaderActivityChangedStatus (ReliableReaderActivityChangedStatus *src*)

Construct a new status identical to the given status.

Exceptions:

NullPointerException if the given status is null.

8.198.3 Member Data Documentation

8.198.3.1 int active_count

The current number of reliable readers currently matched with this reliable writer.

8.198.3.2 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. 1384) times, each heartbeat having been separated from the previous by the current heartbeat period.

8.198.3.3 int active_count_change

The most recent change in the number of active remote reliable readers.

8.198.3.4 int inactive_count_change

The most recent change in the number of inactive remote reliable readers.

8.198.3.5 final InstanceHandle_t last_instance_handle

The instance handle of the last reliable remote reader to be determined inactive.

8.199 ReliableWriterCacheChangedStatus Class Reference

<<*eXtension*>> *(p. 270)* 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.

ˆ int **unacknowledged_sample_count_peak**

The highest value that unacknowledged_sample_count has reached until now.

8.199.1 Detailed Description

*<<**eXtension**>>* *(p. 270)* A summary of the state of a data writer's cache of unacknowledged samples written.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

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.

8.199.2 Constructor & Destructor Documentation

8.199.2.1 ReliableWriterCacheChangedStatus ()

Construct a new status object.

8.199.2.2 ReliableWriterCacheChangedStatus (ReliableWriterCacheChangedStatus *src*)

A copy constructor.

Parameters:

src Source to copy from.

Exceptions:

NullPointerException if the source object is null.

8.199.3 Member Data Documentation

8.199.3.1 final ReliableWriterCacheEventCount empty_reliable_writer_cache

The number of times the reliable writer's cache of unacknowledged samples has become empty.

8.199.3.2 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.

Applies to writer's cache when the send window is enabled (when both **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size** (p. 1389) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size** (p. 1390) are **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)).

Otherwise, applies when the number of unacknowledged samples has reached the send window limit.

8.199.3.3 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.

8.199.3.4 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.

8.199.3.5 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.

8.199.3.6 int unacknowledged_sample_count_peak

The highest value that unacknowledged_sample_count has reached until now.

8.200 ReliableWriterCacheEventCount Class Reference

<<*eXtension*>> *(p. 270)* 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.

8.200.1 Detailed Description

<<*eXtension*>> *(p. 270)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

See also:

com.rti.dds.publication.ReliableWriterCacheChangedStatus (p. 1345)

8.200.2 Member Data Documentation

8.200.2.1 int total_count

The total number of times the event has occurred.

8.200.2.2 int total_count_change

The incremental number of times the event has occurred since the listener was last invoked or the status read.

8.201 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.

8.201.1 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. 618)

8.201.2 Member Data Documentation

8.201.2.1 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. 617) 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. 1168) of **LivelinessQosPolicyKind.AUTOMATIC_LIVELINESS_QOS** (p. 1169) or **LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_-LIVELINESS_QOS** (p. 1169) and by asserting itself (at the **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.participant_-liveliness_assert_period** (p. 618)) over the Simple Discovery Protocol.

8.201.2.2 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.

8.202 RequestedDeadlineMissedStatus Class Reference

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. 473).*

- 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. 473) for which a deadline was detected.*

8.202.1 Detailed Description

StatusKind.REQUESTED_DEADLINE_MISSED_STATUS.

8.202.2 Member Data Documentation

8.202.2.1 int total_count

Total cumulative count of the deadlines detected for any instance read by the **com.rti.dds.subscription.DataReader** (p. 473).

8.202.2.2 int total_count_change

The incremental number of deadlines detected since the last time the listener was called or the status was read.

8.202.2.3 final InstanceHandle_t last_instance_handle

Handle to the last instance in the **com.rti.dds.subscription.DataReader** (p. 473) for which a deadline was detected.

8.203 RequestedIncompatibleQosStatus Class Reference

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. 473) discovered a **com.rti.dds.publication.DataWriter** (p. 538) for the same **com.rti.dds.topic.Topic** (p. 1545) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 473).*

- 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 PolicyId_t 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. 473) discovered a **com.rti.dds.publication.DataWriter** (p. 538) for the same **com.rti.dds.topic.Topic** (p. 1545) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 473).*

8.203.1 Detailed Description

StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS.

See also:

DURABILITY (p. 65)
PRESENTATION (p. 86)
RELIABILITY (p. 101)
OWNERSHIP (p. 83)
LIVELINESS (p. 78)
DEADLINE (p. 50)

LATENCY_BUDGET (p. 76)
DESTINATION_ORDER (p. 51)

8.203.2 Member Data Documentation

8.203.2.1 int total_count

Total cumulative count of how many times the concerned **com.rti.dds.subscription.DataReader** (p. 473) discovered a **com.rti.dds.publication.DataWriter** (p. 538) for the same **com.rti.dds.topic.Topic** (p. 1545) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 473).

8.203.2.2 int total_count_change

The change in `total_count` since the last time the listener was called or the status was read.

8.203.2.3 QosPolicyId_t last_policy_id

The PolicyId_t of one of the policies that was found to be incompatible the last time an incompatibility was detected.

8.203.2.4 final QosPolicyCountSeq policies

A list containing, for each policy, the total number of times that the concerned **com.rti.dds.subscription.DataReader** (p. 473) discovered a **com.rti.dds.publication.DataWriter** (p. 538) for the same **com.rti.dds.topic.Topic** (p. 1545) with an offered QoS that is incompatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 473).

8.204 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. 538) (or* ***com.rti.dds.subscription.DataReader*** *(p. 473)).*

- int **max_instances**

  Represents the maximum number of instances a ***com.rti.dds.publication.DataWriter*** *(p. 538) (or* ***com.rti.dds.subscription.DataReader*** *(p. 473)) can manage.*

- int **max_samples_per_instance**

  Represents the maximum number of samples of any one instance a ***com.rti.dds.publication.DataWriter*** *(p. 538) (or* ***com.rti.dds.subscription.DataReader*** *(p. 473)) can manage.*

- int **initial_samples**

  <<**eXtension**>> (p. 270) *Represents the initial samples the middleware will store for any one* ***com.rti.dds.publication.DataWriter*** *(p. 538) (or* ***com.rti.dds.subscription.DataReader*** *(p. 473)).*

- int **initial_instances**

  <<**eXtension**>> (p. 270) *Represents the initial number of instances a* ***com.rti.dds.publication.DataWriter*** *(p. 538) (or* ***com.rti.dds.subscription.DataReader*** *(p. 473)) will manage.*

- int **instance_hash_buckets**

  <<**eXtension**>> (p. 270) *Number of hash buckets for instances.*

Static Public Attributes

- static final int **LENGTH_UNLIMITED**

  A special value indicating an unlimited quantity.

8.204.1 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. 1545), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Status:

StatusKind.SAMPLE_REJECTED_STATUS (p. 1460), **com.rti.dds.subscription.SampleRejectedStatus** (p. 1422)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **UNTIL ENABLE** (p. 98)

8.204.2 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. 765)), this QoS policy determines the actual maximum queue size when the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) is set to **HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS** (p. 1076).

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. 538) objects are communicating samples faster than they are ultimately taken by the

com.rti.dds.subscription.DataReader (p. 473) 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. 473) cannot keep up with its corresponding **com.rti.dds.publication.DataWriter** (p. 538). The behavior in this case depends on the setting for the **RELIABILITY** (p. 101). If reliability is **ReliabilityQosPolicyKind.BEST_EFFORT_RELIABILITY_QOS** (p. 1340), then RTI Connext is allowed to drop samples. If the reliability is **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341), RTI Connext will block the **com.rti.dds.publication.DataWriter** (p. 538) or discard the sample at the **com.rti.dds.subscription.DataReader** (p. 473) in order not to lose existing samples.

The constant **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) may be used to indicate the absence of a particular limit. For example setting **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360) to **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102) 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. 538) may block on a write() call even though the **com.rti.dds.infrastructure.HistoryQosPolicy** (p. 1071) is **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075). To guarantee the writer does not block for **HistoryQosPolicyKind.KEEP_LAST_HISTORY_QOS** (p. 1075), 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. 1336)
com.rti.dds.infrastructure.HistoryQosPolicy (p. 1071)

8.204.3 Consistency

The setting of **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) must be consistent with **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360). For these two values to be consistent, it must be true that **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) >= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples_per_instance** (p. 1360). As described above, this limit will not be enforced if **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_**

samples_per_instance (p. 1360) is set to **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

The setting of **RESOURCE_LIMITS** (p. 102) `max_samples_per_instance` must be consistent with the **HISTORY** (p. 75) `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. 1071)

8.204.4 Member Data Documentation

8.204.4.1 int max_samples

Represents the maximum samples the middleware can store for any one **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)).

Specifies the maximum number of data samples a **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) 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 **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

When batching is enabled, the maximum number of data samples a **com.rti.dds.publication.DataWriter** (p. 538) can manage will also be limited by **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 600).

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_samples, >= max_samples_per_instance, >= **com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy.max_samples_per_remote_writer** (p. 528) or >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeats_per_max_samples** (p. 1385)

For **com.rti.dds.publication.DataWriterQos** (p. 588) max_samples >= com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer.heartbeats_per_max_samples if batching is disabled.

8.204.4.2 int max_instances

Represents the maximum number of instances a **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) can manage.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 1 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= initial_instances

8.204.4.3 int max_samples_per_instance

Represents the maximum number of samples of any one instance a **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) can manage.

For unkeyed types, this value has to be equal to max_samples or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102).

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1, 100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), <= max_samples or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), >= **com.rti.dds.infrastructure.HistoryQosPolicy.depth** (p. 1074)

8.204.4.4 int initial_samples

<<***eXtension***>> *(p. 270)* Represents the initial samples the middleware will store for any one **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)).

Specifies the initial number of data samples a **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) will manage across all the instances associated with it.

[default] 32

[range] [1,100 million], <= max_samples

8.204.4.5 int initial_instances

<<***eXtension***>> *(p. 270)* Represents the initial number of instances a **com.rti.dds.publication.DataWriter** (p. 538) (or **com.rti.dds.subscription.DataReader** (p. 473)) will manage.

[default] 32

[range] [1,1 million], <= max_instances

8.204.4.6 int instance_hash_buckets

<<***eXtension***>> *(*p. 270*)* 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]

8.205 RETCODE_ALREADY_DELETED Class Reference

The object target of this operation has already been deleted.

Inheritance diagram for RETCODE_ALREADY_DELETED::

8.205.1 Detailed Description

The object target of this operation has already been deleted.

8.206 RETCODE_BAD_PARAMETER Class Reference

Illegal parameter value.

Inheritance diagram for RETCODE_BAD_PARAMETER::

8.206.1 Detailed Description

Illegal parameter value.

The value of the parameter that is passed in has llegal value. Things that falls into this category includes null parameters and parameter values that are out of range.

8.207 RETCODE_ERROR Class Reference

Generic, unspecified error.

Inheritance diagram for RETCODE_ERROR::

8.207.1 Detailed Description

Generic, unspecified error.

8.208 RETCODE_ILLEGAL_OPERATION Class Reference

The operation was called under improper circumstances.

Inheritance diagram for RETCODE_ILLEGAL_OPERATION::

8.208.1 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 **RETCODE_PRECONDITION_NOT_MET** (p. 1371), except that there is no precondition that could be changed to make the operation succeed.

8.209 RETCODE_IMMUTABLE_POLICY Class Reference

Application attempted to modify an immutable QoS policy.

Inheritance diagram for RETCODE_IMMUTABLE_POLICY::

8.209.1 Detailed Description

Application attempted to modify an immutable QoS policy.

8.210 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::

8.210.1 Detailed Description

Application specified a set of QoS policies that are not consistent with each other.

8.211 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::

8.211.1 Detailed Description

Indicates a transient situation where the operation did not return any data but there is no inherent error.

8.212 RETCODE_NOT_ENABLED Class Reference

Operation invoked on a **com.rti.dds.infrastructure.Entity** (p. 912) that is not yet enabled.

Inheritance diagram for RETCODE_NOT_ENABLED::

8.212.1 Detailed Description

Operation invoked on a **com.rti.dds.infrastructure.Entity** (p. 912) that is not yet enabled.

8.213 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::

8.213.1 Detailed Description

RTI Connext ran out of the resources needed to complete the operation.

8.214 RETCODE_PRECONDITION_NOT_MET Class Reference

A pre-condition for the operation was not met.

Inheritance diagram for RETCODE_PRECONDITION_NOT_MET::

8.214.1 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.

8.215 RETCODE_TIMEOUT Class Reference

The operation timed out.

Inheritance diagram for RETCODE_TIMEOUT::

8.215.1 Detailed Description

The operation timed out.

8.216 RETCODE_UNSUPPORTED Class Reference

Unsupported operation. Can only returned by operations that are unsupported.

Inheritance diagram for RETCODE_UNSUPPORTED::

8.216.1 Detailed Description

Unsupported operation. Can only returned by operations that are unsupported.

8.217 RtpsReliableReaderProtocol_t Class Reference

Qos (p. 1313) 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**)

  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.

8.217.1 Detailed Description

Qos (p. 1313) related to reliable reader protocol defined in RTPS.

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

QoS:

com.rti.dds.infrastructure.DataReaderProtocolQosPolicy (p. 504)
com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615)

8.217.2 Constructor & Destructor Documentation

8.217.2.1 RtpsReliableReaderProtocol_t ()

Constructor with default values.

8.217.2.2 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*)

Constructor with given durations.

8.217.3 Member Data Documentation

8.217.3.1 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

8.217.3.2 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.DiscoveryConfigQosPolicy** (p. 615) : 0 seconds

For **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504) : 0.5 seconds

[range] [0, 1 year], >= min_heartbeat_response_delay

8.217.3.3 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],

8.217.3.4 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]

8.217.3.5 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]

8.217.3.6 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]

8.218 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 cache of a reliable writer meets or falls below this threshold, the ***StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS*** *(p. 1462) is considered to have changed.*

- int **high_watermark**

  When the number of unacknowledged samples in the cache of a reliable writer meets or exceeds this threshold, the ***StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS*** *(p. 1462) 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 heartbeats per send queue.

^ 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 dropped 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.

8.218.1 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. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

QoS:

com.rti.dds.infrastructure.DataWriterProtocolQosPolicy (p. 571)
com.rti.dds.infrastructure.DiscoveryConfigQosPolicy (p. 615)

8.218.2 Member Data Documentation

8.218.2.1 int low_watermark

When the number of unacknowledged samples in the cache of a reliable writer meets or falls below this threshold, the **StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1462) is considered to have changed.

This value is measured in units of samples, except with batching configurations in non-MultiChannel DataWriters 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. 1382) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.fast_heartbeat_period** (p. 1383), respectively). When the number of unacknowledged samples in the queue of a reliable **com.rti.dds.publication.DataWriter** (p. 538) meets or exceeds high_watermark, the **StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1462) is changed, and the DataWriter will start heartbeating at fast_heartbeat_rate. When the number of samples meets or falls below low_watermark, **StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1462) is changed, and the heartbeat rate will return to the "normal" rate (heartbeat_rate).

[**default**] 0

[**range**] [0, 100 million], < high_watermark

See also:

> **Multi-channel DataWriters** (p. 204) for additional details on reliability with MultChannel DataWriters.

8.218.2.2 int high_watermark

When the number of unacknowledged samples in the cache of a reliable writer meets or exceeds this threshold, the **StatusKind.RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1462) is considered to have changed.

This value is measured in units of samples, except with batching configurations in non-MultiChannel DataWriters 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. 739):

For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 621)

high_watermark<=com.rti.dds.domain.DomainParticipantQos.resource_limits.local_writer_allocation.max_count

For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 621)

high_watermark<=com.rti.dds.domain.DomainParticipantQos.resource_limits.local_reader_allocation.max_count

In **com.rti.dds.publication.DataWriterQos.protocol** (p. 592):

For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 575),

high_watermark<=com.**rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) if batching is disabled or the DataWriter is a MultiChannel DataWriter. Otherwise,

high_watermark<=com.**rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 600)

[default] 1

[range] [1, 100 million] or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), > low_watermark <= *maximum* which depends on the container policy

See also:

Multi-channel DataWriters (p. 204) for additional details on reliability with MultChannel DataWriters.

8.218.2.3 final Duration_t heartbeat_period

The period at which to send heartbeats.

A reliable writer will send periodic heartbeats at this rate.

[default] 3 seconds

[range] [1 nanosec,1 year], >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.fast_heartbeat_period** (p. 1383), >= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.late_joiner_heartbeat_period** (p. 1383)

8.218.2.4 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] 3 seconds

[range] [1 nanosec,1 year], <= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1382)

8.218.2.5 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.

[default] 3 seconds

[range] [1 nanosec,1 year], <= **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1382)

8.218.2.6 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] com.rti.dds.infrastructure.Duration_t.AUTO. If **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) is set to **PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS** (p. 1243), this value

is set to **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.heartbeat_period** (p. 1382). Otherwise, the value is set to com.rti.dds.infrastructure.Duration_t.INFINITE.

[range] > 1 nanosec, com.rti.dds.infrastructure.Duration_t.INFINITE, or com.rti.dds.infrastructure.Duration_t.AUTO

8.218.2.7 int samples_per_virtual_heartbeat

The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

[default] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[range] [1,1000000], **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.218.2.8 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 **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

8.218.2.9 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. 1384) have been sent without receiving any ack/nacks back.

By setting this true, it changes the conditions of inactivating a remote reader: a reader will be considered inactive when it either does not send any ack/nacks or keeps sending non-progressing nacks for **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_**

heartbeat_retries (p. 1384) number of heartbeat periods, where a non-progressing nack is one whose oldest sample requested has not advanced from the oldest sample requested of the previous nack.

[**default**] false

8.218.2.10 int heartbeats_per_max_samples

The number of heartbeats per send queue.

If batching is disabled or the DataWriter is a MultiChannel DataWriter: a piggyback heartbeat will be sent every [**com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359)/heartbeats_per_max_samples] number of samples.

Otherwise: a piggyback heartbeat will be sent every [**com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 600)/heartbeats_per_max_samples] number of batches.

If set to zero, no piggyback heartbeat will be sent. If maximum is **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), 100 million is assumed as the maximum value in the calculation.

[**default**] 8

[**range**] [0, 100 million]

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.publication_writer** (p. 621):

  heartbeats_per_max_samples<= com.rti.dds.domain.DomainParticipantQos.resource_limits.local_writer_allocation.max_count

- For **com.rti.dds.infrastructure.DiscoveryConfigQosPolicy.subscription_writer** (p. 621):

  heartbeats_per_max_samples<= com.rti.dds.domain.DomainParticipantQos.resource_limits.local_reader_allocation.max_count

- For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_reliable_writer** (p. 575):

  heartbeats_per_max_samples<= **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) if batching is disabled or the DataWriter is a MultiChannel DataWriter. Otherwise:

  heartbeats_per_max_samples<= **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.max_batches** (p. 600).

8.218.2.11 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

8.218.2.12 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.DiscoveryConfigQosPolicy** (p. 615) : 0 seconds

For **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy** (p. 571) : 0.2 seconds

[range] [0,1 day], >= min_nack_response_delay

8.218.2.13 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],

8.218.2.14 int max_bytes_per_nack_response

The maximum total message size when resending dropped 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.

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] 131072

[range] [0, 1 GB]

8.218.2.15 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. 573) = true) or a data reader (**com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 507) = 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. 1387)

8.218.2.16 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. 573) = true) or a data reader (**com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.disable_positive_acks** (p. 507) = 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. 1387)

8.218.2.17 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. 1388) = 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. 1389). 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. 1388).

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. 1387) and **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_max_sample_keep_duration** (p. 1387), respectively.

When **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1388) = false, the sample keep duration is set to **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.disable_positive_acks_min_sample_keep_duration** (p. 1387) .

[default] true

8.218.2.18 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. 1388) = 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

8.218.2.19 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. 1388) = 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

8.218.2.20 int min_send_window_size

Minimum size of send window of unacknowledged samples.

A **com.rti.dds.publication.DataWriter** (p. 538) 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 both min_send_window_size and max_send_window_size are **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), then **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) serves as the effective send window limit.

When **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_-**

samples (p. 1359) is less than max_send_window_size, then it serves as the effective max send window. If it is less than min_send_window_size, then effectively both min and max send window sizes are equal to max samples. In addition, the low and high watermarks are scaled down linearly to stay within the send window size, and the full reliable queue status is set when the send window is full.

[**default**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[**range**] > 0, <= max_send_window_size, or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

See also:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.max_send_window_size (p. 1390)
com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1381)
com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark (p. 1381)
com.rti.dds.publication.ReliableWriterCacheChangedStatus.full_reliable_writer_cache (p. 1347)

8.218.2.21 int max_send_window_size

Maximum size of send window of unacknowledged samples.

A **com.rti.dds.publication.DataWriter** (p. 538) 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 both min_send_window_size and max_send_window_size are **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102), then **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) serves as the effective send window limit. When **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359) is less than max_send_window_size, then it serves as the effective max send window. If it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max samples. In addition, the low and high watermarks are scaled down linearly to stay within the send window size, and the full reliable queue status is set when the send window is full.

[**default**] **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

[**range**] > 0, >= min_send_window_size, or **ResourceLimitsQosPolicy.LENGTH_UNLIMITED** (p. 102)

See also:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.min_send_window_size (p. 1389)
com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark (p. 1381)
com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark (p. 1381)
com.rti.dds.publication.ReliableWriterCacheChangedStatus.full_reliable_writer_cache (p. 1347)

8.218.2.22 final Duration_t send_window_update_period

Period in which send window may be dynamically changed.

The **com.rti.dds.publication.DataWriter** (p. 538)'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**] 3 seconds

[**range**] > [0,1 year]

See also:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_increase_factor (p. 1391), **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_decrease_factor** (p. 1392)

8.218.2.23 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. 538)'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. 1391), **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_decrease_factor** (p. 1392)

8.218.2.24 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. 538)'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] 70

[range] [0, 100]

See also:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_update_period (p. 1391), **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.send_window_increase_factor** (p. 1391)

8.218.2.25 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]

8.218.2.26 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**] false

8.219 RtpsReservedPortKind Class Reference

RTPS reserved port kind, used to identify the types of ports that can be reserved on **domain** (p. 317) 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. 1717).*

- static final int **MASK_NONE**

  No bits are set.

- static final int **MASK_ALL**

  All bits are set.

8.219.1 Detailed Description

RTPS reserved port kind, used to identify the types of ports that can be reserved on **domain** (p. 317) participant enable.

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_reserved_port_mask (p. 1717)

8.219.2 Member Data Documentation

8.219.2.1 final int BUILTIN_UNICAST = 0x0001 << 0 [static]

Select the **metatraffic** unicast port.

8.219.2.2 final int BUILTIN_MULTICAST = 0x0001 << 1 [static]

Select the **metatraffic** multicast port.

8.219.2.3 final int USER_UNICAST = 0x0001 << 2 [static]

Select the **usertraffic** unicast port.

8.219.2.4 final int USER_MULTICAST = 0x0001 << 3 [static]

Select the **usertraffic** multicast port.

8.220 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*** *(*p. 317*) 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.

8.220.1 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. 1396). 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. 1396), the port numbers depend on:

- `domain_id`, as specified in **com.rti.dds.domain.DomainParticipantFactory.create_participant** (p. 714)

- `participant_id`, as specified using **com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id** (p. 1714)

The `domain_id` parameter ensures no port conflicts exist between Participants belonging to different domains. This also means that discovery metatraffic in one **domain** (p. 317) is not visible to Participants in a different **domain** (p. 317). The `participant_id` parameter ensures that unique unicast port numbers are assigned to Participants belonging to the same **domain** (p. 317) 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) + builtin_u
```

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. 625) on a **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 522), **com.rti.dds.subscription.DataReaderQos.unicast** (p. 522), or by the **com.rti.dds.publication.DataWriterQos.unicast** (p. 593) 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
```

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. 1590).

```
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 **RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 131)).

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** (p. 317). 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].

QoS:

com.rti.dds.infrastructure.WireProtocolQosPolicy (p. 1709)

8.220.2 Member Data Documentation

8.220.2.1 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.

8.220.2.2 int domain_id_gain

Tunable **domain** (p. 317) 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. 629) instances within a single **domain** (p. 317) occupy a consecutive range of `domain_id_gain` ports. Precisely, all ports occupied by the **domain** (p. 317) 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. 629) 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.

8.220.2.3 int participant_id_gain

Tunable participant gain parameter.

Multiplier of the `participant_id`. See **com.rti.dds.infrastructure.RtpsWellKnownPorts_t.domain_id_gain** (p. 1399) 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.

8.220.2.4 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.

8.220.2.5 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.

8.220.2.6 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.

8.220.2.7 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.

8.221 SampleIdentity_t Class Reference

Type definition for an Sample Identity.

Inherits Struct.

Public Member Functions

- **SampleIdentity_t** (**SampleIdentity_t** other)

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**

8.221.1 Detailed Description

Type definition for an Sample Identity.

A SampleIdentity defines a pair (Virtual Writer GUID, Sequence Number) that uniquely identifies a sample within a DDS **domain** (p. 317) and a Topic.

8.221.2 Constructor & Destructor Documentation

8.221.2.1 SampleIdentity_t (SampleIdentity_t *other*)

8.221.3 Member Data Documentation

8.221.3.1 final SampleIdentity_t AUTO_SAMPLE_IDENTITY [static]

Initial value:

```
new SampleIdentity_t(GUID_t.GUID_AUTO, SequenceNumber_t.AUTO_SEQUENCE_NUMBER)
```

8.221.3.2 final GUID_t writer_guid = new GUID_t(GUID_t.GUID_AUTO)

16-byte identifier identifying the virtual GUID.

8.221.3.3 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.

8.222 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. 270) *The timestamp when the sample was committed by a* ***DataReader*** *(p. 473).*

- final **SequenceNumber_t publication_sequence_number**

  <<**eXtension**>> (p. 270) *The* ***publication*** *(p. 338) sequence number.*

- final **SequenceNumber_t reception_sequence_number**

  <<**eXtension**>> (p. 270) *The reception sequence number when sample was committed by a* ***DataReader*** *(p. 473)*

- final **GUID_t original_publication_virtual_guid**

  <<**eXtension**>> (p. 270) *The original* ***publication*** *(p. 338) virtual GUID.*

- final **SequenceNumber_t original_publication_virtual_sequence_number**

  <<**eXtension**>> (p. 270) *The original* ***publication*** *(p. 338) virtual sequence number.*

Static Package Functions

- static **SampleInfo get_from_native** (long native_sample_info)

8.222.1 Detailed Description

Information that accompanies each sample that is `read` or `taken`.

8.222.2 Interpretation of the SampleInfo

The **com.rti.dds.subscription.SampleInfo** (p. 1404) 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)
- the `valid_data` 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=` InstanceStateKind.ALIVE_INTANCE_STATE) 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. 473). This rank provides a preview of what is available within the **com.rti.dds.subscription.DataReader** (p. 473).
- The `source_timestamp` of the sample. This is the timestamp provided by the **com.rti.dds.publication.DataWriter** (p. 538) at the time the sample was produced.

8.222.3 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. 1410) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1411), relative to each **DataReader** (p. 473):

- The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1410) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1411) are initialized to zero when the **com.rti.dds.subscription.DataReader** (p. 473) first detects the presence of a never-seen-before instance.

- The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1410) is incremented each time the instance_state of the corresponding instance changes from **InstanceStateKind.NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 1088) to InstanceStateKind.ALIVE_INTANCE_STATE.

- The **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1411) is incremented each time the instance_state of the corresponding instance changes from **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088) to InstanceStateKind.ALIVE_INTANCE_STATE.

- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **com.rti.dds.subscription.SampleInfo.disposed_generation_count** (p. 1410) and **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1411) available in the **com.rti.dds.subscription.SampleInfo** (p. 1404) capture a snapshot of the corresponding counters at the time the sample was received.

8.222.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **com.rti.dds.subscription.SampleInfo.sample_rank** (p. 1411) and **com.rti.dds.subscription.SampleInfo.generation_rank** (p. 1411) available in the **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1411) 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. 1411) available in the **com.rti.dds.subscription.SampleInfo** (p. 1404) 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. 1411) 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. 1411) available in the **com.rti.dds.subscription.SampleInfo** (p. 1404) 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)
```

8.222.5 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. 1404), the application can anticipate what other information regarding the same instance appears in the returned collection, as well as in the **infrastructure** (p. 323) and thus make appropriate decisions.

For **example** (p. 349), 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. 1430),
com.rti.dds.subscription.InstanceStateKind (p. 1086),

com.rti.dds.subscription.ViewStateKind (p. 1689),
com.rti.dds.subscription.SampleInfo.valid_data (p. 1412)

8.222.6 Member Function Documentation

8.222.6.1 Object copy_from (Object *other*)

Implementation of the `Copyable` interface.

See also:

com.rti.dds.infrastructure.Copyable.copy_from (p. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.222.6.2 static SampleInfo get_from_native (long *native_sample_info*) [static, package]

Given a pointer to a native **SampleInfo** (p. 1404) object from the queue, get a reference to the corresponding Java **SampleInfo** (p. 1404) object.

8.222.7 Member Data Documentation

8.222.7.1 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. 1430)

8.222.7.2 int view_state

The view state of the instance.

Indicates whether the **com.rti.dds.subscription.DataReader** (p. 473) has already seen samples for the most-current generation of the related instance.

See also:

com.rti.dds.subscription.ViewStateKind (p. 1689)

8.222.7.3 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. 1086)

8.222.7.4 final Time_t source_timestamp

The timestamp when the sample was written by a DataWriter.

8.222.7.5 final InstanceHandle_t instance_handle

Identifies locally the corresponding instance.

8.222.7.6 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. 1080) that is returned by the operation **com.rti.dds.subscription.DataReader.get_matched_publications** (p. 486) and can also be used as a parameter to the operation **com.rti.dds.subscription.DataReader.get_matched_publication_data** (p. 487).

8.222.7.7 int disposed_generation_count

The disposed generation count of the instance at the time of sample reception.

Indicates the number of times the instance had become alive after it was disposed explicitly by a **com.rti.dds.publication.DataWriter** (p. 538), at the time the sample was received.

See also:

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1406) **Interpretation of the SampleInfo counters and ranks** (p. 1408)

8.222.7.8 int no_writers_generation_count

The no writers generation count of the instance at the time of sample reception.

Indicates the number of times the instance had become alive after it was disposed because there were no writers, at the time the sample was received.

See also:

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1406) **Interpretation of the SampleInfo counters and ranks** (p. 1408)

8.222.7.9 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. 1407) **Interpretation of the SampleInfo counters and ranks** (p. 1408)

8.222.7.10 int generation_rank

The 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 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. 1407) **Interpretation of the SampleInfo counters and ranks** (p. 1408)

8.222.7.11 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. 1407) **Interpretation of the SampleInfo counters and ranks** (p. 1408)

8.222.7.12 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. 1404) and some Data. However there are situations where a `DataSample` contains only the **com.rti.dds.subscription.SampleInfo** (p. 1404) 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** (p. 349) of this situation is when the RTI Connext detects that an instance has no writers and changes the corresponding instance_state to **InstanceStateKind.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 1088).

The application can distinguish whether a particular `DataSample` has data by examining the value of the `valid_data` flag. If this flag is set to true, then the `DataSample` contains valid Data. If the flag is set to false, the `DataSample` contains no Data.

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 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. 1404).

8.222.7.13 final Time_t reception_timestamp

*<<**eXtension**>> (p. 270)* The timestamp when the sample was committed by a **DataReader** (p. 473).

8.222.7.14 final SequenceNumber_t publication_sequence_number

<<*eXtension*>> *(p. 270)* The **publication** (p. 338) sequence number.

8.222.7.15 final SequenceNumber_t reception_sequence_number

<<*eXtension*>> *(p. 270)* The reception sequence number when sample was committed by a **DataReader** (p. 473)

8.222.7.16 final GUID_t original_publication_virtual_guid

<<*eXtension*>> *(p. 270)* The original **publication** (p. 338) virtual GUID.

If the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) of the **com.rti.dds.publication.Publisher** (p. 1277) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS, this field contains the **com.rti.dds.publication.Publisher** (p. 1277) virtual GUID that uniquely identifies the DataWriter group.

8.222.7.17 final SequenceNumber_t original_publication_virtual_sequence_number

<<*eXtension*>> *(p. 270)* The original **publication** (p. 338) virtual sequence number.

If the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) of the **com.rti.dds.publication.Publisher** (p. 1277) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS, this field contains the **com.rti.dds.publication.Publisher** (p. 1277) virtual sequence number that uniquely identifies a sample within the DataWriter group.

8.223 SampleInfoSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.SampleInfo** (p. 1404) > .

Inheritance diagram for SampleInfoSeq::

8.223.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.SampleInfo** (p. 1404) > .

See also:

com.rti.dds.util.Sequence (p. 1432)

8.224 SampleLostStatus Class Reference

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. 1545).*

- int **total_count_change**

  The incremental number of samples lost since the last time the listener was called or the status was read.

8.224.1 Detailed Description

StatusKind.SAMPLE_LOST_STATUS_STATUS.

8.224.2 Member Data Documentation

8.224.2.1 int total_count

Total cumulative count of all samples lost across all instances of data published under the **com.rti.dds.topic.Topic** (p. 1545).

8.224.2.2 int total_count_change

The incremental number of samples lost since the last time the listener was called or the status was read.

8.225 SampleLostStatusKind Class Reference

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 DataWriter removed the sample before being received by the **com.rti.dds.subscription.DataReader** (p. 473).*

- static final **SampleLostStatusKind LOST_BY_INSTANCES_LIMIT**

  A resource limit on the number of instances 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. 473) may read was reached.*

- static final **SampleLostStatusKind LOST_BY_INCOMPLETE_COHERENT_SET**

  A sample is lost because it is part of an incomplete coherent set.

- static final **SampleLostStatusKind LOST_BY_LARGE_COHERENT_SET**

  A sample is lost because it is part of a large coherent set.

- static final **SampleLostStatusKind LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT**

  *A resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 473) may store was reached.*

- 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. 473) may read was reached.*

- static final **SampleLostStatusKind LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT**

  A resource limit on the number of remote writers per sample was reached.

- static final **SampleLostStatusKind LOST_BY_AVAILABILITY_WAITING_TIME**

  com.rti.dds.infrastructure.AvailabilityQosPolicy.max_data_availability_waiting_time *(p. 394) 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. 473) may store was reached.*

8.225.1 Detailed Description

Kinds of reasons why a sample was lost.

8.225.2 Member Data Documentation

8.225.2.1 final SampleLostStatusKind NOT_LOST [static]

Initial value:

```
new com.rti.dds.subscription.SampleLostStatusKind(
          "NOT_LOST", 0)
```

The sample was not lost.

See also:

ResourceLimitsQosPolicy

8.225.2.2 final SampleLostStatusKind LOST_BY_WRITER [static]

Initial value:

```
new com.rti.dds.subscription.SampleLostStatusKind(
          "LOST_BY_WRITER", 1)
```

A DataWriter removed the sample before being received by the **com.rti.dds.subscription.DataReader** (p. 473).

This constant is an extension to the DDS standard.

8.225.2.3 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 was reached.

This constant is an extension to the DDS standard.

See also:

ResourceLimitsQosPolicy

8.225.2.4 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. 473) may read was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.225.2.5 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.

This constant is an extension to the DDS standard.

8.225.2.6 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.

This constant is an extension to the DDS standard.

8.225.2.7 final SampleLostStatusKind LOST_BY_-SAMPLES_PER_REMOTE_WRITER_LIMIT [static]

Initial value:

```
new SampleLostStatusKind(
              "LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT", 6)
```

A resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 473) may store was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.225.2.8 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. 473) may read was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.225.2.9 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 was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.225.2.10 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. 394) expired.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.225.2.11 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. 473) may store was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.226 SampleRejectedStatus Class Reference

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. 473).*

- 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.

8.226.1 Detailed Description

StatusKind.SAMPLE_REJECTED_STATUS.

8.226.2 Member Data Documentation

8.226.2.1 int total_count

Total cumulative count of samples rejected by the **com.rti.dds.subscription.DataReader** (p. 473).

8.226.2.2 int total_count_change

The incremental number of samples rejected since the last time the listener was called or the status was read.

8.226.2.3 SampleRejectedStatusKind last_reason

Reason for rejecting the last sample rejected.

See also:

SampleRejectedStatusKind (p. 1424)

8.226.2.4 final InstanceHandle_t last_instance_handle

Handle to the instance being updated by the last sample that was rejected.

8.227 SampleRejectedStatusKind Class Reference

Kinds of reasons for rejecting a sample.

Inheritance diagram for SampleRejectedStatusKind::

Static Public Attributes

- static final **SampleRejectedStatusKind NOT_REJECTED**

  Samples are never rejected.

- static final **SampleRejectedStatusKind REJECTED_BY_INSTANCES_LIMIT**

  A resource limit on the number of instances was reached.

- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_LIMIT**

  A resource limit on the number of samples was reached.

- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT**

  A resource limit on the number of samples per instance was reached.

- static final **SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITERS_LIMIT**

  *A resource limit on the number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read was reached.*

- static final **SampleRejectedStatusKind REJECTED_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. 473) may read was reached.*

- static final **SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT**

  *A resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 473) may store was reached.*

- static final **SampleRejectedStatusKind REJECTED_BY_VIRTUAL_WRITERS_LIMIT**

*A resource limit on the number of virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read was reached.*

- static final **SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT**

  A resource limit on the number of remote writers per sample was reached.

- 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. 473) may store was reached.*

8.227.1 Detailed Description

Kinds of reasons for rejecting a sample.

8.227.2 Member Data Documentation

8.227.2.1 final SampleRejectedStatusKind NOT_REJECTED [static]

Initial value:

```
new com.rti.dds.subscription.SampleRejectedStatusKind(
          "NOT_REJECTED", 0)
```

Samples are never rejected.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.2 final SampleRejectedStatusKind REJECTED_BY_INSTANCES_LIMIT [static]

Initial value:

```
new com.rti.dds.subscription.SampleRejectedStatusKind(
          "REJECTED_BY_INSTANCES_LIMIT", 1)
```

A resource limit on the number of instances was reached.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.3 final SampleRejectedStatusKind REJECTED_BY_SAMPLES_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
          "REJECTED_BY_SAMPLES_LIMIT", 2)
```

A resource limit on the number of samples was reached.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.4 final SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
              "REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT", 3)
```

A resource limit on the number of samples per instance was reached.

See also:

ResourceLimitsQosPolicy

8.227.2.5 final SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITERS_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
              "REJECTED_BY_REMOTE_WRITERS_LIMIT", 4)
```

A resource limit on the number of remote writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.6 final SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
                "REJECTED_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT", 5)
```

A resource limit on the number of remote writers for a single instance from which a **com.rti.dds.subscription.DataReader** (p. 473) may read was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.7 final SampleRejectedStatusKind REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
                "REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT", 6)
```

A resource limit on the number of samples from a given remote writer that a **com.rti.dds.subscription.DataReader** (p. 473) may store was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.8 final SampleRejectedStatusKind REJECTED_BY_VIRTUAL_WRITERS_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
              "REJECTED_BY_VIRTUAL_WRITERS_LIMIT", 7)
```

A resource limit on the number of virtual writers from which a **com.rti.dds.subscription.DataReader** (p. 473) may read was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.9 final SampleRejectedStatusKind REJECTED_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT [static]

Initial value:

```
new SampleRejectedStatusKind(
              "REJECTED_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT", 8)
```

A resource limit on the number of remote writers per sample was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.227.2.10 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. 473) may store was reached.

This constant is an extension to the DDS standard.

See also:

com.rti.dds.infrastructure.DataReaderResourceLimitsQosPolicy (p. 524)

8.228 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 ***SampleStateKind.READ_SAMPLE_STATE*** *(*p. 1430*)* | ***SampleStateKind.NOT_READ_SAMPLE_STATE*** *(*p. 1431*).*

8.228.1 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. 473). The sample state can be either:

- **SampleStateKind.READ_SAMPLE_STATE** (p. 1430) indicates that the **com.rti.dds.subscription.DataReader** (p. 473) has already accessed that sample by means of a read or take operation.

- **SampleStateKind.NOT_READ_SAMPLE_STATE** (p. 1431) indicates that the **com.rti.dds.subscription.DataReader** (p. 473) 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.

8.228.2 Member Data Documentation

8.228.2.1 final int READ_SAMPLE_STATE = 0x0001 << 0 [static]

Sample has been read.

8.228.2.2 final int NOT_READ_SAMPLE_STATE = 0x0001 << 1 [static]

Sample has not been read.

8.229 Sequence Interface Reference

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo`.

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** ()

8.229.1 Detailed Description

*<<**interface**>> (p. 271) <<**generic**>> (p. 271)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo`.

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 **Sequence.setMaximum** (p. 1433).

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.dds.topic.example.FooDataWriter, com.rti.dds.topic.example.FooDataReader, **com.rti.dds.topic.example.FooTypeSupport** (p. 1060), **rtiddsgen** (p. 290)

8.229.2 Member Function Documentation

8.229.2.1 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 **Sequence.setMaximum** (p. 1433).

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implemented in **BooleanSeq** (p. 409), **ByteSeq** (p. 432), **CharSeq** (p. 449), **ConditionSeq** (p. 453), **DoubleSeq** (p. 763), **FloatSeq** (p. 940), **IntSeq** (p. 1093), **LongSeq** (p. 1203), **ShortSeq** (p. 1450), **PublisherSeq** (p. 1306), **DataReaderSeq** (p. 536), **SubscriberSeq** (p. 1508), and **LoanableSequence** (p. 1172).

8.229.2.2 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.

Returns:

true on success, false if the preconditions are not met. In that case the sequence is not modified.

Implemented in **AbstractSequence** (p. 382), and **LoanableSequence** (p. 1172).

8.229.2.3 Class getElementType ()

Returns:

a common supertype for all elements in this sequence.

Implemented in **AbstractPrimitiveSequence** (p. 377), and **AbstractSequence** (p. 383).

8.230 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.

8.230.1 Detailed Description

Type for *sequence* number representation.

Represents a 64-bit sequence number.

8.230.2 Constructor & Destructor Documentation

8.230.2.1 SequenceNumber_t ()

Constructor.

8.230.2.2 SequenceNumber_t (SequenceNumber_t *sn*)

Copy constructor.

Parameters:

sn The sequence number instance to copy. It must not be null.

8.230.2.3 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. *1363)*

8.230.3 Member Function Documentation

8.230.3.1 int compare (SequenceNumber_t *sn*)

Compares two sequence numbers.

Parameters:

sn *<<**in**>>* *(*p. *271)* 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.

8.230.3.2 void plusplus ()

Increases the value of this by one.

8.230.3.3 void minusminus ()

Decreases the value of this by one.

8.230.3.4 SequenceNumber_t add (SequenceNumber_t *val*)

Returns a sequence number whose value is (this + val).

Returns:

(this+val)

8.230.3.5 SequenceNumber_t subtract (SequenceNumber_t *val*)

Returns a sequence number whose value is (this - val).

Returns:

(this-val)

8.230.4 Member Data Documentation

8.230.4.1 final SequenceNumber_t SEQUENCE_NUMBER_UNKNOWN [static]

Unknown sequence number.

8.230.4.2 final SequenceNumber_t AUTO_SEQUENCE_NUMBER [static]

The sequence number is internally determined by RTI Connext.

8.230.4.3 final SequenceNumber_t SEQUENCE_NUMBER_ZERO [static]

Zero value for the sequence number.

8.230.4.4 final SequenceNumber_t SEQUENCE_NUMBER_MAX [static]

Highest, most positive value for the sequence number.

8.230.4.5 int high

The most significant part of the sequence number.

8.230.4.6 long low

The least significant part of the sequence number.

8.231 ShmemTransport Interface Reference

Built-in **transport** (p. 367) plug-in for inter-process communications using shared memory.

Inheritance diagram for ShmemTransport::

Classes

- ˆ class **Property_t**

  *Subclass of **Transport.Property_t** (p. 1570) allowing specification of parameters that are specific to the shared-memory **transport** (p. 367).*

8.231.1 Detailed Description

Built-in **transport** (p. 367) plug-in for inter-process communications using shared memory.

This plugin uses System Shared Memory to send messages between processes on the same node.

The **transport** (p. 367) plugin has exactly one "receive interface"; since the `address_bit_count` is 0, it can be assigned any address. Thus the interface is located by the "network address" associated with the **transport** (p. 367) plugin.

8.231.2 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** (p. 367) 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** (p. 367) are compatible with those specified in its **transport** (p. 367) 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** (p. 367) 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** (p. 367) plugin instances should be adjusted, so that they are compatible.

8.231.3 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** (p. 367) 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** (p. 367) 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.

8.231.4 Shared Resource Keys

The **transport** (p. 367) uses the **shared memory segment keys**, given by the formula below.

```
0x400000 + port
```

The **transport** (p. 367) also uses signaling **shared semaphore keys** given by the formula below.

```
0x800000 + port
```

The **transport** (p. 367) 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. 1714)

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id (p. 1714)
TransportSupport.set_builtin_transport_property() (p. 1603)

8.231.5 Creating and Registering Shared Memory Transport Plugin

RTI Connext can implicitly create this plugin and register with the **com.rti.dds.domain.DomainParticipant** (p. 629) if this **transport** (p. 367) is specified in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580).

To specify the properties of the builtin shared memory **transport** (p. 367) that is implicitly registered, you can either:

- call **TransportSupport.set_builtin_transport_property** (p. 1603) or
- specify the pre-defined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) associated with the **com.rti.dds.domain.DomainParticipant** (p. 629). (see **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1441)).

Builtin **transport** (p. 367) plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) always overwrite the ones specified through **TransportSupport.set_builtin_transport_property()** (p. 1603). The default value is assumed on any unspecified property. Note that all properties should be set before the **transport** (p. 367) is implicitly created and registered by RTI Connext. See **Built-in Transport Plugins** (p. 216) for details on when a builtin **transport** (p. 367) is registered.

8.231.6 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.infrastructure.PropertyQosPolicy** (p. 1252) of a **com.rti.dds.domain.DomainParticipant** (p. 629) to configure the builtin shared memory **transport** (p. 367) plugin.

Name	Descriptions
dds.transport.shmem.builtin.parent.address_bit_count	See **Transport.Property_t.address_bit_count** (p. 1573)
dds.transport.shmem.builtin.parent.properties_bitmap	See **Transport.Property_t.properties_bitmap** (p. 1574)
dds.transport.shmem.builtin.parent.gather_send_buffer_count_max	See **Transport.Property_t.gather_send_buffer_count_max** (p. 1574)
dds.transport.shmem.builtin.parent.message_size_max	See **Transport.Property_t.message_size_max** (p. 1574)
dds.transport.shmem.builtin.parent.allow_interfaces	See **Transport.Property_t.allow_interfaces_list** (p. 1575) and Transport.Property_t.allow_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_interfaces	See **Transport.Property_t.deny_interfaces_list** (p. 1575) and Transport.Property_t.deny_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.allow_multicast_interfaces	See **Transport.Property_t.allow_multicast_interfaces_list** (p. 1576) and Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_multicast_interfaces	See **Transport.Property_t.deny_multicast_interfaces_list** (p. 1576) and Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.shmem.builtin.received_message_count_max	See **ShmemTransport.Property_t.received_message_count_max** (p. 1443)
dds.transport.shmem.builtin.receive_buffer_size	See **ShmemTransport.Property_t.receive_buffer_size** (p. 1444)

8.232 ShmemTransport.Property_t Class Reference

Subclass of **Transport.Property_t** (p. 1570) allowing specification of parameters that are specific to the shared-memory **transport** (p. 367).

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.

8.232.1 Detailed Description

Subclass of **Transport.Property_t** (p. 1570) allowing specification of parameters that are specific to the shared-memory **transport** (p. 367).

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.232.2 Constructor & Destructor Documentation

8.232.2.1 Property_t ()

Create an empty **ShmemTransport** (p. 1439) property with default values

8.232.3 Member Data Documentation

8.232.3.1 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 **Transport.Property_t.message_size_max** (p. 1574). The total number of bytes that can be buffered for a **transport** (p. 367) plug-in is actually controlled by `receive_buffer_size`.

See also:

NDDS_Transport_Property_t, NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT

8.232.3.2 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** (p. 367) 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** (p. 366), 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

8.233 ShortSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < 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. 1448) that returns a java.lang.Short.*

- Object **set** (int index, Object element)

  *A wrapper for **setShort()** (p. 1449).*

- void **add** (int index, Object element)

  A wrapper for addShort(int, int).

8.233.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < short >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

short
com.rti.dds.util.Sequence (p. 1432)

8.233.2 Constructor & Destructor Documentation

8.233.2.1 ShortSeq ()

Constructs an empty sequence of short integers with an initial maximum of zero.

8.233.2.2 ShortSeq (int *initialMaximum*)

Constructs an empty sequence of short integers with the given initial maximum.

8.233.2.3 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

8.233.3 Member Function Documentation

8.233.3.1 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.

8.233.3.2 boolean addAllShort (short[] *elements*)

Exceptions:

NullPointerException if the given array is null

8.233.3.3 void addShort (short *element*)

Append the element to the end of the sequence.

8.233.3.4 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.

8.233.3.5 short getShort (int *index*)

Returns the short at the given index.

Exceptions:

IndexOutOfBoundsException if the index is out of bounds.

8.233.3.6 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.

8.233.3.7 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.

8.233.3.8 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.

8.233.3.9 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. 1451), or explicitly by calling Sequence.setMaximum.

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.233.3.10 Object get (int *index*) [virtual]

A wrapper for **getShort(int)** (p. 1448) that returns a java.lang.Short.

See also:

java.util.List.get(int)

Implements **AbstractPrimitiveSequence** (p. 377).

8.233.3.11 Object set (int *index*, Object *element*) [virtual]

A wrapper for **setShort()** (p. 1449).

Exceptions:

ClassCastException if the element is not of type Short.

See also:

java.util.List.set(int, java.lang.Object)

Implements **AbstractPrimitiveSequence** (p. 377).

8.233.3.12 void add (int *index*, Object *element*) [virtual]

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)

Implements **AbstractPrimitiveSequence** (p. 378).

8.234 StatusCondition Interface Reference

<<*interface*>> *(p. 271)* A specific **com.rti.dds.infrastructure.Condition** (p. 451) that is associated with each **com.rti.dds.infrastructure.Entity** (p. 912).

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. 912).*

- 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. 1452).*

- **Entity get_entity** ()

  Get the ***com.rti.dds.infrastructure.Entity*** *(p. 912) associated with the* ***com.rti.dds.infrastructure.StatusCondition*** *(p. 1452).*

8.234.1 Detailed Description

<<*interface*>> *(p. 271)* A specific **com.rti.dds.infrastructure.Condition** (p. 451) that is associated with each **com.rti.dds.infrastructure.Entity** (p. 912).

The `trigger_value` of the **com.rti.dds.infrastructure.StatusCondition** (p. 1452) 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. 1452).

See also:

Status Kinds (p. 106)
com.rti.dds.infrastructure.WaitSet (p. 1695),
com.rti.dds.infrastructure.Condition (p. 451)
com.rti.dds.infrastructure.Listener (p. 1154)

8.234.2 Member Function Documentation

8.234.2.1 int get_enabled_statuses ()

Get the list of statuses enabled on an **com.rti.dds.infrastructure.Entity** (p. 912).

Returns:

list of enabled statuses.

8.234.2.2 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. 1452).

This operation may change the `trigger_value` of the **com.rti.dds.infrastructure.StatusCondition** (p. 1452).

com.rti.dds.infrastructure.WaitSet (p. 1695) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **com.rti.dds.infrastructure.WaitSet** (p. 1695) to which the **com.rti.dds.infrastructure.StatusCondition** (p. 1452) 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. 271)* the list of enables statuses (see **Status Kinds** (p. 106))

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.234.2.3 Entity get_entity ()

Get the **com.rti.dds.infrastructure.Entity** (p. 912) associated with the **com.rti.dds.infrastructure.StatusCondition** (p. 1452).

There is exactly one **com.rti.dds.infrastructure.Entity** (p. 912) assocated with each **com.rti.dds.infrastructure.StatusCondition** (p. 1452).

Returns:

com.rti.dds.infrastructure.Entity (p. 912) associated with the **com.rti.dds.infrastructure.StatusCondition** (p. 1452).

8.235 StatusKind Class Reference

Type for *status* kinds.

Static Public Attributes

- static final int **INCONSISTENT_TOPIC_STATUS**

  *Another **topic** (p. 350) 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. 538) has committed through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) was not respected for a specific instance.*

- static final int **REQUESTED_DEADLINE_MISSED_STATUS**

  *The deadline that the **com.rti.dds.subscription.DataReader** (p. 473) was expecting through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) was not respected for a specific instance.*

- static final int **OFFERED_INCOMPATIBLE_QOS_STATUS**

  *A **QosPolicy** (p. 1314) value was incompatible with what was requested.*

- static final int **REQUESTED_INCOMPATIBLE_QOS_STATUS**

  *A **QosPolicy** (p. 1314) 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. 538) has committed to through its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164) was not respected, thus **com.rti.dds.subscription.DataReader** (p. 473) entities*

will consider the ***com.rti.dds.publication.DataWriter*** *(p. 538) as no longer alive.*

^ static final int **LIVELINESS_CHANGED_STATUS**

The liveliness of one or more ***com.rti.dds.publication.DataWriter*** *(p. 538) that were writing instances read through the* ***com.rti.dds.subscription.DataReader*** *(p. 473) has changed. Some* ***com.rti.dds.publication.DataWriter*** *(p. 538) have become alive or not_alive.*

^ static final int **PUBLICATION_MATCHED_STATUS**

The ***com.rti.dds.publication.DataWriter*** *(p. 538) has found* ***com.rti.dds.subscription.DataReader*** *(p. 473) that matches the* ***com.rti.dds.topic.Topic*** *(p. 1545) and has compatible QoS.*

^ static final int **SUBSCRIPTION_MATCHED_STATUS**

The ***com.rti.dds.subscription.DataReader*** *(p. 473) has found* ***com.rti.dds.publication.DataWriter*** *(p. 538) that matches the* ***com.rti.dds.topic.Topic*** *(p. 1545) and has compatible QoS.*

^ static final int **RELIABLE_WRITER_CACHE_CHANGED_STATUS**

<<**eXtension**>> (p. 270) *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. 270) *One or more reliable readers has become active or inactive.*

^ static final int **DDS_DATA_WRITER_CACHE_STATUS**

<<**eXtension**>> (p. 270) *The status of the writer's cache.*

^ static final int **DDS_DATA_WRITER_PROTOCOL_STATUS**

<<**eXtension**>> (p. 270) *The status of a writer's internal protocol related metrics*

^ static final int **DDS_DATA_READER_CACHE_STATUS**

<<**eXtension**>> (p. 270) *The status of the reader's cache.*

^ static final int **DATA_READER_PROTOCOL_STATUS**

<<**eXtension**>> (p. 270) *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.

8.235.1 Detailed Description

Type for *status* kinds.

Each concrete **com.rti.dds.infrastructure.Entity** (p. 912) is associated with a set of ∗`Status` objects whose values represent the communication status of that **com.rti.dds.infrastructure.Entity** (p. 912).

The communication statuses whose changes can be communicated to the application depend on the **com.rti.dds.infrastructure.Entity** (p. 912).

Each status value can be accessed with a corresponding method on the **com.rti.dds.infrastructure.Entity** (p. 912). The changes on these status values cause activation of the corresponding **com.rti.dds.infrastructure.StatusCondition** (p. 1452) objects and trigger invocation of the proper **com.rti.dds.infrastructure.Listener** (p. 1154) objects to asynchronously inform the application.

See also:

com.rti.dds.infrastructure.Entity (p. 912),
com.rti.dds.infrastructure.StatusCondition (p. 1452),
com.rti.dds.infrastructure.Listener (p. 1154)

8.235.2 Member Data Documentation

8.235.2.1 final int INCONSISTENT_TOPIC_STATUS [static]

Another **topic** (p. 350) exists with the same name but different characteristics.

Entity:

com.rti.dds.topic.Topic (p. 1545)

Status:

com.rti.dds.topic.InconsistentTopicStatus (p. 1077)

Listener:

com.rti.dds.topic.TopicListener (p. 1564)

8.235.2.2 final int OFFERED_DEADLINE_MISSED_STATUS [static]

The deadline that the **com.rti.dds.publication.DataWriter** (p. 538) has committed through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) was not respected for a specific instance.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

QoS:

DEADLINE (p. 50)

Status:

com.rti.dds.publication.OfferedDeadlineMissedStatus (p. 1212)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

8.235.2.3 final int REQUESTED_DEADLINE_MISSED_STATUS [static]

The deadline that the **com.rti.dds.subscription.DataReader** (p. 473) was expecting through its **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) was not respected for a specific instance.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

QoS:

DEADLINE (p. 50)

Status:

com.rti.dds.subscription.RequestedDeadlineMissedStatus (p. 1353)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.4 final int OFFERED_INCOMPATIBLE_QOS_STATUS [static]

A **QosPolicy** (p. 1314) value was incompatible with what was requested.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Status:

com.rti.dds.publication.OfferedIncompatibleQosStatus (p. 1214)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

8.235.2.5 final int REQUESTED_INCOMPATIBLE_QOS_STATUS [static]

A **QosPolicy** (p. 1314) value was incompatible with what is offered.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Status:

com.rti.dds.subscription.RequestedIncompatibleQosStatus (p. 1354)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.6 final int SAMPLE_LOST_STATUS [static]

A sample has been lost (i.e. was never received).

Entity:

com.rti.dds.subscription.Subscriber (p. 1478)

Status:

com.rti.dds.subscription.SampleLostStatus (p. 1415)

Listener:

com.rti.dds.subscription.SubscriberListener (p. 1504)

8.235.2.7 final int SAMPLE_REJECTED_STATUS [static]

A (received) sample has been rejected.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

QoS:

RESOURCE_LIMITS (p. 102)

Status:

com.rti.dds.subscription.SampleRejectedStatus (p. 1422)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.8 final int DATA_ON_READERS_STATUS [static]

New data is available.

Entity:

com.rti.dds.subscription.Subscriber (p. 1478)

Listener:

com.rti.dds.subscription.SubscriberListener (p. 1504)

8.235.2.9 final int DATA_AVAILABLE_STATUS [static]

One or more new data samples have been received.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.10 final int LIVELINESS_LOST_STATUS [static]

The liveliness that the **com.rti.dds.publication.DataWriter** (p. 538) has committed to through its **com.rti.dds.infrastructure.LivelinessQosPolicy** (p. 1164) was not respected, thus **com.rti.dds.subscription.DataReader** (p. 473) entities will consider the **com.rti.dds.publication.DataWriter** (p. 538) as no longer alive.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

QoS:

LIVELINESS (p. 78)

Status:

com.rti.dds.publication.LivelinessLostStatus (p. 1162)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

8.235.2.11 final int LIVELINESS_CHANGED_STATUS [static]

The liveliness of one or more **com.rti.dds.publication.DataWriter** (p. 538) that were writing instances read through the **com.rti.dds.subscription.DataReader** (p. 473) has changed. Some **com.rti.dds.publication.DataWriter** (p. 538) have become alive or not_alive.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

QoS:

LIVELINESS (p. 78)

Status:

com.rti.dds.subscription.LivelinessChangedStatus (p. 1159)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.12 final int PUBLICATION_MATCHED_STATUS [static]

The **com.rti.dds.publication.DataWriter** (p. 538) has found **com.rti.dds.subscription.DataReader** (p. 473) that matches the **com.rti.dds.topic.Topic** (p. 1545) and has compatible QoS.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Status:

com.rti.dds.publication.PublicationMatchedStatus (p. 1274)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

8.235.2.13 final int SUBSCRIPTION_MATCHED_STATUS [static]

The **com.rti.dds.subscription.DataReader** (p. 473) has found **com.rti.dds.publication.DataWriter** (p. 538) that matches the **com.rti.dds.topic.Topic** (p. 1545) and has compatible QoS.

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Status:

com.rti.dds.subscription.SubscriptionMatchedStatus (p. 1520)

Listener:

com.rti.dds.subscription.DataReaderListener (p. 501)

8.235.2.14 final int RELIABLE_WRITER_CACHE_CHANGED_STATUS [static]

*<<**eXtension**>>* *(p. 270)* 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 unacknowledge samples), full (i.e. the sample count has reached the value specified in **com.rti.dds.infrastructure.ResourceLimitsQosPolicy.max_samples** (p. 1359)), or the number of samples has reached a high (see **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.high_watermark** (p. 1381)) or low (see **com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t.low_watermark** (p. 1381)) watermark.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Status:

com.rti.dds.publication.ReliableWriterCacheChangedStatus (p. 1345)

Listener:

com.rti.dds.publication.DataWriterListener (p. 566)

8.235.2.15 final int RELIABLE_READER_ACTIVITY_CHANGED_STATUS [static]

<<***eXtension***>> *(p. 270)* 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.RtpsReliableWriterProtocol_t** (p. 1378) and **com.rti.dds.publication.ReliableReaderActivityChangedStatus** (p. 1342).

See also:

com.rti.dds.infrastructure.RtpsReliableWriterProtocol_t (p. 1378)
com.rti.dds.publication.ReliableReaderActivityChangedStatus (p. 1342)

8.235.2.16 final int DDS_DATA_WRITER_CACHE_STATUS [static]

<<***eXtension***>> *(p. 270)* The status of the writer's cache.

8.235.2.17 final int DDS_DATA_WRITER_PROTOCOL_STATUS [static]

<<*eXtension*>> *(*p. 270*)* The status of a writer's internal protocol related metrics

The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

8.235.2.18 final int DDS_DATA_READER_CACHE_STATUS [static]

<<*eXtension*>> *(*p. 270*)* The status of the reader's cache.

8.235.2.19 final int DATA_READER_PROTOCOL_STATUS [static]

<<*eXtension*>> *(*p. 270*)* 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.

8.236 StringDataReader Class Reference

<<***interface***>> *(p. 271)* Instantiates `DataReader` < 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. 473).*

- 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. 473).*

- 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. 1466) the samples that match the criteria specified in the* ***com.rti.dds.subscription.ReadCondition*** *(p. 1326).*

- 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. 1466) except it accesses samples via the* ***com.rti.dds.type.builtin.StringDataReader.take*** *(p. 1466) operation.*

- String **read_next_sample** (**SampleInfo** sample_info)

  Copies the next not-previously-accessed data value from the ***com.rti.dds.subscription.DataReader*** *(p. 473).*

- String **take_next_sample** (**SampleInfo** sample_info)

  Copies the next not-previously-accessed data value from the ***com.rti.dds.subscription.DataReader*** *(p. 473).*

8.236.1 Detailed Description

<<***interface***>> *(p. 271)* Instantiates `DataReader` < String >.

See also:

com.rti.dds.topic.example.FooDataReader
com.rti.dds.subscription.DataReader (p. 473)

8.236.2 Member Function Documentation

8.236.2.1 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read

8.236.2.2 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. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take

8.236.2.3 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. 1466) the samples that match the criteria specified in the **com.rti.dds.subscription.ReadCondition** (p. 1326).

See also:

com.rti.dds.topic.example.FooDataReader.read_w_condition

8.236.2.4 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. 1466) except it accesses samples via the **com.rti.dds.type.builtin.StringDataReader.take** (p. 1466) operation.

See also:

com.rti.dds.topic.example.FooDataReader.take_w_condition

8.236.2.5 String read_next_sample (SampleInfo *sample_info*)

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473).

See also:

com.rti.dds.topic.example.FooDataReader.read_next_sample

8.236.2.6 String take_next_sample (SampleInfo *sample_info*)

Copies the next not-previously-accessed data value from the **com.rti.dds.subscription.DataReader** (p. 473).

See also:

com.rti.dds.topic.example.FooDataReader.take_next_sample

8.237 StringDataWriter Class Reference

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < 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. 1468) except that it also provides the value for the* `source_timestamp`.

8.237.1 Detailed Description

<<*interface*>> *(p. 271)* Instantiates `DataWriter` < String >.

See also:

com.rti.dds.topic.example.FooDataWriter
com.rti.dds.publication.DataWriter (p. 538)

8.237.2 Member Function Documentation

8.237.2.1 void write (String *instance_data*, InstanceHandle_t *handle*)

Modifies the value of a string data instance.

See also:

com.rti.dds.topic.example.FooDataWriter.write

8.237.2.2 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. 1468) except that it also provides the value for the `source_timestamp`.

See also:

com.rti.dds.topic.example.FooDataWriter.write_w_timestamp

8.238 StringSeq Class Reference

Declares IDL `sequence` < 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

8.238.1 Detailed Description

Declares IDL `sequence` < String > .

See also:

com.rti.dds.util.Sequence (p. 1432)

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < String > with value type semantics. **StringSeq** (p. 1470) is a sequence that contains strings.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.util.Sequence (p. 1432)

8.238.2 Constructor & Destructor Documentation

8.238.2.1 StringSeq ()

Constructs an empty sequence of strings with an initial maximum of zero.

8.238.2.2 StringSeq (int *initialMaximum*)

Constructs an empty sequence of strings with the given initial maximum.

8.238.2.3 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

8.238.3 Member Function Documentation

8.238.3.1 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. 466)(java.lang.Object)

Implements **Copyable** (p. 466).

8.238.3.2 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)

8.238.3.3 static void writeStringArray (String[] *value*, CdrObjectOutput *out*, int *length*, int *maxStringLength*) throws IOException [static]

Write array of string up to the specified length

8.239 StringTypeSupport Class Reference

<<*interface*>> *(p. 271)* String type support.

Inheritance diagram for StringTypeSupport::

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 String data type.

- static void **unregister_type** (**DomainParticipant** participant, String type_name)

  Allows an application to unregister the 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 String type.

8.239.1 Detailed Description

<<*interface*>> *(p. 271)* String type support.

8.239.2 Member Function Documentation

8.239.2.1 static void register_type (DomainParticipant *participant*, String *type_name*) [static]

Allows an application to communicate to RTI Connext the existence of the String data type.

By default, The String built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **com.rti.dds.type.builtin.StringTypeSupport.get_type_name** (p. 1475). 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.StringTypeSupport** (p. 1473) with a **com.rti.dds.domain.DomainParticipant** (p. 629) using different values for the type_name.

If `register_type` is called multiple times with the same **com.rti.dds.domain.DomainParticipant** (p. 629) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>> (p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to register the data type String with. Cannot be null.

type_name *<<in>> (p. 271)* the type name under with the data type String is registered with the participant; this type name is used when creating a new **com.rti.dds.topic.Topic** (p. 1545). (See **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_PRECONDITION_NOT_MET or RETCODE_OUT_OF_RESOURCES.

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. 670)

8.239.2.2 static void unregister_type (DomainParticipant *participant*, String *type_name*) [static]

Allows an application to unregister the String data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.

Precondition:

The String type with `type_name` is registered with the participant and all **com.rti.dds.topic.Topic** (p. 1545) 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. 1545) is associated with the type, the operation will fail with RETCODE_ERROR.

Postcondition:

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters:

participant *<<in>>* *(p. 271)* the **com.rti.dds.domain.DomainParticipant** (p. 629) to unregister the data type String from. Cannot be null.

type_name *<<in>>* *(p. 271)* the type name under with the data type 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. 104), RETCODE_BAD_PARAMETER or RETCODE_ERROR

MT Safety:

SAFE.

See also:

com.rti.dds.type.builtin.StringTypeSupport.register_type (p. 1473)

8.239.2.3 static String get_type_name () [static]

Get the default name for the String type.

Can be used for calling **com.rti.dds.type.builtin.StringTypeSupport.register_type** (p. 1473) or creating **com.rti.dds.topic.Topic** (p. 1545).

Returns:

default name for the String type.

See also:

com.rti.dds.type.builtin.StringTypeSupport.register_type (p. 1473)
com.rti.dds.domain.DomainParticipant.create_topic (p. 670)

8.240 StructMember Class Reference

A description of a member of a struct.

Inherits java.io.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.

8.240.1 Detailed Description

A description of a member of a struct.

See also:

TypeCodeFactory.create_struct_tc (p. 1644)

8.240.2 Constructor & Destructor Documentation

8.240.2.1 StructMember (String *name*, boolean *is_pointer*, short *bits*, boolean *is_key*, TypeCode *type*)

Constructs a **StructMember** (p. 1476) object initialized with the given values.

8.240.3 Member Data Documentation

8.240.3.1 String name

The name of the struct member.

Cannot be null.

8.240.3.2 TypeCode type

The type of the struct member.

Cannot be null.

8.240.3.3 boolean is_pointer

Indicates whether the struct member is a pointer or not.

8.240.3.4 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 **TypeCode.NOT_BITFIELD** (p. 1640).

8.240.3.5 boolean is_key

Indicates if the struct member is a key member or not.

8.241 Subscriber Interface Reference

<<***interface***>> *(p. 271)* A subscriber is the object responsible for actually receiving data from a **subscription** (p. 343).

Inheritance diagram for Subscriber::

Public Member Functions

- void **get_default_datareader_qos** (**DataReaderQos** qos)

  *Copies the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values into the provided **com.rti.dds.subscription.DataReaderQos** (p. 518) instance.*

- void **set_default_datareader_qos** (**DataReaderQos** qos)

  *Sets the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values for this subscriber.*

- void **set_default_datareader_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Set the default **com.rti.dds.subscription.DataReaderQos** (p. 518) 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. 473) that will be attached and belong to the **com.rti.dds.subscription.Subscriber** (p. 1478).*

- **DataReader create_datareader_with_profile** (**TopicDescription** topic, String library_name, String profile_name, **DataReaderListener** listener, int mask)

  <<**eXtension**>> (p. 270) *Creates a **com.rti.dds.subscription.DataReader** (p. 473) object using the **com.rti.dds.subscription.DataReaderQos** (p. 518) associated with the input XML QoS profile.*

- void **delete_datareader** (**DataReader** a_datareader)

  *Deletes a **com.rti.dds.subscription.DataReader** (p. 473) that belongs to the **com.rti.dds.subscription.Subscriber** (p. 1478).*

- **DataReader lookup_datareader** (String topic_name)

  *Retrieves an existing **com.rti.dds.subscription.DataReader** (p. 473).*

- 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. 473) 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. 1478).*

- void **notify_datareaders** ()

  *Invokes the operation **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 503) on the **com.rti.dds.subscription.DataReaderListener** (p. 501) objects attached to contained **com.rti.dds.subscription.DataReader** (p. 473) entities with StatusKind.DATA_AVAILABLE_STATUS that is considered changed as described in **Changes in read communication status** (p. 108).*

- void **set_qos** (**SubscriberQos** qos)

  Sets the subscriber QoS.

- void **set_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *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. 270) *Gets the default XML library associated with a **com.rti.dds.subscription.Subscriber** (p. 1478).*

- void **set_default_library** (String library_name)

  <<**eXtension**>> (p. 270) *Sets the default XML library for a **com.rti.dds.subscription.Subscriber** (p. 1478).*

- String **get_default_profile** ()

  <<**eXtension**>> (p. 270) *Gets the default XML profile associated with a **com.rti.dds.subscription.Subscriber** (p. 1478).*

- void **set_default_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Sets the default XML profile for a **com.rti.dds.subscription.Subscriber** (p. 1478).*

- String **get_default_profile_library** ()

  <<**eXtension**>> (p. 270) *Gets the library where the default XML QoS profile is contained for a* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- 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. 473) objects attached to the* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- void **end_access** ()

  Indicates that the application has finished accessing the data samples in ***com.rti.dds.subscription.DataReader*** *(p. 473) objects managed by the* ***com.rti.dds.subscription.Subscriber*** *(p. 1478).*

- void **copy_from_topic_qos** (**DataReaderQos** datareader_qos, **TopicQos** topic_qos)

  Copies the policies in the ***com.rti.dds.topic.TopicQos*** *(p. 1566) to the corresponding policies in the* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518).*

- **DomainParticipant get_participant** ()

  Returns the ***com.rti.dds.domain.DomainParticipant*** *(p. 629) to which the* ***com.rti.dds.subscription.Subscriber*** *(p. 1478) 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. 1478).*

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. 473) with a combination of the default* ***com.rti.dds.subscription.DataReaderQos*** *(p. 518) and the* ***com.rti.dds.topic.TopicQos*** *(p. 1566).*

8.241.1 Detailed Description

<<***interface***>> *(p. 271)* A subscriber is the object responsible for actually receiving data from a **subscription** (p. 343).

QoS:

com.rti.dds.subscription.SubscriberQos (p. 1506)

Status:

StatusKind.DATA_ON_READERS_STATUS

Listener:

com.rti.dds.subscription.SubscriberListener (p. 1504)

A subscriber acts on the behalf of one or several **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473) objects through the operation **get_datareaders()** (p. 1491) and then access the data available through operations on the **com.rti.dds.subscription.DataReader** (p. 473).

The following operations may be called even if the **com.rti.dds.subscription.Subscriber** (p. 1478) is not enabled. Other operations will the value RETCODE_NOT_ENABLED if called on a disabled **com.rti.dds.subscription.Subscriber** (p. 1478):

- The base-class operations **com.rti.dds.subscription.Subscriber.set_qos** (p. 1493), **com.rti.dds.subscription.Subscriber.set_qos_with_profile** (p. 1494), **com.rti.dds.subscription.Subscriber.get_qos** (p. 1495), **com.rti.dds.subscription.Subscriber.set_listener** (p. 1498), **com.rti.dds.subscription.Subscriber.get_listener**

(p. 1498), **com.rti.dds.infrastructure.Entity.enable** (p. 915), **com.rti.dds.infrastructure.Entity.get_statuscondition** (p. 917), **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917)

- ^ **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485), **com.rti.dds.subscription.Subscriber.create_datareader_with_profile** (p. 1487), **com.rti.dds.subscription.Subscriber.delete_datareader** (p. 1489), **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1501), **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1483), **com.rti.dds.subscription.Subscriber.set_default_datareader_qos_with_profile** (p. 1484), **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1482), **com.rti.dds.subscription.Subscriber.set_default_library** (p. 1495), **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1496)

All operations except for the base-class operations **set_qos()** (p. 1493), **set_qos_with_profile()** (p. 1494), **get_qos()** (p. 1495), **set_listener()** (p. 1498), **get_listener()** (p. 1498), **enable()** (p. 915) and **create_datareader()** (p. 1485) may fail with RETCODE_NOT_ENABLED.

See also:

Operations Allowed in Listener Callbacks (p. 1156)

8.241.2 Member Function Documentation

8.241.2.1 void get_default_datareader_qos (DataReaderQos *qos*)

Copies the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values into the provided **com.rti.dds.subscription.DataReaderQos** (p. 518) 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. 1483), or **com.rti.dds.subscription.Subscriber.set_default_datareader_qos_with_profile** (p. 1484), or else, if the call was never made, the default values from is owning **com.rti.dds.domain.DomainParticipant** (p. 629).

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. 1483)

Parameters:

qos *<<inout>>* *(p. 271)* **com.rti.dds.subscription.DataReaderQos** (p. 518) to be filled-up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

Subscriber.DATAREADER_QOS_DEFAULT (p. 190)
com.rti.dds.subscription.Subscriber.create_datareader (p. 1485)

8.241.2.2 void set_default_datareader_qos (DataReaderQos *qos*)

Sets the default **com.rti.dds.subscription.DataReaderQos** (p. 518) values for this subscriber.

This call causes the default values inherited from the owning **com.rti.dds.domain.DomainParticipant** (p. 629) to be overridden.

This default value will be used for newly created **com.rti.dds.subscription.DataReader** (p. 473) if **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) is specified as the `qos` parameter when **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

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. 1483), **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1482) or calling **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485) with **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) as the `qos` parameter.

Parameters:

qos <<*in*>> *(p. 271)* The default **com.rti.dds.subscription.DataReaderQos** (p. 518) to be set to. The special value **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) 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. 1483) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), or or RETCODE_INCONSISTENT_POLICY

8.241.2.3 void set_default_datareader_qos_with_profile (String *library_name*, String *profile_name*)

<<*eXtension*>> *(p. 270)* Set the default **com.rti.dds.subscription.DataReaderQos** (p. 518) 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. 473) if **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) is specified as the `qos` parameter when **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485) is called.

Precondition:

The **com.rti.dds.subscription.DataReaderQos** (p. 518) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with RETCODE_INCONSISTENT_POLICY

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **com.rti.dds.subscription.Subscriber** (p. 1478) while another thread may be simultaneously calling **com.rti.dds.subscription.Subscriber.set_default_datareader_qos** (p. 1483), **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1482) or calling **com.rti.dds.subscription.Subscriber.create_datareader** (p. 1485) with **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) as the `qos` parameter.

Parameters:

library_name *<<in>>* *(p. 271)* 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. 1495)).

profile_name *<<in>>* *(p. 271)* 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. 1496)).

If the input profile cannot be found the method fails with RETCODE_ERROR.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_INCONSISTENT_POLICY

See also:

Subscriber.DATAREADER_QOS_DEFAULT (p. 190)
com.rti.dds.subscription.Subscriber.create_datareader_with_profile (p. 1487)

8.241.2.4 DataReader create_datareader (TopicDescription *topic*, DataReaderQos *qos*, DataReaderListener *listener*, int *mask*)

Creates a **com.rti.dds.subscription.DataReader** (p. 473) that will be attached and belong to the **com.rti.dds.subscription.Subscriber** (p. 1478).

For each application-defined type `Foo`, there is an implied, auto-generated class `com.rti.dds.topic.example.FooDataReader` (an incarnation of FooDataReader) that extends **com.rti.dds.subscription.DataReader** (p. 473) and contains the operations to read data of type `Foo`.

Note that a common application pattern to construct the QoS for the **com.rti.dds.subscription.DataReader** (p. 473) is to:

- Retrieve the QoS policies on the associated **com.rti.dds.topic.Topic** (p. 1545) by means of the **com.rti.dds.topic.Topic.get_qos** (p. 1548) operation.
- Retrieve the default **com.rti.dds.subscription.DataReader** (p. 473) qos by means of the **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1482) operation.

- Combine those two QoS policies (for **example** (p. 349), using **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1500)) and selectively modify policies as desired
- Use the resulting QoS policies to construct the **com.rti.dds.subscription.DataReader** (p. 473).

When a **com.rti.dds.subscription.DataReader** (p. 473) is created, only those transports already registered are available to the **com.rti.dds.subscription.DataReader** (p. 473). See **Built-in Transport Plugins** (p. 216) for details on when a **builtin** (p. 348) transport is registered.

MT Safety:

UNSAFE. If **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) 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. 1483).

Precondition:

If subscriber is enabled, the **topic** (p. 350) must be enabled. If it is not, this operation will fail and no **com.rti.dds.subscription.DataReader** (p. 473) will be created.
The given **com.rti.dds.topic.TopicDescription** (p. 1561) 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 **Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 191), `topic` (p. 350) cannot be **com.rti.dds.topic.MultiTopic** (p. 1208), or else this method will return NULL.

Parameters:

topic *(p. **350**) <<**in**>> (p. 271)* The **com.rti.dds.topic.TopicDescription** (p. 1561) that the **com.rti.dds.subscription.DataReader** (p. 473) will be associated with. Cannot be NULL.

qos *<<**in**>> (p. 271)* The qos of the **com.rti.dds.subscription.DataReader** (p. 473). The special value **Subscriber.DATAREADER_QOS_DEFAULT** (p. 190) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 473) should be created with the default **com.rti.dds.subscription.DataReaderQos** (p. 518) set in the **com.rti.dds.subscription.Subscriber** (p. 1478). If **com.rti.dds.topic.TopicDescription** (p. 1561) is of type **com.rti.dds.topic.Topic** (p. 1545)

or **com.rti.dds.topic.ContentFilteredTopic** (p. 458), the special value **Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 191) can be used to indicate that the **com.rti.dds.subscription.DataReader** (p. 473) should be created with the combination of the default **com.rti.dds.subscription.DataReaderQos** (p. 518) set on the **com.rti.dds.subscription.Subscriber** (p. 1478) and the **com.rti.dds.topic.TopicQos** (p. 1566) (in the case of a **com.rti.dds.topic.ContentFilteredTopic** (p. 458), the **com.rti.dds.topic.TopicQos** (p. 1566) of the related **com.rti.dds.topic.Topic** (p. 1545)). if **Subscriber.DATAREADER_QOS_USE_TOPIC_QOS** (p. 191) is used, `topic` (p. 350) cannot be a **com.rti.dds.topic.MultiTopic** (p. 1208). Cannot be NULL.

listener *<<**in**>>* *(p. 271)* The listener of the **com.rti.dds.subscription.DataReader** (p. 473).

mask *<<**in**>>* *(p. 271)*. Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.subscription.DataReader** (p. 473) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataReader
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
com.rti.dds.subscription.Subscriber.create_datareader_with_profile (p. 1487)
com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1482)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1500)
com.rti.dds.subscription.DataReader.set_listener (p. 482)

8.241.2.5 DataReader create_datareader_with_profile (TopicDescription *topic*, String *library_name*, String *profile_name*, DataReaderListener *listener*, int *mask*)

*<<**eXtension**>>* *(p. 270)* Creates a **com.rti.dds.subscription.DataReader**

(p. 473) object using the **com.rti.dds.subscription.DataReaderQos** (p. 518) associated with the input XML QoS profile.

The **com.rti.dds.subscription.DataReader** (p. 473) will be attached and belong to the **com.rti.dds.subscription.Subscriber** (p. 1478).

For each application-defined type `Foo`, there is an implied, auto-generated class `com.rti.dds.topic.example.FooDataReader` (an incarnation of FooDataReader) that extends **com.rti.dds.subscription.DataReader** (p. 473) and contains the operations to read data of type `Foo`.

When a **com.rti.dds.subscription.DataReader** (p. 473) is created, only those transports already registered are available to the **com.rti.dds.subscription.DataReader** (p. 473). See **Built-in Transport Plugins** (p. 216) for details on when a **builtin** (p. 348) transport is registered.

Precondition:

If subscriber is enabled, the **topic** (p. 350) must be enabled. If it is not, this operation will fail and no **com.rti.dds.subscription.DataReader** (p. 473) will be created.
The given **com.rti.dds.topic.TopicDescription** (p. 1561) 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 *(p. **350**)* *<<in>> (p. 271)* The **com.rti.dds.topic.TopicDescription** (p. 1561) that the **com.rti.dds.subscription.DataReader** (p. 473) will be associated with. Cannot be NULL.

library_name *<<in>> (p. 271)* 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. 1495)).

profile_name *<<in>> (p. 271)* 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. 1496)).

listener *<<in>> (p. 271)* The listener of the **com.rti.dds.subscription.DataReader** (p. 473).

mask *<<in>> (p. 271).* Changes of communication status to be invoked on the listener.

Returns:

A **com.rti.dds.subscription.DataReader** (p. 473) of a derived class specific to the data-type associated with the **com.rti.dds.topic.Topic** (p. 1545) or NULL if an error occurred.

See also:

com.rti.dds.topic.example.FooDataReader
Specifying QoS on entities (p. 96) for information on setting QoS before entity creation
com.rti.dds.subscription.DataReaderQos (p. 518) for rules on consistency among QoS
Subscriber.DATAREADER_QOS_DEFAULT (p. 190)
Subscriber.DATAREADER_QOS_USE_TOPIC_QOS (p. 191)
com.rti.dds.subscription.Subscriber.create_datareader (p. 1485)
com.rti.dds.subscription.Subscriber.get_default_datareader_qos (p. 1482)
com.rti.dds.topic.Topic.set_qos (p. 1547)
com.rti.dds.subscription.Subscriber.copy_from_topic_qos (p. 1500)
com.rti.dds.subscription.DataReader.set_listener (p. 482)

8.241.2.6 void delete_datareader (DataReader *a_datareader*)

Deletes a **com.rti.dds.subscription.DataReader** (p. 473) that belongs to the **com.rti.dds.subscription.Subscriber** (p. 1478).

Precondition:

If the **com.rti.dds.subscription.DataReader** (p. 473) does not belong to the **com.rti.dds.subscription.Subscriber** (p. 1478), or if ther are any existing **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324) objects that are attached to the **com.rti.dds.subscription.DataReader** (p. 473), 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 RETCODE_PRECONDITION_NOT_MET.

Postcondition:

Listener installed on the **com.rti.dds.subscription.DataReader** (p. 473) will not be called after this method completes successfully.

Parameters:

a_datareader *<<in>> (p. 271)* The **com.rti.dds.subscription.DataReader** (p. 473) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET.

8.241.2.7 DataReader lookup_datareader (String *topic_name*)

Retrieves an existing **com.rti.dds.subscription.DataReader** (p. 473).

Use this operation on the built-in **com.rti.dds.subscription.Subscriber** (p. 1478) (**Built-in Topics** (p. 153)) to access the built-in **com.rti.dds.subscription.DataReader** (p. 473) entities for the built-in topics.

The built-in **com.rti.dds.subscription.DataReader** (p. 473) is created when this operation is called on a built-in **topic** (p. 350) for the first time. The built-in **com.rti.dds.subscription.DataReader** (p. 473) is deleted automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) is deleted.

To ensure that **builtin** (p. 348) **com.rti.dds.subscription.DataReader** (p. 473) entities receive all the discovery traffic, it is suggested that you lookup the **builtin** (p. 348) **com.rti.dds.subscription.DataReader** (p. 473) before the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled. Looking up **builtin** (p. 348) **com.rti.dds.subscription.DataReader** (p. 473) may implicitly register **builtin** (p. 348) transports due to creation of **com.rti.dds.subscription.DataReader** (p. 473) (see **Built-in Transport Plugins** (p. 216) for details on when a **builtin** (p. 348) transport is registered). Therefore, if you are want to modify **builtin** (p. 348) transport properties, do so *before* using this operation.

Therefore the suggested sequence when looking up **builtin** (p. 348) DataReaders is:

- Create a disabled **com.rti.dds.domain.DomainParticipant** (p. 629).
- (optional) Modify **builtin** (p. 348) transport properties
- Call **com.rti.dds.domain.DomainParticipant.get_builtin_subscriber()** (p. 684).
- Call **lookup_datareader()** (p. 1490).
- Call **enable()** (p. 915) on the DomainParticipant.

Parameters:

topic_name *<<in>>* *(p. 271)* Name of the **com.rti.dds.topic.TopicDescription** (p. 1561) that the retrieved **com.rti.dds.subscription.DataReader** (p. 473) is attached to. Cannot be NULL.

Returns:

A **com.rti.dds.subscription.DataReader** (p. 473) that belongs to the **com.rti.dds.subscription.Subscriber** (p. 1478) attached to the

com.rti.dds.topic.TopicDescription (p. 1561) with `topic_name`. If no such **com.rti.dds.subscription.DataReader** (p. 473) exists, this operation returns NULL.

The returned **com.rti.dds.subscription.DataReader** (p. 473) may be enabled or disabled.

If more than one **com.rti.dds.subscription.DataReader** (p. 473) is attached to the **com.rti.dds.subscription.Subscriber** (p. 1478), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **com.rti.dds.subscription.DataReader** (p. 473) in one thread while another thread is simultaneously creating or destroying that **com.rti.dds.subscription.DataReader** (p. 473).

8.241.2.8 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. 473) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a **begin_access()** (p. 1499)/end_access() block, or if the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) of the **com.rti.dds.subscription.Subscriber** (p. 1478) is PresentationQosPolicyAccessScopeKind.INSTANCE_PRESENTATION_QOS or PresentationQosPolicyAccessScopeKind.TOPIC_PRESENTATION_QOS, or if the **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) of the **com.rti.dds.subscription.Subscriber** (p. 1478) is false, the returned collection is a 'set' containing each **com.rti.dds.subscription.DataReader** (p. 473) at most once, in no specified order.

If the application is within a **begin_access()** (p. 1499)/end_access() block, and the **PRESENTATION** (p. 86) policy of the **com.rti.dds.subscription.Subscriber** (p. 1478) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS or PresentationQosPolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS, and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) in the **com.rti.dds.subscription.Subscriber** (p. 1478) is true, the returned collection is a 'list' of DataReaders where a **DataReader** (p. 473) 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. 1277) configured with PresentationQosPolicyAccessScopeKind.GROUP_-PRESENTATION_QOS), the application should read()/take() from each **DataReader** (p. 473) in the same order as it appears in the output sequence. The application will move to the next **DataReader** (p. 473) when the read()/take() operation fails with RETCODE_NO_DATA.

Parameters:

readers *<<**inout**>>* *(p. 271)* a **com.rti.dds.subscription.DataReaderSeq** (p. 536) object where the set or list of readers will be returned. Cannot be NULL.

sample_states *<<**in**>>* *(p. 271)* the returned **DataReader** (p. 473) must contain samples that have one of these `sample_states`.

view_states *<<**in**>>* *(p. 271)* the returned **DataReader** (p. 473) must contain samples that have one of these `view_states`.

instance_states *<<**in**>>* *(p. 271)* the returned **DataReader** (p. 473) must contain samples that have one of these `instance_states`.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_-ENABLED.

See also:

Access to data samples (p. 346)
com.rti.dds.subscription.Subscriber.begin_access (p. 1499)
com.rti.dds.subscription.Subscriber.end_access (p. 1500)
PRESENTATION (p. 86)

8.241.2.9 void get_all_datareaders (DataReaderSeq *readers*)

Retrieve all the DataReaders created from this **Subscriber** (p. 1478).

Parameters:

readers *<<**inout**>>* *(p. 271)* Sequence where the DataReaders will be added

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.241.2.10 void notify_datareaders ()

Invokes the operation **com.rti.dds.subscription.DataReaderListener.on_data_available()** (p. 503) on the **com.rti.dds.subscription.DataReaderListener** (p. 501) objects attached to contained **com.rti.dds.subscription.DataReader** (p. 473) entities with StatusKind.DATA_AVAILABLE_STATUS that is considered changed as described in **Changes in read communication status** (p. 108).

This operation is typically invoked from the **com.rti.dds.subscription.SubscriberListener.on_data_on_readers** (p. 1505) operation in the **com.rti.dds.subscription.SubscriberListener** (p. 1504). That way the **com.rti.dds.subscription.SubscriberListener** (p. 1504) can delegate to the **com.rti.dds.subscription.DataReaderListener** (p. 501) objects the handling of the data.

The operation will notify the data readers that have a `sample_state` of **SampleStateKind.NOT_READ_SAMPLE_STATE** (p. 1431), `view_state` of **SampleStateKind.ANY_SAMPLE_STATE** (p. 197) and `instance_state` of **InstanceStateKind.ANY_INSTANCE_STATE** (p. 199).

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_NOT_ENABLED.

8.241.2.11 void set_qos (SubscriberQos *qos*)

Sets the subscriber QoS.

This operation modifies the QoS of the **com.rti.dds.subscription.Subscriber** (p. 1478).

The **com.rti.dds.subscription.SubscriberQos.group_data** (p. 1507), **com.rti.dds.subscription.SubscriberQos.partition** (p. 1507) and **com.rti.dds.subscription.SubscriberQos.entity_factory** (p. 1507) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 271)* **com.rti.dds.subscription.SubscriberQos** (p. 1506) to be set to. Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.subscription.Subscriber** (p. 1478) is enabled. The special value **DomainParticipant.SUBSCRIBER_QOS_DEFAULT** (p. 149) can be used to indicate that the QoS of the **com.rti.dds.subscription.Subscriber**

(p. 1478) should be changed to match the current default **com.rti.dds.subscription.SubscriberQos** (p. 1506) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.subscription.SubscriberQos (p. 1506) for rules on consistency among QoS
set_qos (abstract) (p. 913)
Operations Allowed in Listener Callbacks (p. 1156)

8.241.2.12 void set_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> (p. 270) 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. 1478).

The **com.rti.dds.subscription.SubscriberQos.group_data** (p. 1507), **com.rti.dds.subscription.SubscriberQos.partition** (p. 1507) and **com.rti.dds.subscription.SubscriberQos.entity_factory** (p. 1507) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>> (p. 271)* 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. 1495)).

profile_name *<<in>> (p. 271)* 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. 1496)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY, or RETCODE_INCONSISTENT_POLICY.

See also:

com.rti.dds.subscription.SubscriberQos (p. 1506) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1156)

8.241.2.13 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. 271)* **com.rti.dds.subscription.SubscriberQos** (p. 1506) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.241.2.14 String get_default_library ()

<<eXtension>> *(p. 270)* Gets the default XML library associated with a **com.rti.dds.subscription.Subscriber** (p. 1478).

Returns:

The default library or null if the default library was not set.

See also:

com.rti.dds.subscription.Subscriber.set_default_library (p. 1495)

8.241.2.15 void set_default_library (String *library_name*)

<<eXtension>> *(p. 270)* Sets the default XML library for a **com.rti.dds.subscription.Subscriber** (p. 1478).

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'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. 1478) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 629) (see **com.rti.dds.domain.DomainParticipant.set_default_library** (p. 679)).

Parameters:

library_name *<<**in**>>* *(p. 271)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.subscription.Subscriber.get_default_library (p. 1495)

8.241.2.16 String get_default_profile ()

*<<**eXtension**>>* *(p. 270)* Gets the default XML profile associated with a **com.rti.dds.subscription.Subscriber** (p. 1478).

Returns:

The default profile or null if the default profile was not set.

See also:

com.rti.dds.subscription.Subscriber.set_default_profile (p. 1496)

8.241.2.17 void set_default_profile (String *library_name*, String *profile_name*)

*<<**eXtension**>>* *(p. 270)* Sets the default XML profile for a **com.rti.dds.subscription.Subscriber** (p. 1478).

This method specifies the profile that will be used as the default the next time a default **Subscriber** (p. 1478) profile is needed during a call to one of this Subscriber's operations. When calling a **com.rti.dds.subscription.Subscriber** (p. 1478) 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. 1478) inherits the default from the **com.rti.dds.domain.DomainParticipant** (p. 629) (see **com.rti.dds.domain.DomainParticipant.set_default_profile** (p. 680)).

This method does not set the default QoS for **com.rti.dds.subscription.DataReader** (p. 473) objects created by this **com.rti.dds.subscription.Subscriber** (p. 1478); for this functionality, use **com.rti.dds.subscription.Subscriber.set_default_datareader_qos_with_profile** (p. 1484) (you may pass in NULL after having called **set_default_profile()** (p. 1496)).

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. 650).

Parameters:

library_name *<<in>> (p. 271)* The library name containing the profile.

profile_name *<<in>> (p. 271)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.subscription.Subscriber.get_default_profile (p. 1496)
com.rti.dds.subscription.Subscriber.get_default_profile_library (p. 1497)

8.241.2.18 String get_default_profile_library ()

<<eXtension>> (p. 270) Gets the library where the default XML QoS profile is contained for a **com.rti.dds.subscription.Subscriber** (p. 1478).

The default profile library is automatically set when **com.rti.dds.subscription.Subscriber.set_default_profile** (p. 1496) is called.

This library can be different than the **com.rti.dds.subscription.Subscriber** (p. 1478) default library (see **com.rti.dds.subscription.Subscriber.get_default_library** (p. 1495)).

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. 1496)

8.241.2.19 void set_listener (SubscriberListener *l*, int *mask*)

Sets the subscriber listener.

Parameters:

l *<<in>>* *(p. 271)* **com.rti.dds.subscription.SubscriberListener** (p. 1504) to set to.

mask *<<in>>* *(p. 271)* com.rti.dds.infrastructure.StatusMask associated with the **com.rti.dds.subscription.SubscriberListener** (p. 1504).

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.241.2.20 SubscriberListener get_listener ()

Get the subscriber listener.

Returns:

com.rti.dds.subscription.SubscriberListener (p. 1504) of the **com.rti.dds.subscription.Subscriber** (p. 1478).

See also:

get_listener (abstract) (p. 915)

8.241.2.21 void call_listenerT (int *mask*)

Call the subscriber listener.

8.241.2.22 void begin_access ()

Indicates that the application is about to access the data samples in any of the **com.rti.dds.subscription.DataReader** (p. 473) objects attached to the **com.rti.dds.subscription.Subscriber** (p. 1478).

If the **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) of the **com.rti.dds.subscription.Subscriber** (p. 1478) is PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS or PresentationQosPolicyAccessScopeKind.HIGHEST_OFFERED_PRESENTATION_QOS and **com.rti.dds.infrastructure.PresentationQosPolicy.ordered_access** (p. 1241) is 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. 1277) with **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) set to PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS).

In the above case, the operation **begin_access()** (p. 1499) must be called prior to calling any of the sample-accessing operations, **com.rti.dds.subscription.Subscriber.get_datareaders** (p. 1491) on the **com.rti.dds.subscription.Subscriber** (p. 1478), and com.rti.dds.topic.example.FooDataReader.read, com.rti.dds.topic.example.FooDataReader.take, com.rti.dds.topic.example.FooDataReader.read_w_condition, and com.rti.dds.topic.example.FooDataReader.take_w_condition on any **com.rti.dds.subscription.DataReader** (p. 473).

Once the application has finished accessing the data samples, it must call **com.rti.dds.subscription.Subscriber.end_access** (p. 1500).

The application is not required to call **begin_access()** (p. 1499) / **end_access()** (p. 1500) to access the samples in order if the **PRESENTATION** (p. 86) policy in the **com.rti.dds.publication.Publisher** (p. 1277) has **com.rti.dds.infrastructure.PresentationQosPolicy.access_scope** (p. 1241) set to something other than PresentationQosPolicyAccessScopeKind.GROUP_PRESENTATION_QOS. In this case, calling **begin_access()** (p. 1499) / **end_access()** (p. 1500) is not considered an error and has no effect.

Calls to **begin_access()** (p. 1499) / **end_access()** (p. 1500) may be nested and must be balanced.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_NOT_ENABLED.

See also:

Access to data samples (p. 346)

com.rti.dds.subscription.Subscriber.get_datareaders (p. 1491)
PRESENTATION (p. 86)

8.241.2.23 void end_access ()

Indicates that the application has finished accessing the data samples in **com.rti.dds.subscription.DataReader** (p. 473) objects managed by the **com.rti.dds.subscription.Subscriber** (p. 1478).

This operation must be used to close a corresponding **begin_access()** (p. 1499).

This call must close a previous call to **com.rti.dds.subscription.Subscriber.begin_access()** (p. 1499), otherwise the operation will fail with the error RETCODE_PRECONDITION_NOT_MET.

Exceptions:

One of the **Standard Return Codes** (p. 104) or RETCODE_PRECONDITION_NOT_MET or RETCODE_NOT_ENABLED.

8.241.2.24 void copy_from_topic_qos (DataReaderQos *datareader_qos*, TopicQos *topic_qos*)

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1566) to the corresponding policies in the **com.rti.dds.subscription.DataReaderQos** (p. 518).

Copies the policies in the **com.rti.dds.topic.TopicQos** (p. 1566) to the corresponding policies in the **com.rti.dds.subscription.DataReaderQos** (p. 518) (replacing values in the **com.rti.dds.subscription.DataReaderQos** (p. 518), if present).

This is a "convenience" operation most useful in combination with the operations **com.rti.dds.subscription.Subscriber.get_default_datareader_qos** (p. 1482) and **com.rti.dds.topic.Topic.get_qos** (p. 1548). The operation **com.rti.dds.subscription.Subscriber.copy_from_topic_qos** (p. 1500) can be used to merge the **com.rti.dds.subscription.DataReader** (p. 473) default QoS policies with the corresponding ones on the **com.rti.dds.topic.Topic** (p. 1545). The resulting QoS can then be used to create a new **com.rti.dds.subscription.DataReader** (p. 473), or set its QoS.

This operation does not check the resulting **com.rti.dds.subscription.DataReaderQos** (p. 518) for consistency. This is because the 'merged' **com.rti.dds.subscription.DataReaderQos** (p. 518) 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. 473).

Parameters:

datareader_qos *<<inout>>* *(p. 271)* **com.rti.dds.subscription.DataReaderQos** (p. 518) to be filled-up. Cannot be NULL.

topic_qos *<<in>>* *(p. 271)* **com.rti.dds.topic.TopicQos** (p. 1566) to be merged with **com.rti.dds.subscription.DataReaderQos** (p. 518). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.241.2.25 DomainParticipant get_participant ()

Returns the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.subscription.Subscriber** (p. 1478) belongs.

Returns:

the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.subscription.Subscriber** (p. 1478) belongs.

8.241.2.26 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. 1478).

Deletes all contained **com.rti.dds.subscription.DataReader** (p. 473) objects. This pattern is applied recursively. In this manner, the operation **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1501) on the **com.rti.dds.subscription.Subscriber** (p. 1478) will end up deleting all the entities recursively contained in the **com.rti.dds.subscription.Subscriber** (p. 1478), that is also the **com.rti.dds.subscription.QueryCondition** (p. 1324) and **com.rti.dds.subscription.ReadCondition** (p. 1326) objects belonging to the contained **com.rti.dds.subscription.DataReader** (p. 473).

The operation will fail with RETCODE_PRECONDITION_NOT_MET if any of the contained entities is in a state where it cannot be deleted. This will occur, for **example** (p. 349), if a contained **com.rti.dds.subscription.DataReader** (p. 473) cannot be deleted because the application has called a com.rti.dds.topic.example.FooDataReader.read or com.rti.dds.topic.example.FooDataReader.take operation and has not called the corresponding com.rti.dds.topic.example.FooDataReader.return_loan operation to return the loaned samples.

Once **com.rti.dds.subscription.Subscriber.delete_contained_entities** (p. 1501) completes successfully, the application may delete the **com.rti.dds.subscription.Subscriber** (p. 1478), knowing that it has no contained **com.rti.dds.subscription.DataReader** (p. 473) objects.

Exceptions:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET

8.242 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 StatusKind.DATA_ON_READERS_STATUS communication status.

8.242.1 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.

8.242.2 Member Function Documentation

8.242.2.1 void on_data_on_readers (Subscriber *subs*)

Handles the StatusKind.DATA_ON_READERS_STATUS communication status.

Implements **SubscriberListener** (p. 1505).

8.243 SubscriberListener Interface Reference

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for status about a subscriber.

Inheritance diagram for SubscriberListener::

Public Member Functions

- void **on_data_on_readers** (**Subscriber** subs)

  Handles the StatusKind.DATA_ON_READERS_STATUS communication status.

8.243.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for status about a subscriber.

Entity:

com.rti.dds.subscription.Subscriber (p. 1478)

Status:

StatusKind.DATA_AVAILABLE_STATUS;
StatusKind.DATA_ON_READERS_STATUS;
StatusKind.LIVELINESS_CHANGED_STATUS,
com.rti.dds.subscription.LivelinessChangedStatus (p. 1159);
StatusKind.REQUESTED_DEADLINE_MISSED_STATUS,
com.rti.dds.subscription.RequestedDeadlineMissedStatus (p. 1353);
StatusKind.REQUESTED_INCOMPATIBLE_QOS_STATUS,
com.rti.dds.subscription.RequestedIncompatibleQosStatus (p. 1354);
StatusKind.SAMPLE_LOST_STATUS_STATUS,
com.rti.dds.subscription.SampleLostStatus (p. 1415);
StatusKind.SAMPLE_REJECTED_STATUS,
com.rti.dds.subscription.SampleRejectedStatus (p. 1422);
StatusKind.SUBSCRIPTION_MATCHED_STATUS,
com.rti.dds.subscription.SubscriptionMatchedStatus (p. 1520)

See also:

com.rti.dds.infrastructure.Listener (p. 1154)

Status Kinds (p. 106)
Operations Allowed in Listener Callbacks (p. 1156)

8.243.2 Member Function Documentation

8.243.2.1 void on_data_on_readers (Subscriber *subs*)

Handles the StatusKind.DATA_ON_READERS_STATUS communication status.

Implemented in **SubscriberAdapter** (p. 1503).

8.244 SubscriberQos Class Reference

QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1478) entity.

Inheritance diagram for SubscriberQos::

Public Attributes

- final **PresentationQosPolicy presentation**
  Presentation policy, ***PRESENTATION*** *(p. 86).*
- final **PartitionQosPolicy partition**
  Partition policy, ***PARTITION*** *(p. 85).*
- final **GroupDataQosPolicy group_data**
  Group data policy, ***GROUP_DATA*** *(p. 73).*
- final **EntityFactoryQosPolicy entity_factory**
  Entity factory policy, ***ENTITY_FACTORY*** *(p. 69).*
- final **ExclusiveAreaQosPolicy exclusive_area**
  <<**eXtension**>> (p. 270) *Exclusive area for the subscriber and all entities that are created by the subscriber.*

8.244.1 Detailed Description

QoS policies supported by a **com.rti.dds.subscription.Subscriber** (p. 1478) entity.

You must set certain members in a consistent manner:

length of com.rti.dds.subscription.SubscriberQos.group_data.value <= com.rti.dds.domain.DomainParticipantQos.resource_limits.subscriber_group_data_max_length

length of com.rti.dds.subscription.SubscriberQos.partition.name <= com.rti.dds.domain.DomainParticipantQos.resource_limits.max_partitions

combined number of characters (including terminating 0) in com.rti.dds.subscription.SubscriberQos.partition.name <= com.rti.dds.domain.DomainParticipantQos.resource_limits.max_partition_cumulative_characters

If any of the above are not true, **com.rti.dds.subscription.Subscriber.set_qos** (p. 1493) and **com.rti.dds.subscription.Subscriber.set_qos_with_profile** (p. 1494) will fail with RETCODE_INCONSISTENT_POLICY

8.244.2 Member Data Documentation

8.244.2.1 final PresentationQosPolicy presentation

Presentation policy, **PRESENTATION** (p. 86).

8.244.2.2 final PartitionQosPolicy partition

Partition policy, **PARTITION** (p. 85).

8.244.2.3 final GroupDataQosPolicy group_data

Group data policy, **GROUP_DATA** (p. 73).

8.244.2.4 final EntityFactoryQosPolicy entity_factory

Entity factory policy, **ENTITY_FACTORY** (p. 69).

8.244.2.5 final ExclusiveAreaQosPolicy exclusive_area

<<***eXtension***>> *(p. 270)* Exclusive area for the subscriber and all entities that are created by the subscriber.

8.245 SubscriberSeq Class Reference

Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1478) > .

Inherits AbstractNativeSequence.

Public Member Functions

- ^ **SubscriberSeq** (Collection subscribers)
- ^ int **getMaximum** ()

  Get the current maximum number of elements that can be stored in this sequence.

8.245.1 Detailed Description

Declares IDL `sequence` < **com.rti.dds.subscription.Subscriber** (p. 1478) > .

See also:

com.rti.dds.util.Sequence (p. 1432)

8.245.2 Constructor & Destructor Documentation

8.245.2.1 SubscriberSeq (Collection *subscribers*)

Exceptions:

NullPointerException if the given collection is null

8.245.3 Member Function Documentation

8.245.3.1 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. 383), or explicitly by calling **Sequence.setMaximum** (p. 1433).

Returns:

the current maximum of the sequence.

See also:

Sequence.size()

Implements **Sequence** (p. 1433).

8.246 SubscriptionBuiltinTopicData Class Reference

Entry created when a **com.rti.dds.subscription.DataReader** (p. 473) is discovered in association with its **Subscriber** (p. 1478).

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. 473) belongs.*

- String **topic_name**

  *Name of the related **com.rti.dds.topic.Topic** (p. 1545).*

- String **type_name**

  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).*

- final **DurabilityQosPolicy durability**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **OwnershipQosPolicy ownership**

  *Policy of the corresponding **DataReader** (p. 473).*

- final **DestinationOrderQosPolicy destination_order**

  *Policy of the corresponding **DataReader** (p. 473).*

^ final **UserDataQosPolicy user_data**

*Policy of the corresponding **DataReader** (p. 473).*

^ final **TimeBasedFilterQosPolicy time_based_filter**

*Policy of the corresponding **DataReader** (p. 473).*

^ final **PresentationQosPolicy presentation**

*Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.*

^ final **PartitionQosPolicy partition**

*Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.*

^ final **TopicDataQosPolicy topic_data**

Policy of the related Topic.

^ final **GroupDataQosPolicy group_data**

*Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.*

^ **TypeCode type_code**

<<**eXtension**>> (p. 270) *Type code information of the corresponding Topic*

^ final **BuiltinTopicKey_t subscriber_key**

<<**eXtension**>> (p. 270) *DCPS key of the subscriber to which the **DataReader** (p. 473) belongs.*

^ final **PropertyQosPolicy property**

<<**eXtension**>> (p. 270) *Properties of the corresponding **DataReader** (p. 473).*

^ final **LocatorSeq unicast_locators**

<<**eXtension**>> (p. 270) *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. 270) *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. 270) *This field provides all the required information to enable content filtering on the Writer side.*

- final **GUID_t virtual_guid**

  <<**eXtension**>> (p. 270) *Virtual GUID associated to the **DataReader** (p. 473).*

- final **ProtocolVersion_t rtps_protocol_version**

  <<**eXtension**>> (p. 270) *Version number of the RTPS wire protocol used.*

- final **VendorId_t rtps_vendor_id**

  <<**eXtension**>> (p. 270) *ID of vendor implementing the RTPS wire protocol.*

- final **ProductVersion_t product_version**

  <<**eXtension**>> (p. 270) *This is a vendor specific parameter. It gives the current version for rti-dds.*

- boolean **disable_positive_acks**

  <<**eXtension**>> (p. 270) *This is a vendor specific parameter. Determines whether the corresponding **DataReader** (p. 473) sends positive acknowledgements for reliability.*

- final **EntityNameQosPolicy subscription_name**

  <<**eXtension**>> (p. 270) *The **subscription** (p. 343) name and role name.*

8.246.1 Detailed Description

Entry created when a **com.rti.dds.subscription.DataReader** (p. 473) is discovered in association with its **Subscriber** (p. 1478).

Data associated with the built-in **topic** (p. 350) **SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME** (p. 1519). It contains QoS policies and additional information that apply to the remote **com.rti.dds.subscription.DataReader** (p. 473) the related **com.rti.dds.subscription.Subscriber** (p. 1478).

See also:

SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME (p. 1519)
builtin.SubscriptionBuiltinTopicDataDataReader (p. 1517)

8.246.2 Member Data Documentation

8.246.2.1 final BuiltinTopicKey_t key

DCPS key to distinguish entries.

8.246.2.2 final BuiltinTopicKey_t participant_key

DCPS key of the participant to which the **DataReader** (p. 473) belongs.

8.246.2.3 String topic_name

Name of the related **com.rti.dds.topic.Topic** (p. 1545).

The length of this string is limited to 255 characters.

8.246.2.4 String type_name

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).

The length of this string is limited to 255 characters.

8.246.2.5 final DurabilityQosPolicy durability

Policy of the corresponding **DataReader** (p. 473).

8.246.2.6 final DeadlineQosPolicy deadline

Policy of the corresponding **DataReader** (p. 473).

8.246.2.7 final LatencyBudgetQosPolicy latency_budget

Policy of the corresponding **DataReader** (p. 473).

8.246.2.8 final LivelinessQosPolicy liveliness

Policy of the corresponding **DataReader** (p. 473).

8.246.2.9 final ReliabilityQosPolicy reliability

Policy of the corresponding **DataReader** (p. 473).

8.246.2.10 final OwnershipQosPolicy ownership

Policy of the corresponding **DataReader** (p. 473).

8.246.2.11 final DestinationOrderQosPolicy destination_order

Policy of the corresponding **DataReader** (p. 473).

8.246.2.12 final UserDataQosPolicy user_data

Policy of the corresponding **DataReader** (p. 473).

8.246.2.13 final TimeBasedFilterQosPolicy time_based_filter

Policy of the corresponding **DataReader** (p. 473).

8.246.2.14 final PresentationQosPolicy presentation

Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.

8.246.2.15 final PartitionQosPolicy partition

Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.

8.246.2.16 final TopicDataQosPolicy topic_data

Policy of the related Topic.

8.246.2.17 final GroupDataQosPolicy group_data

Policy of the **Subscriber** (p. 1478) to which the **DataReader** (p. 473) belongs.

8.246.2.18 TypeCode type_code

<<***eXtension***>> *(p. 270)* Type code information of the corresponding Topic

8.246.2.19 final BuiltinTopicKey_t subscriber_key

<<***eXtension***>> *(p. 270)* DCPS key of the subscriber to which the **DataReader** (p. 473) belongs.

8.246.2.20 final PropertyQosPolicy property

<<*eXtension*>> *(p. 270)* Properties of the corresponding **DataReader** (p. 473).

8.246.2.21 final LocatorSeq unicast_locators

<<*eXtension*>> *(p. 270)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.246.2.22 final LocatorSeq multicast_locators

<<*eXtension*>> *(p. 270)* Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

8.246.2.23 final ContentFilterProperty_t content_filter_property

<<*eXtension*>> *(p. 270)* This field provides all the required information to enable content filtering on the Writer side.

8.246.2.24 final GUID_t virtual_guid

<<*eXtension*>> *(p. 270)* Virtual GUID associated to the **DataReader** (p. 473).

See also:

com.rti.dds.infrastructure.GUID_t (p. 1069)

8.246.2.25 final ProtocolVersion_t rtps_protocol_version

<<*eXtension*>> *(p. 270)* Version number of the RTPS wire protocol used.

8.246.2.26 final VendorId_t rtps_vendor_id

<<*eXtension*>> *(p. 270)* ID of vendor implementing the RTPS wire protocol.

8.246.2.27 final ProductVersion_t product_version

<<*eXtension*>> *(p. 270)* This is a vendor specific parameter. It gives the current version for rti-dds.

8.246.2.28 boolean disable_positive_acks

<<*eXtension*>> *(p. 270)* This is a vendor specific parameter. Determines whether the corresponding **DataReader** (p. 473) sends positive acknowledgements for reliability.

8.246.2.29 final EntityNameQosPolicy subscription_name

<<*eXtension*>> *(p. 270)* The **subscription** (p. 343) name and role name.

This member contains the name and the role name of the discovered **subscription** (p. 343).

8.247 SubscriptionBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` (p. 473) < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

Inherits DataReaderImpl.

8.247.1 Detailed Description

Instantiates `DataReader` (p. 473) < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

com.rti.dds.subscription.DataReader (p. 473) of **topic** (p. 350) **SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME** (p. 1519) used for accessing **builtin.SubscriptionBuiltinTopicData** (p. 1510) of the remote **com.rti.dds.subscription.DataReader** (p. 473) and the associated **com.rti.dds.subscription.Subscriber** (p. 1478).

Instantiates:

<<***generic***>> *(p. 271)* com.rti.dds.topic.example.FooDataReader

See also:

builtin.SubscriptionBuiltinTopicData (p. 1510)
SubscriptionBuiltinTopicDataTypeSupport.SUBSCRIPTION_TOPIC_NAME (p. 1519)

8.248 SubscriptionBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

Inherits AbstractBuiltinTopicDataSeq.

8.248.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

Instantiates:

<<***generic***>> *(*p. *271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

builtin.SubscriptionBuiltinTopicData (p. 1510)

8.249 SubscriptionBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

Inheritance diagram for SubscriptionBuiltinTopicDataTypeSupport::

Static Public Attributes

- static final String **SUBSCRIPTION_TOPIC_NAME** = DDS_SUBSCRIPTION_TOPIC_NAME()

  *Subscription **topic** (p. 350) name.*

8.249.1 Detailed Description

Instantiates `TypeSupport` < **builtin.SubscriptionBuiltinTopicData** (p. 1510) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

See also:

builtin.SubscriptionBuiltinTopicData (p. 1510)

8.249.2 Member Data Documentation

8.249.2.1 final String SUBSCRIPTION_TOPIC_NAME = DDS_SUBSCRIPTION_TOPIC_NAME() [static]

Subscription **topic** (p. 350) name.

Topic name of **builtin.SubscriptionBuiltinTopicDataDataReader** (p. 1517)

See also:

builtin.SubscriptionBuiltinTopicData (p. 1510)
builtin.SubscriptionBuiltinTopicDataDataReader (p. 1517)

8.250 SubscriptionMatchedStatus Class Reference

StatusKind.SUBSCRIPTION_MATCHED_STATUS.

Inherits Status.

Public Attributes

- int **total_count**

  *The total cumulative number of times the concerned **com.rti.dds.subscription.DataReader** (p. 473) discovered a "match" with a **com.rti.dds.publication.DataWriter** (p. 538).*

- int **total_count_change**

  The change in total_count since the last time the listener was called or the status was read.

- int **current_count**

  *The current number of writers with which the **com.rti.dds.subscription.DataReader** (p. 473) is matched.*

- int **current_count_peak**

  <<**eXtension**>> (p. 270) *The highest value that current_count has reached until now.*

- 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_publication_handle**

  *A handle to the last **com.rti.dds.publication.DataWriter** (p. 538) that caused the status to change.*

8.250.1 Detailed Description

StatusKind.SUBSCRIPTION_MATCHED_STATUS.

A "match" happens when the **com.rti.dds.subscription.DataReader** (p. 473) finds a **com.rti.dds.publication.DataWriter** (p. 538) for the same **com.rti.dds.topic.Topic** (p. 1545) with an offered QoS that is compatible with that requested by the **com.rti.dds.subscription.DataReader** (p. 473).

This status is also changed (and the listener, if any, called) when a match is ended. A local **com.rti.dds.subscription.DataReader** (p. 473) will become "unmatched" from a remote **com.rti.dds.publication.DataWriter** (p. 538) when that **com.rti.dds.publication.DataWriter** (p. 538) goes away for any reason.

8.250.2 Member Data Documentation

8.250.2.1 int total_count

The total cumulative number of times the concerned **com.rti.dds.subscription.DataReader** (p. 473) discovered a "match" with a **com.rti.dds.publication.DataWriter** (p. 538).

This number increases whenever a new match is discovered. It does not change when an existing match goes away.

8.250.2.2 int total_count_change

The change in total_count since the last time the listener was called or the status was read.

8.250.2.3 int current_count

The current number of writers with which the **com.rti.dds.subscription.DataReader** (p. 473) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away.

8.250.2.4 int current_count_peak

<<***eXtension***>> *(p. 270)* The highest value that current_count has reached until now.

8.250.2.5 int current_count_change

The change in current_count since the last time the listener was called or the status was read.

8.250.2.6 final InstanceHandle_t last_publication_handle

A handle to the last **com.rti.dds.publication.DataWriter** (p. 538) that caused the status to change.

8.251 SystemException Class Reference

System exception.

Inheritance diagram for SystemException::

8.251.1 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`

8.252 SystemResourceLimitsQosPolicy Class Reference

Configures **com.rti.dds.domain.DomainParticipant** (p. 629)-independent resources used by RTI Connext. Mainly used to change the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 708).*

8.252.1 Detailed Description

Configures **com.rti.dds.domain.DomainParticipant** (p. 629)-independent resources used by RTI Connext. Mainly used to change the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 629) entities that can be created within a single process (address space).

This QoS policy is an extension to the DDS standard.

Entity:

com.rti.dds.domain.DomainParticipantFactory (p. 708)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.252.2 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. 629) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **com.rti.dds.domain.DomainParticipant** (p. 629) entities that can be created in a single process/address space.

8.252.3 Member Data Documentation

8.252.3.1 int max_objects_per_thread

The maximum number of objects that can be stored per thread for a **com.rti.dds.domain.DomainParticipantFactory** (p. 708).

Before increasing this value to allow you to create more participants, carefully consider the application design that requires you to create so many participants. Remember: a **com.rti.dds.domain.DomainParticipant** (p. 629) is a heavy-weight object. It spawns several threads and maintains its own discovery database (see **DISCOVERY** (p. 54)). Creating more participants than RTI Connext strictly requires – one per **domain** (p. 317) per process/address space – can adversely affect the performance and resource utilization of your application.

[**default**] 1024; this value allows you to create about 10 or 11 **com.rti.dds.domain.DomainParticipant** (p. 629) entities.

[**range**] [1, 1 billion]

8.253 TCKind Class Reference

Enumeration type for **TypeCode** (p. 1611) 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.

ˆ static final **TCKind TK_UNION**

union type.

ˆ static final **TCKind TK_ENUM**

enumerated type.

ˆ static final **TCKind TK_STRING**

string type.

ˆ static final **TCKind TK_SEQUENCE**

sequence type.

ˆ static final **TCKind TK_ARRAY**

array type.

ˆ static final **TCKind TK_ALIAS**

alias (typedef) type.

ˆ static final **TCKind TK_LONGLONG**

long long type.

ˆ static final **TCKind TK_ULONGLONG**

unsigned long long type.

ˆ static final **TCKind TK_LONGDOUBLE**

long double type.

ˆ static final **TCKind TK_WCHAR**

wide char type.

ˆ static final **TCKind TK_WSTRING**

wide string type.

ˆ static final **TCKind TK_VALUE**

value type.

ˆ static final **TCKind TK_SPARSE**

A sparse value type.

8.253.1 Detailed Description

Enumeration type for **TypeCode** (p. 1611) kinds.

Type code kinds are modeled as values of this type.

8.253.2 Member Data Documentation

8.253.2.1 final TCKind TK_NULL [static]

Indicates that a type code does not describe anything.

8.253.2.2 final TCKind TK_SHORT [static]

short type.

8.253.2.3 final TCKind TK_LONG [static]

long type.

8.253.2.4 final TCKind TK_USHORT [static]

unsigned short type.

8.253.2.5 final TCKind TK_ULONG [static]

unsigned long type.

8.253.2.6 final TCKind TK_FLOAT [static]

float type.

8.253.2.7 final TCKind TK_DOUBLE [static]

double type.

8.253.2.8 final TCKind TK_BOOLEAN [static]

boolean type.

8.253.2.9 final TCKind TK_CHAR [static]

char type.

8.253.2.10 final TCKind TK_OCTET [static]

octet type.

8.253.2.11 final TCKind TK_STRUCT [static]

struct type.

8.253.2.12 final TCKind TK_UNION [static]

union type.

8.253.2.13 final TCKind TK_ENUM [static]

enumerated type.

8.253.2.14 final TCKind TK_STRING [static]

string type.

8.253.2.15 final TCKind TK_SEQUENCE [static]

sequence type.

8.253.2.16 final TCKind TK_ARRAY [static]

array type.

8.253.2.17 final TCKind TK_ALIAS [static]

alias (typedef) type.

8.253.2.18 final TCKind TK_LONGLONG [static]

long long type.

8.253.2.19 final TCKind TK_ULONGLONG [static]

unsigned long long type.

8.253.2.20 final TCKind TK_LONGDOUBLE [static]

long double type.

8.253.2.21 final TCKind TK_WCHAR [static]

wide char type.

8.253.2.22 final TCKind TK_WSTRING [static]

wide string type.

8.253.2.23 final TCKind TK_VALUE [static]

value type.

8.253.2.24 final TCKind TK_SPARSE [static]

A sparse value type.

A sparse value type is one in which all of the fields are not necessarily sent on the network as a part of every sample.

Fields of a sparse value type fall into one of three categories:

- Key fields (see **TypeCode.KEY_MEMBER** (p. 1639))
- Non-key, but required members (see **TypeCode.NONKEY_REQUIRED_MEMBER** (p. 1639))
- Non-key, optional members (see **TypeCode.NONKEY_MEMBER** (p. 1638))

Fields of the first two kinds must appear in every sample. These are also the only kinds of fields on which you can perform content filtering (see **com.rti.dds.topic.ContentFilteredTopic** (p. 458)), because filter evaluation on a non-existent field is not well defined.

8.254 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.

8.254.1 Detailed Description

The properties of a thread of execution.

QoS:

com.rti.dds.infrastructure.EventQosPolicy (p. 930)
com.rti.dds.infrastructure.DatabaseQosPolicy (p. 468)
com.rti.dds.infrastructure.ReceiverPoolQosPolicy (p. 1331)
com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy (p. 387)

8.254.2 Member Data Documentation

8.254.2.1 int mask

Describes the type of thread.

[**default**] 0, use default options of the OS

8.254.2.2 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 if set in a QoS profile document

8.254.2.3 int stack_size

The thread stack-size.

[range] Platform-dependent.

8.254.2.4 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. 1331). It is ignored within other QoS policies that include **com.rti.dds.infrastructure.ThreadSettings_t** (p. 1531).

See also:

Controlling CPU Core Affinity for RTI Threads (p. 1534)

[default] Empty sequence

8.254.2.5 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. 1331). It is ignored within other QoS policies that include **com.rti.dds.infrastructure.ThreadSettings_t** (p. 1531).

See also:

Controlling CPU Core Affinity for RTI Threads (p. 1534)

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.

8.255 ThreadSettingsCpuRotationKind Class Reference

Determines how **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1532) 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.

8.255.1 Detailed Description

Determines how **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1532) affects processor affinity for thread-related QoS policies that apply to multiple threads.

8.255.2 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. 930), **com.rti.dds.infrastructure.DatabaseQosPolicy** (p. 468), and **com.rti.dds.infrastructure.AsynchronousPublisherQosPolicy** (p. 387)). However, the thread settings in the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy** (p. 1331) 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.

If **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1532) is empty, the rotation is irrelevant since

no affinity adjustment will occur. Suppose instead that **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1532) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **com.rti.dds.infrastructure.ThreadSettings_t.cpu_rotation** (p. 1532) is **com.rti.dds.infrastructure.ThreadSettingsCpuRotationKind.THREAD_SETTINGS_CPU_NO_ROTATION** (p. 1535), 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. 1532) is **com.rti.dds.infrastructure.ThreadSettingsCpuRotationKind.THREAD_SETTINGS_CPU_RR_ROTATION** (p. 1535), each thread will be assigned in round-robin fashion to one of the processors in **com.rti.dds.infrastructure.ThreadSettings_t.cpu_list** (p. 1532); 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.

8.255.3 Member Data Documentation

8.255.3.1 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.

8.255.3.2 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.

8.256 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.
- static final int **THREAD_SETTINGS_STDIO**
  Code executed within the thread may access standard I/O.
- static final int **THREAD_SETTINGS_REALTIME_PRIORITY**
  The thread will be schedule on a real-time basis.
- static final int **THREAD_SETTINGS_PRIORITY_ENFORCE**
  Strictly enforce this thread's priority.

8.256.1 Detailed Description

A collection of flags used to configure threads of execution.

Not all of these options may be relevant for all operating systems.

See also:

com.rti.dds.infrastructure.ThreadSettingsKindMask

8.256.2 Member Data Documentation

8.256.2.1 final int THREAD_SETTINGS_FLOATING_POINT [static]

Code executed within the thread may perform floating point operations.

8.256.2.2 final int THREAD_SETTINGS_STDIO [static]

Code executed within the thread may access standard I/O.

8.256.2.3 final int THREAD_SETTINGS_REALTIME_PRIORITY [static]

The thread will be schedule on a real-time basis.

8.256.2.4 final int THREAD_SETTINGS_PRIORITY_ENFORCE [static]

Strictly enforce this thread's priority.

8.257 Time_t Class Reference

Type for *time* representation.

Inherits Struct, and java.io.Externalizable.

Public Member Functions

- **Time_t** (**Time_t** time)
  Copy constructor.
- **Time_t** (int **sec**, int **nanosec**)
  Constructor.
- boolean **is_invalid** ()
- boolean **is_zero** ()
  Check if time is zero.

Public Attributes

- int **sec**
  seconds
- int **nanosec**
  nanoseconds

Static Public Attributes

- static final int **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.

8.257.1 Detailed Description

Type for *time* representation.

A **com.rti.dds.infrastructure.Time_t** (p. 1538) represents a moment in time.

8.257.2 Constructor & Destructor Documentation

8.257.2.1 Time_t (Time_t *time*)

Copy constructor.

Parameters:

time The instance to copy. It must not be null.

8.257.2.2 Time_t (int *sec*, int *nanosec*)

Constructor.

Parameters:

sec must be >=0

nanosec must be >=0

Exceptions:

RETCODE_BAD_PARAMETER *(*p. *1363**)* if either time value is negative

8.257.3 Member Function Documentation

8.257.3.1 boolean is_invalid ()

Returns:

true if the given time is not valid (i.e. is negative)

8.257.3.2 boolean is_zero ()

Check if time is zero.

Returns:

true if the given time is equal to com.rti.dds.infrastructure.Time_t.ZERO or false otherwise.

8.257.4 Member Data Documentation

8.257.4.1 final int TIME_INVALID_SEC [static]

A sentinel indicating an invalid second of time.

8.257.4.2 final int TIME_INVALID_NSEC [static]

A sentinel indicating an invalid nano-second of time.

8.257.4.3 int sec

seconds

8.257.4.4 int nanosec

nanoseconds

8.258 TimeBasedFilterQosPolicy Class Reference

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 473) 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 subequent samples.

8.258.1 Detailed Description

Filter that allows a **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

8.258.2 Usage

You can use this QoS policy to reduce the amount of data received by a **com.rti.dds.subscription.DataReader** (p. 473). **com.rti.dds.publication.DataWriter** (p. 538) entities may send data faster than needed by a **com.rti.dds.subscription.DataReader** (p. 473). For example, a **com.rti.dds.subscription.DataReader** (p. 473) 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. 538) of sensor information may have other **com.rti.dds.subscription.DataReader** (p. 473) 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. 473) 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. 473) will not be delivered to that **com.rti.dds.subscription.DataReader** (p. 473).

The **TIME_BASED_FILTER** (p. 113) also applies to each instance separately; that is, the constraint is that the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), 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. 538) has an infinite **com.rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1167), RTI Connext will only send as many packets to a **com.rti.dds.subscription.DataReader** (p. 473) as required by the TIME_BASED_FILTER, no matter how fast com.rti.dds.topic.example.FooDataWriter.write 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. 538) 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. 538) (for example, when the **com.rti.dds.publication.DataWriter** (p. 538) is reliable, when the data type is keyed, or when the **com.rti.dds.publication.DataWriter** (p. 538) has a finite **com.rti.dds.infrastructure.LivelinessQosPolicy.lease_duration** (p. 1167)), only the data that passes the TIME_BASED_FILTER will be stored in the receive queue of the **com.rti.dds.subscription.DataReader** (p. 473). 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).

8.258.3 Consistency

It is inconsistent for a **com.rti.dds.subscription.DataReader** (p. 473) to have a `minimum_separation` longer than its **DEADLINE** (p. 50) `period`.

However, it is important to be aware of certain edge cases that can occur when your **publication** (p. 338) 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** (p. 338), 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** (p. 338) 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** (p. 338) 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** (p. 338) rate. You may miss deadlines.

See **com.rti.dds.infrastructure.DeadlineQosPolicy** (p. 604) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 113) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 75) and **RELIABILITY** (p. 101) QoS. The **TIME_BASED_FILTER** (p. 113) specifies the samples that are of interest to the **com.rti.dds.subscription.DataReader** (p. 473). The **HISTORY** (p. 75) and **RELIABILITY** (p. 101) 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. 473); that is, they apply *after* the **TIME_BASED_FILTER** (p. 113) has been applied.

In the case where the reliability QoS kind is **ReliabilityQosPolicyKind.RELIABLE_RELIABILITY_QOS** (p. 1341), in steady-state – defined as the situation where the **com.rti.dds.publication.DataWriter** (p. 538) 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. 473).

See also:

DeadlineQosPolicy (p. 604)
HistoryQosPolicy (p. 1071)
ReliabilityQosPolicy (p. 1336)

8.258.4 Member Data Documentation

8.258.4.1 final Duration_t minimum_separation

The minimum separation duration between subequent samples.

[default] 0 (meaning the **com.rti.dds.subscription.DataReader** (p. 473) is potentially interested in all values)

[range] [0,1 year], < **com.rti.dds.infrastructure.DeadlineQosPolicy.period** (p. 606)

8.259 Topic Interface Reference

*<<**interface**>> (p. 271)* 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 StatusKind.INCONSISTENT_TOPIC_STATUS status of a **com.rti.dds.topic.Topic** (p. 1545).*

- void **set_qos** (**TopicQos** qos)

  *Set the **topic** (p. 350) QoS.*

- void **set_qos_with_profile** (String library_name, String profile_name)

  <<**eXtension**>> (p. 270) *Change the QoS of this **topic** (p. 350) using the input XML QoS profile.*

- void **get_qos** (**TopicQos** qos)

  *Get the **topic** (p. 350) QoS.*

- void **set_listener** (**TopicListener** l, int mask)

  *Set the **topic** (p. 350) listener.*

- **TopicListener get_listener** ()

  *Get the **topic** (p. 350) listener.*

8.259.1 Detailed Description

*<<**interface**>> (p. 271)* The most basic description of the data to be published and subscribed.

QoS:

com.rti.dds.topic.TopicQos (p. 1566)

Status:

StatusKind.INCONSISTENT_TOPIC_STATUS,
com.rti.dds.topic.InconsistentTopicStatus (p. 1077)

Listener:

com.rti.dds.topic.TopicListener (p. 1564)

A **com.rti.dds.topic.Topic** (p. 1545) is identified by its name, which must be unique in the whole **domain** (p. 317). In addition (by virtue of extending **com.rti.dds.topic.TopicDescription** (p. 1561)) it fully specifies the type of the data that can be communicated when publishing or subscribing to the **com.rti.dds.topic.Topic** (p. 1545).

com.rti.dds.topic.Topic (p. 1545) is the only **com.rti.dds.topic.TopicDescription** (p. 1561) that can be used for publications and therefore associated with a **com.rti.dds.publication.DataWriter** (p. 538).

The following operations may be called even if the **com.rti.dds.topic.Topic** (p. 1545) is not enabled. Other operations will fail with the value RETCODE_NOT_ENABLED if called on a disabled **com.rti.dds.topic.Topic** (p. 1545):

- All the base-class operations **set_qos()** (p. 1547), **set_qos_with_profile()** (p. 1548), **get_qos()** (p. 1548), **set_listener()** (p. 1549), **get_listener()** (p. 1549) **enable()** (p. 915), **get_statuscondition()** (p. 917) and **get_status_changes()** (p. 917)
- **get_inconsistent_topic_status()** (p. 1546)

See also:

Operations Allowed in Listener Callbacks (p. 1156)

8.259.2 Member Function Documentation

8.259.2.1 void get_inconsistent_topic_status (InconsistentTopicStatus *status*)

Allows the application to retrieve the StatusKind.INCONSISTENT_TOPIC_STATUS status of a **com.rti.dds.topic.Topic** (p. 1545).

Retrieve the current **com.rti.dds.topic.InconsistentTopicStatus** (p. 1077)

Parameters:

status *<<inout>> (p. 271)* Status to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

com.rti.dds.topic.InconsistentTopicStatus (p. 1077)

8.259.2.2 void set_qos (TopicQos *qos*)

Set the **topic** (p. 350) QoS.

The **com.rti.dds.topic.TopicQos.topic_data** (p. 1567) and **com.rti.dds.topic.TopicQos.deadline** (p. 1568), **com.rti.dds.topic.TopicQos.latency_budget** (p. 1568), **com.rti.dds.topic.TopicQos.transport_priority** (p. 1568) and **com.rti.dds.topic.TopicQos.lifespan** (p. 1568) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 271)* Set of policies to be applied to **com.rti.dds.topic.Topic** (p. 1545).

Policies must be consistent. Immutable policies cannot be changed after **com.rti.dds.topic.Topic** (p. 1545) is enabled. The special value **DomainParticipant.TOPIC_QOS_DEFAULT** (p. 148) can be used to indicate that the QoS of the **com.rti.dds.topic.Topic** (p. 1545) should be changed to match the current default **com.rti.dds.topic.TopicQos** (p. 1566) set in the **com.rti.dds.domain.DomainParticipant** (p. 629). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY if immutable policy is changed, or RETCODE_INCONSISTENT_POLICY if policies are inconsistent

See also:

com.rti.dds.topic.TopicQos (p. 1566) for rules on consistency among QoS
set_qos (abstract) (p. 913)
Operations Allowed in Listener Callbacks (p. 1156)

8.259.2.3 void set_qos_with_profile (String *library_name*, String *profile_name*)

<<eXtension>> (p. 270) Change the QoS of this **topic** (p. 350) using the input XML QoS profile.

The **com.rti.dds.topic.TopicQos.topic_data** (p. 1567) and **com.rti.dds.topic.TopicQos.deadline** (p. 1568), **com.rti.dds.topic.TopicQos.latency_budget** (p. 1568), **com.rti.dds.topic.TopicQos.transport_priority** (p. 1568) and **com.rti.dds.topic.TopicQos.lifespan** (p. 1568) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>> (p. 271)* 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. 679)).

profile_name *<<in>> (p. 271)* 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. 680)).

Exceptions:

One of the **Standard Return Codes** (p. 104), RETCODE_IMMUTABLE_POLICY if immutable policy is changed, or RETCODE_INCONSISTENT_POLICY if policies are inconsistent

See also:

com.rti.dds.topic.TopicQos (p. 1566) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1156)

8.259.2.4 void get_qos (TopicQos *qos*)

Get the **topic** (p. 350) QoS.

This method may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters:

qos *<<inout>> (p. 271)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

get_qos (abstract) (p. 914)

8.259.2.5 void set_listener (TopicListener *l*, int *mask*)

Set the **topic** (p. 350) listener.

Parameters:

l <<***in***>> *(p. 271)* Listener to be installed on entity.

mask <<***in***>> *(p. 271)* Changes of communication status to be invoked on the listener.

Exceptions:

One of the **Standard Return Codes** (p. 104)

See also:

set_listener (abstract) (p. 914)

8.259.2.6 TopicListener get_listener ()

Get the **topic** (p. 350) listener.

Returns:

Existing listener attached to the **com.rti.dds.topic.Topic** (p. 1545).

See also:

get_listener (abstract) (p. 915)

8.260 TopicAdapter Class Reference

*<<**eXtension**>> (p. 270)* 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 StatusKind.INCONSISTENT_TOPIC_STATUS status.

8.260.1 Detailed Description

*<<**eXtension**>> (p. 270)* 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.

8.260.2 Member Function Documentation

8.260.2.1 void on_inconsistent_topic (Topic *topic*, InconsistentTopicStatus *status*)

Handle the StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1545) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1545) of the same **topic** (p. 350) name.

Parameters:

topic *(p. 350) <<**out**>> (p. 271)* Locally created **com.rti.dds.topic.Topic** (p. 1545) that triggers the listener callback

status *<<**out**>> (p. 271)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1545)

Implements **TopicListener** (p. 1565).

8.261 TopicBuiltinTopicData Class Reference

Entry created when a **Topic** (p. 1545) 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. 1545).*
- String **type_name**
  *Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).*
- final **DurabilityQosPolicy durability**
  *durability policy of the corresponding **Topic** (p. 1545)*
- final **DurabilityServiceQosPolicy durability_service**
  *durability service policy of the corresponding **Topic** (p. 1545)*
- final **DeadlineQosPolicy deadline**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **LatencyBudgetQosPolicy latency_budget**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **LivelinessQosPolicy liveliness**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **ReliabilityQosPolicy reliability**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **TransportPriorityQosPolicy transport_priority**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **LifespanQosPolicy lifespan**
  *Policy of the corresponding **Topic** (p. 1545).*
- final **DestinationOrderQosPolicy destination_order**
  *Policy of the corresponding **Topic** (p. 1545).*

- final **HistoryQosPolicy history**

  Policy of the corresponding ***Topic*** *(p. 1545).*

- final **ResourceLimitsQosPolicy resource_limits**

  Policy of the corresponding ***Topic*** *(p. 1545).*

- final **OwnershipQosPolicy ownership**

  Policy of the corresponding ***Topic*** *(p. 1545).*

- final **TopicDataQosPolicy topic_data**

  Policy of the corresponding ***Topic*** *(p. 1545).*

8.261.1 Detailed Description

Entry created when a **Topic** (p. 1545) object discovered.

Data associated with the built-in **topic** (p. 350) **TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME** (p. 1558). It contains QoS policies and additional information that apply to the remote **com.rti.dds.topic.Topic** (p. 1545).

Note: The DDS_TopicBuiltinTopicData built-in **topic** (p. 350) is meant to convey information about discovered Topics. This Topic'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 (builtin.PublicationBuiltinTopicData and builtin.SubscriptionBuiltinTopicData). Therefore **TopicBuiltinTopicData** (p. 1552) DataReaders will not receive any data.

See also:

TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME (p. 1558)
builtin.TopicBuiltinTopicDataDataReader (p. 1556)

8.261.2 Member Data Documentation

8.261.2.1 final BuiltinTopicKey_t key

DCPS key to distinguish entries.

8.261.2.2 String name

Name of the **com.rti.dds.topic.Topic** (p. 1545).
The length of this string is limited to 255 characters.

8.261.2.3 String type_name

Name of the type attached to the **com.rti.dds.topic.Topic** (p. 1545).
The length of this string is limited to 255 characters.

8.261.2.4 final DurabilityQosPolicy durability

durability policy of the corresponding **Topic** (p. 1545)

8.261.2.5 final DurabilityServiceQosPolicy durability_service

durability service policy of the corresponding **Topic** (p. 1545)

8.261.2.6 final DeadlineQosPolicy deadline

Policy of the corresponding **Topic** (p. 1545).

8.261.2.7 final LatencyBudgetQosPolicy latency_budget

Policy of the corresponding **Topic** (p. 1545).

8.261.2.8 final LivelinessQosPolicy liveliness

Policy of the corresponding **Topic** (p. 1545).

8.261.2.9 final ReliabilityQosPolicy reliability

Policy of the corresponding **Topic** (p. 1545).

8.261.2.10 final TransportPriorityQosPolicy transport_priority

Policy of the corresponding **Topic** (p. 1545).

8.261.2.11 final LifespanQosPolicy lifespan

Policy of the corresponding **Topic** (p. 1545).

8.261.2.12 final DestinationOrderQosPolicy destination_order

Policy of the corresponding **Topic** (p. 1545).

8.261.2.13 final HistoryQosPolicy history

Policy of the corresponding **Topic** (p. 1545).

8.261.2.14 final ResourceLimitsQosPolicy resource_limits

Policy of the corresponding **Topic** (p. 1545).

8.261.2.15 final OwnershipQosPolicy ownership

Policy of the corresponding **Topic** (p. 1545).

8.261.2.16 final TopicDataQosPolicy topic_data

Policy of the corresponding **Topic** (p. 1545).

8.262 TopicBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` < **builtin.TopicBuiltinTopicData** (p. 1552) > .

Inherits DataReaderImpl.

8.262.1 Detailed Description

Instantiates `DataReader` < **builtin.TopicBuiltinTopicData** (p. 1552) > .

com.rti.dds.subscription.DataReader (p. 473) of **topic** (p. 350) **TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME** (p. 1558) used for accessing **builtin.TopicBuiltinTopicData** (p. 1552) of the remote **com.rti.dds.topic.Topic** (p. 1545).

Note: The **builtin.TopicBuiltinTopicData** (p. 1552) built-in **topic** (p. 350) is meant to convey information about discovered Topics. This Topic'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 (builtin.PublicationBuiltinTopicData and builtin.SubscriptionBuiltinTopicData). Therefore **TopicBuiltinTopicData** (p. 1552) DataReaders will not receive any data.

Instantiates:

<<***generic***>> *(p. 271)* com.rti.dds.topic.example.FooDataReader

See also:

builtin.TopicBuiltinTopicData (p. 1552)
TopicBuiltinTopicDataTypeSupport.TOPIC_TOPIC_NAME (p. 1558)

8.263 TopicBuiltinTopicDataSeq Class Reference

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.TopicBuiltinTopicData** (p. 1552) > .

Inherits AbstractBuiltinTopicDataSeq.

8.263.1 Detailed Description

Instantiates `com.rti.dds.util.Sequence` (p. 1432) < **builtin.TopicBuiltinTopicData** (p. 1552) > .

Instantiates:

<<*generic*>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

builtin.TopicBuiltinTopicData (p. 1552)

8.264 TopicBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` (p. 1651) < **builtin.TopicBuiltinTopicData** (p. 1552) > .

Inheritance diagram for TopicBuiltinTopicDataTypeSupport::

Static Public Attributes

- static final String **TOPIC_TOPIC_NAME** = DDS_TOPIC_TOPIC_NAME()

  Topic *(p. 1545)* ***topic*** *(p. 350) name.*

8.264.1 Detailed Description

Instantiates `TypeSupport` (p. 1651) < **builtin.TopicBuiltinTopicData** (p. 1552) > .

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

See also:

builtin.TopicBuiltinTopicData (p. 1552)

8.264.2 Member Data Documentation

8.264.2.1 final String TOPIC_TOPIC_NAME = DDS_TOPIC_TOPIC_NAME() [static]

Topic (p. 1545) **topic** (p. 350) name.

Topic (p. 1545) name of **builtin.TopicBuiltinTopicDataDataReader** (p. 1556)

See also:

builtin.TopicBuiltinTopicData (p. 1552)
builtin.TopicBuiltinTopicDataDataReader (p. 1556)

8.265 TopicDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Inheritance diagram for TopicDataQosPolicy::

Public Attributes

- final **ByteSeq value**

  a sequence of octets

8.265.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Entity:

com.rti.dds.topic.Topic (p. 1545)

Properties:

RxO (p. 97) = NO
Changeable (p. 98) = **YES** (p. 98)

See also:

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.265.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **com.rti.dds.topic.Topic** (p. 1545) 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. 501), **com.rti.dds.publication.DataWriterListener** (p. 566), or operations such as **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 687), 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. 753).

8.265.3 Member Data Documentation

8.265.3.1 final ByteSeq value

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

8.266 TopicDescription Interface Reference

com.rti.dds.topic.Topic (p. 1545) 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. 1561) .*

- **DomainParticipant get_participant** ()

  Get the ***com.rti.dds.domain.DomainParticipant*** *(p. 629) to which the* ***com.rti.dds.topic.TopicDescription*** *(p. 1561) belongs.*

8.266.1 Detailed Description

com.rti.dds.topic.Topic (p. 1545) entity and associated elements

<<***interface***>> *(p. 271)* Base class for **com.rti.dds.topic.Topic** (p. 1545), **com.rti.dds.topic.ContentFilteredTopic** (p. 458), and **com.rti.dds.topic.MultiTopic** (p. 1208). **com.rti.dds.topic.TopicDescription** (p. 1561) 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** (p. 338) or the **subscription** (p. 343) and therefore creates an implicit association with a **TypeSupport** (p. 1651).

com.rti.dds.topic.TopicDescription (p. 1561) has also a `name` that allows it to be retrieved locally.

See also:

TypeSupport (p. 1651), **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

8.266.2 Member Function Documentation

8.266.2.1 String get_type_name ()

Get the associated `type_name`.

The type name defines a locally unique type for the **publication** (p. 338) or the **subscription** (p. 343).

The `type_name` corresponds to a unique string used to register a type via the **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) method.

Thus, the `type_name` implies an association with a corresponding **TypeSupport** (p. 1651) and this **com.rti.dds.topic.TopicDescription** (p. 1561).

Returns:

the type name. The returned type name is valid until the **com.rti.dds.topic.TopicDescription** (p. 1561) is deleted.

Postcondition:

The result is non-NULL.

See also:

TypeSupport (p. 1651), **com.rti.dds.topic.example.FooTypeSupport** (p. 1060)

8.266.2.2 String get_name ()

Get the name used to create this **com.rti.dds.topic.TopicDescription** (p. 1561) .

Returns:

the name used to create this **com.rti.dds.topic.TopicDescription** (p. 1561). The returned **topic** (p. 350) name is valid until the **com.rti.dds.topic.TopicDescription** (p. 1561) is deleted.

Postcondition:

The result is non-NULL.

8.266.2.3 DomainParticipant get_participant ()

Get the **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.topic.TopicDescription** (p. 1561) belongs.

Returns:

The **com.rti.dds.domain.DomainParticipant** (p. 629) to which the **com.rti.dds.topic.TopicDescription** (p. 1561) belongs.

Postcondition:

The result is non-NULL.

8.267 TopicListener Interface Reference

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for **com.rti.dds.topic.Topic** (p. 1545) entities.

Inheritance diagram for TopicListener::

Public Member Functions

- void **on_inconsistent_topic** (**Topic** topic, **InconsistentTopicStatus** status)

  Handle the StatusKind.INCONSISTENT_TOPIC_STATUS status.

8.267.1 Detailed Description

<<*interface*>> *(p. 271)* **com.rti.dds.infrastructure.Listener** (p. 1154) for **com.rti.dds.topic.Topic** (p. 1545) entities.

Entity:

com.rti.dds.topic.Topic (p. 1545)

Status:

StatusKind.INCONSISTENT_TOPIC_STATUS,
com.rti.dds.topic.InconsistentTopicStatus (p. 1077)

This is the interface that can be implemented by an application-provided class and then registered with the **com.rti.dds.topic.Topic** (p. 1545) such that the application can be notified by RTI Connext of relevant status changes.

See also:

Status Kinds (p. 106)
com.rti.dds.infrastructure.Listener (p. 1154)
com.rti.dds.topic.Topic.set_listener (p. 1549)
Operations Allowed in Listener Callbacks (p. 1156)

8.267.2 Member Function Documentation

8.267.2.1 void on_inconsistent_topic (Topic *topic*, InconsistentTopicStatus *status*)

Handle the StatusKind.INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **com.rti.dds.topic.Topic** (p. 1545) is discovered but is inconsistent with the locally created **com.rti.dds.topic.Topic** (p. 1545) of the same **topic** (p. 350) name.

Parameters:

topic *(p. **350**)* <<***out***>> *(p. 271)* Locally created **com.rti.dds.topic.Topic** (p. 1545) that triggers the listener callback

status <<***out***>> *(p. 271)* Current inconsistent status of locally created **com.rti.dds.topic.Topic** (p. 1545)

Implemented in **DomainParticipantAdapter** (p. 704), and **TopicAdapter** (p. 1550).

8.268 TopicQos Class Reference

QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1545) entity.

Inheritance diagram for TopicQos::

Public Attributes

- final **TopicDataQosPolicy topic_data**
  Topic *(p. 1545) data policy,* ***TOPIC_DATA*** *(p. 114).*
- final **DurabilityQosPolicy durability**
  Durability policy, ***DURABILITY*** *(p. 65).*
- final **DurabilityServiceQosPolicy durability_service**
  DurabilityService policy, ***DURABILITY_SERVICE*** *(p. 66).*
- final **DeadlineQosPolicy deadline**
  Deadline policy, ***DEADLINE*** *(p. 50).*
- final **LatencyBudgetQosPolicy latency_budget**
  Latency budget policy, ***LATENCY_BUDGET*** *(p. 76).*
- final **LivelinessQosPolicy liveliness**
  Liveliness policy, ***LIVELINESS*** *(p. 78).*
- final **ReliabilityQosPolicy reliability**
  Reliability policy, ***RELIABILITY*** *(p. 101).*
- final **DestinationOrderQosPolicy destination_order**
  Destination order policy, ***DESTINATION_ORDER*** *(p. 51).*
- final **HistoryQosPolicy history**
  History policy, ***HISTORY*** *(p. 75).*
- final **ResourceLimitsQosPolicy resource_limits**
  Resource limits policy, ***RESOURCE_LIMITS*** *(p. 102).*
- final **TransportPriorityQosPolicy transport_priority**
  Transport priority policy, ***TRANSPORT_PRIORITY*** *(p. 121).*

- final **LifespanQosPolicy lifespan**

  Lifespan policy, ***LIFESPAN*** *(p. 77).*

- final **OwnershipQosPolicy ownership**

  Ownership policy, ***OWNERSHIP*** *(p. 83).*

8.268.1 Detailed Description

QoS policies supported by a **com.rti.dds.topic.Topic** (p. 1545) entity.

You must set certain members in a consistent manner:

length of **com.rti.dds.topic.TopicQos.topic_data** (p. 1567) .value <= **com.rti.dds.domain.DomainParticipantQos.resource_limits** (p. 739) .topic_data_max_length

If any of the above are not true, **com.rti.dds.topic.Topic.set_qos** (p. 1547), **com.rti.dds.topic.Topic.set_qos_with_profile** (p. 1548) and **com.rti.dds.domain.DomainParticipant.set_default_topic_qos** (p. 642) will fail with RETCODE_INCONSISTENT_POLICY and **com.rti.dds.domain.DomainParticipant.create_topic** (p. 670) will return NULL.

Entity:

com.rti.dds.topic.Topic (p. 1545)

See also:

QoS Policies (p. 90) allowed ranges within each Qos.

8.268.2 Member Data Documentation

8.268.2.1 final TopicDataQosPolicy topic_data

Topic (p. 1545) data policy, **TOPIC_DATA** (p. 114).

8.268.2.2 final DurabilityQosPolicy durability

Durability policy, **DURABILITY** (p. 65).

8.268.2.3 final DurabilityServiceQosPolicy durability_service

DurabilityService policy, **DURABILITY_SERVICE** (p. 66).

8.268.2.4 final DeadlineQosPolicy deadline

Deadline policy, **DEADLINE** (p. 50).

8.268.2.5 final LatencyBudgetQosPolicy latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 76).

8.268.2.6 final LivelinessQosPolicy liveliness

Liveliness policy, **LIVELINESS** (p. 78).

8.268.2.7 final ReliabilityQosPolicy reliability

Reliability policy, **RELIABILITY** (p. 101).

8.268.2.8 final DestinationOrderQosPolicy destination_order

Destination order policy, **DESTINATION_ORDER** (p. 51).

8.268.2.9 final HistoryQosPolicy history

History policy, **HISTORY** (p. 75).

8.268.2.10 final ResourceLimitsQosPolicy resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 102).

8.268.2.11 final TransportPriorityQosPolicy transport_priority

Transport priority policy, **TRANSPORT_PRIORITY** (p. 121).

8.268.2.12 final LifespanQosPolicy lifespan

Lifespan policy, **LIFESPAN** (p. 77).

8.268.2.13 final OwnershipQosPolicy ownership

Ownership policy, **OWNERSHIP** (p. 83).

8.269 Transport Interface Reference

RTI Connext's abstract pluggable **transport** (p. 367) interface.

Inheritance diagram for Transport::

Classes

- class **Property_t**

  *Base structure that must be inherited by derived **Transport** (p. 1569) Plugin classes.*

8.269.1 Detailed Description

RTI Connext's abstract pluggable **transport** (p. 367) interface.

8.270 Transport.Property_t Class Reference

Base structure that must be inherited by derived **Transport** (p. 1569) 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** (p. 367). Should be between 0 and 128.*

- final int **properties_bitmap**

  *A bitmap that defines various properties of the **transport** (p. 367) 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** (p. 367) plugin.*

- int **message_size_max**

  *The maximum size of a message in bytes that can be sent or received by the **transport** (p. 367) 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.*

ˆ 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.*

Static Public Attributes

ˆ static final int **NDDS_TRANSPORT_CLASSID_INVALID** = -1

Invalid ***Transport*** *(p. 1569) Class ID.*

ˆ static final int **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** = 1000

Transport-Plugin class IDs below this are reserved by RTI.

ˆ static final int **NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED** = 0x2

Specified zero-copy behavior of ***transport*** *(p. 367).*

ˆ 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. 1569) Plugin implementation.*

8.270.1 Detailed Description

Base structure that must be inherited by derived **Transport** (p. 1569) Plugin classes.

This structure contains properties that must be set before registration of any **transport** (p. 367) 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** (p. 367) plugin may extend from this structure to add transport-specific properties.

WARNING: The **transport** (p. 367) properties of an instance of a **Transport** (p. 1569) 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** (p. 367) plugin itself should not be changed. If those values are modified, the results are undefined.

8.270.2 Member Data Documentation

8.270.2.1 final int NDDS_TRANSPORT_CLASSID_INVALID = -1 [static]

Invalid **Transport** (p. 1569) Class ID.

Transport-Plugins implementations should set their class ID to a value different than this.

8.270.2.2 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.

8.270.2.3 final int NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED = 0x2 [static]

Specified zero-copy behavior of **transport** (p. 367).

A **Transport** (p. 1569) 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** (p. 367) discretion.
3. Never does zero copy.

This bit should be set only if the **Transport** (p. 1569) 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.

8.270.2.4 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. 1569) Plugin implementation.

For the NDDS_Transport_Property_t structure to be valid, the value of **Transport.Property_t.gather_send_buffer_count_max** (p. 1574) must be greater than or equal to this value.

8.270.2.5 final int classid

The Transport-Plugin Class ID.

Assigned by the implementor of the **transport** (p. 367) plugin, Class ID's below **NDDS_TRANSPORT_CLASSID_RESERVED_RANGE** (p. 1572) 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** (p. 367) plugin.

8.270.2.6 final int address_bit_count

Number of bits in a 16-byte address that are used by the **transport** (p. 367). Should be between 0 and 128.

A **transport** (p. 367) plugin should define the range of addresses (starting from 0x0) 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** (p. 367) plugin is connected.

For **example** (p. 366), 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. 370)

8.270.2.7 final int properties_bitmap

A bitmap that defines various properties of the **transport** (p. 367) to the RTI Connext core.

Currently, the only property supported is whether or not the **transport** (p. 367) 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. 1572)

8.270.2.8 int gather_send_buffer_count_max

Specifies the maximum number of buffers that RTI Connext can pass to the `send()` method of a **transport** (p. 367) plugin.

The **transport** (p. 367) 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** (p. 367) 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. 1573).

If the underlying **transport** (p. 367) does not support a gather-send concept directly, then the **transport** (p. 367) 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** (p. 367) 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.

8.270.2.9 int message_size_max

The maximum size of a message in bytes that can be sent or received by the **transport** (p. 367) plugin.

If the maximum size of a message that can be sent by a **transport** (p. 367) plugin is user configurable, the **transport** (p. 367) plugin should provide a default value for this property. In any case, this value must be set before the **transport** (p. 367) plugin is registered, so that RTI Connext can properly use the plugin.

Note:

- If this value is increased from the default for any of the built-in transports, or if custom transports are used, then the **com.rti.dds.infrastructure.ReceiverPoolQosPolicy.buffer_size** (p. 1333) on the **com.rti.dds.domain.DomainParticipant** (p. 629) should also be changed.

8.270.2.10 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.

Multicast output will be sent and may be received over the interfaces in the list.

It is up to the **transport** (p. 367) plugin to interpret the list of strings passed in.

For **example** (p. 366), 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** (p. 367) 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. 629) is deleted.

8.270.2.11 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. Multicast output will be sent and may be received over the interfaces in the list.

It is up to the **transport** (p. 367) plugin to interpret the list of strings passed in.

For **example** (p. 366), 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** (p. 367) 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. 629) is deleted.

8.270.2.12 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** (p. 367) 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** (p. 367) 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. 629) is deleted.

8.270.2.13 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** (p. 367) 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** (p. 367) 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. 629) is deleted.

8.271 TransportBuiltinKind Class Reference

Built-in transport kind.

Static Public Attributes

- static final int **UDPv4**

  Built-in UDPv4 transport, UDPv4Transport.

- static final String **UDPv4_ALIAS**

  Alias name for the UDPv4 built-in transport.

- static final int **SHMEM**

  Built-in shared memory transport, ShmemTransport.

- static final String **SHMEM_ALIAS**

  Alias name for the shared memory built-in transport.

- static final int **UDPv6**

  Built-in UDPv6 transport, UDPv6Transport.

- static final String **UDPv6_ALIAS**

  Alias name for the UDPv6 built-in transport.

- static final int **MASK_NONE**

  *None of the built-in transports will be registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled. The user must explictly register transports using TransportSupport.register_transport.*

- static final int **MASK_DEFAULT**

  *The default value of **com.rti.dds.infrastructure.TransportBuiltinQosPolicy.mask** (p. 1581).*

- static final int **MASK_ALL**

  *All the available built-in transports are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) is enabled.*

8.271.1 Detailed Description

Built-in transport kind.

See also:

com.rti.dds.infrastructure.TransportBuiltinKindMask

8.271.2 Member Data Documentation

8.271.2.1 final int UDPv4 [static]

Built-in UDPv4 transport, UDPv4Transport.

8.271.2.2 final int SHMEM [static]

Built-in shared memory transport, ShmemTransport.

8.271.2.3 final int UDPv6 [static]

Built-in UDPv6 transport, UDPv6Transport.

8.272 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. 629) is enabled.*

8.272.1 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. 629). 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.

Note: If one application is configured to use UDPv4 *and* shared memory, while another application is only configured for UDPv4, and these two applications run on the same node, they will not communicate. This is due to an internal optimization which will default to use shared memory instead of loopback. However if the other peer application does not enable shared memory there is no common transport, therefore they will not communicate.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.272.2 Member Data Documentation

8.272.2.1 int mask

Specifies the built-in transports that are registered automatically when the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 629) is enabled.

[**default**] **TransportBuiltinKind.MASK_DEFAULT** (p. 116)

8.273 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*** *(p. 350) that matches the* ***com.rti.dds.infrastructure.TransportMulticastMapping_t.topic_expression*** *(p. 1584).*

- String **topic_expression**

  A regular expression that will be used to map ***topic*** *(p. 350) names to corresponding multicast receive addresses.*

- **TransportMulticastMappingFunction_t mapping_function**

  Specifies a function that will define the mapping between a ***topic*** *(p. 350) name and a specific multicast address from a list of addresses.*

8.273.1 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** (p. 350) expression and a mapping function.

QoS:

com.rti.dds.infrastructure.TransportMulticastMappingQosPolicy (p. 1587)

8.273.2 Constructor & Destructor Documentation

8.273.2.1 TransportMulticastMapping_t ()

Constructor.

8.273.2.2 TransportMulticastMapping_t (TransportMulticastMapping_t *src*)

Copy constructor.

8.273.3 Member Data Documentation

8.273.3.1 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** (p. 350) that matches the **com.rti.dds.infrastructure.TransportMulticastMapping_t.topic_expression** (p. 1584).

The string must contain IPv4 or IPv6 addresses separated by commas. For example: "239.255.100.1,239.255.100.2,239.255.100.3"

You may specify ranges of addresses by enclosing the start address and the end address in square brackets. For example: "[239.255.100.1,239.255.100.3]"

You may combine the two approaches. For example:

"239.255.200.1,[239.255.100.1,239.255.100.3], 239.255.200.3"

IPv4 addresses must be specified in Dot-decimal notation.

IPv6 addresses must be specified using 8 groups of 16-bit hexadecimal values separated by colons. For example: FF00:0000:0000:0000:0202:B3FF:FE1E:8329

Leading zeroes can be skipped. For example: "FF00:0:0:0:202:B3FF:FE1E:8329"

You may replace a consecutive number of zeroes with a double colon, but only once within an address. For example: "FF00::202:B3FF:FE1E:8329"

[**default**] NULL

8.273.3.2 String topic_expression

A regular expression that will be used to map **topic** (p. 350) names to corresponding multicast receive addresses.

A **topic** (p. 350) name must match the expression before a corresponding address is assigned.

[default] NULL

8.273.3.3 TransportMulticastMappingFunction_t mapping_function

Specifies a function that will define the mapping between a **topic** (p. 350) name and a specific multicast address from a list of addresses.

This function is optional. If not specified, the middleware will use a hash function to perform the mapping.

8.274 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.

8.274.1 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. 1587)

8.274.2 Constructor & Destructor Documentation

8.274.2.1 TransportMulticastMappingFunction_t ()

Constructor.

8.274.2.2 TransportMulticastMappingFunction_t (TransportMulticastMappingFunction_t *src*)

Copy constructor.

8.274.3 Member Data Documentation

8.274.3.1 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

8.274.3.2 String function_name

Specifies the name of a mapping function.

This function must be implemented in the library specified in **com.rti.dds.infrastructure.TransportMulticastMappingFunction_-t.dll** (p. 1586).

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.

8.275 TransportMulticastMappingQosPolicy Class Reference

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) level) transports with which to receive the multicast data.

Inheritance diagram for TransportMulticastMappingQosPolicy::

Public Attributes

- final **TransportMulticastMappingSeq value**

  A sequence of multicast communications settings.

8.275.1 Detailed Description

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) level) transports with which to receive the multicast data.

By default, a **com.rti.dds.publication.DataWriter** (p. 538) will send individually addressed packets for each **com.rti.dds.subscription.DataReader** (p. 473) that subscribes to the **topic** (p. 350) 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. 538) 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. 1545).

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.275.2 Member Data Documentation

8.275.2.1 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 four.

[default] Empty sequence.

8.276 TransportMulticastMappingSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) `>`.

Inherits ArraySequence.

8.276.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) `>`.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.TransportMulticastSettings_t (p. 1594)

8.277 TransportMulticastQosPolicy Class Reference

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) 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.

8.277.1 Detailed Description

Specifies the multicast address on which a **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) level) transports with which to receive the multicast data.

By default, a **com.rti.dds.publication.DataWriter** (p. 538) will send individually addressed packets for each **com.rti.dds.subscription.DataReader** (p. 473) that subscribes to the **topic** (p. 350) 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. 538) 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. 1545).

Entity:

com.rti.dds.subscription.DataReader (p. 473)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.277.2 Member Data Documentation

8.277.2.1 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 four.

[**default**] Empty sequence.

8.277.2.2 TransportMulticastQosPolicyKind kind

A value that specifies a way to determine how to obtain the multicast address.

This field can have two values.

- If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 119) and the **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1591) does not have any elements, then multicast will not be used.

- If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 119) and the **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1591) has at least one element with a valid address, then that address will be used.

- If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 119) and the **com.rti.dds.infrastructure.TransportMulticastQosPolicy.value** (p. 1591) has at least one element with an empty address, then the address will be obtained from **com.rti.dds.infrastructure.TransportMulticastMappingQosPolicy** (p. 1587).

^ If it is set to **com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** (p. 119), then multicast will not be used.

[default] com.rti.dds.infrastructure.TransportMulticastQosPolicyKind.AUTOMATIC_TRANSPORT_MULTICAST_QOS (p. 119)

8.278 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.

8.278.1 Detailed Description

Transport Multicast Policy Kind.

See also:

com.rti.dds.infrastructure.TransportMulticastQosPolicy (p. 1590)

8.279 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.

8.279.1 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. 1590)

8.279.2 Constructor & Destructor Documentation

8.279.2.1 TransportMulticastSettings_t ()

Constructor with default values.

8.279.2.2 TransportMulticastSettings_t (TransportMulticastSettings_t *src*)

Copy constructor.

8.279.3 Member Data Documentation

8.279.3.1 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. 115).

[**default**] Empty sequence; i.e. all the transports available to the entity.

[**range**] Any sequence of non-null, non-empty strings.

8.279.3.2 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. 57)).

[**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. 57)

8.279.3.3 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. 1714)).

[range] [0,0xffffffff]

8.280 TransportMulticastSettingsSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) `>`.

Inherits ArraySequence.

8.280.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportMulticastSettings_t** (p. 1594) `>`.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.TransportMulticastSettings_t (p. 1594)

8.281 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** (p. 323) as to how to set the priority of the underlying transport used to send the data.*

8.281.1 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-com.rti.dds.publication.DataWriter basis that the data sent by that **com.rti.dds.publication.DataWriter** (p. 538) 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.dds.topic.example.FooDataWriter.write. 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. 538), **com.rti.dds.topic.Topic** (p. 1545)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

8.281.2 Usage

In RTI Connext, for the UDPv4Transport, the value set in the Transport Priority QoS policy is used in a `setsockopt` call to set the TOS (type of service) bits of the IPv4 header for datagrams sent by a **com.rti.dds.publication.DataWriter** (p. 538). It is platform-dependent how and whether the `setsockopt` has an effect. On some platforms, such as Windows and Linux, external permissions must be given to the user application in order to set the TOS bits.

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. 538) to a **com.rti.dds.subscription.DataReader** (p. 473). All network elements, including switches and routers must have the capability and be enabled to actually use the TOS bits 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.

8.281.3 Member Data Documentation

8.281.3.1 int value

This policy is a hint to the **infrastructure** (p. 323) 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

8.282 TransportSelectionQosPolicy Class Reference

Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) 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.

8.282.1 Detailed Description

Specifies the physical transports a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) 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 4 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. 538) to send data or by a **com.rti.dds.subscription.DataReader** (p. 473) to receive data.

Entity:

com.rti.dds.subscription.DataReader (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.282.2 Member Data Documentation

8.282.2.1 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. 629), 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. 629).

An empty sequence is a special value that specifies all the transports installed with the **com.rti.dds.domain.DomainParticipant** (p. 629).

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 115).

[**default**] Empty sequence; i.e. all the transports installed with and available to the **com.rti.dds.domain.DomainParticipant** (p. 629).

[**range**] A sequence of non-null, non-empty strings.

See also:

com.rti.dds.domain.DomainParticipantQos.transport_builtin (p. 738).

8.283 TransportSupport Class Reference

<<*interface*>> *(p. 271)* 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*** *(p. 367) 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*** *(p. 367) plugin.*

8.283.1 Detailed Description

<<*interface*>> *(p. 271)* The utility class used to configure RTI Connext pluggable transports.

8.283.2 Member Function Documentation

8.283.2.1 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** (p. 367) plugin.

Retrieves the properties that will be used to create a builtin **transport** (p. 367) 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. 271)* A valid non-null **com.rti.dds.domain.DomainParticipant** (p. 629)

Parameters:

builtin_transport_property_inout *<<inout>>* *(p. 271)* The storage area where the retrieved property will be output. The specific type required by the `builtin_transport_kind_in` must be used.

Returns:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.283.2.2 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** (p. 367) plugin.

Specifies the properties that will be used to create a builtin **transport** (p. 367) plugin.

If the builtin **transport** (p. 367) is already registered when this operation is called, these property changes will *not* have any effect. Builtin **transport** (p. 367) properties should always be set before the **transport** (p. 367) is registered. See **Built-in Transport Plugins** (p. 216) for details on when a builtin **transport** (p. 367) is registered.

Precondition:

A disabled **com.rti.dds.domain.DomainParticipant** (p. 629). The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_transport_kind_in`.

Parameters:

participant_in *<<in>>* *(p. 271)* A valid non-null **com.rti.dds.domain.DomainParticipant** (p. 629) that has not been enabled.

Parameters:

builtin_transport_property_in *<<inout>>* *(p. 271)* The new **transport** (p. 367) property that will be used to the create the builtin **transport** (p. 367) plugin. The specific type required by the `builtin_transport_kind_in` must be used.

Returns:

One of the **Standard Return Codes** (p. 104), or RETCODE_PRECONDITION_NOT_MET.

See also:

TransportSupport.get_builtin_transport_property() (p. 1602)

8.284 TransportUnicastQosPolicy Class Reference

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 912) to receive data.

Inheritance diagram for TransportUnicastQosPolicy::

Public Attributes

- final **TransportUnicastSettingsSeq value**

  A sequence of unicast communication settings.

8.284.1 Detailed Description

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 912) to receive data.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.284.2 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. 912) announces to remote applications a list of (up to 4) unicast addresses to which the remote application should send data (either user data packets or reliable protocol metadata such as ACK/NACKs and heartbeats).

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. 912) (see **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580) and **com.rti.dds.infrastructure.TransportSelectionQosPolicy** (p. 1600)). Also, the associated ports are automatically determined (see **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1396)).

Use this QoS policy to manually set the receive address list for an **Entity** (p. 912). You may optionally set a port to use a non-default receive port as well. Only the first 4 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. 629), 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. 473), 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. 629).

- For a reliable **com.rti.dds.publication.DataWriter** (p. 538), 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. 629).

8.284.3 Member Data Documentation

8.284.3.1 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.DomainParticipantQos.default_unicast** (p. 738)) should be used.

The RTPS wire protocol currently limits the maximum number of unicast locators to four.

[default] Empty sequence.

See also:

com.rti.dds.domain.DomainParticipantQos.default_unicast

(p. 738)

8.285 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.

8.285.1 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. 1605)

8.285.2 Constructor & Destructor Documentation

8.285.2.1 TransportUnicastSettings_t ()

Constructor.

8.285.2.2 TransportUnicastSettings_t (TransportUnicastSettings_t *src*)

Copy constructor.

8.285.3 Member Data Documentation

8.285.3.1 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. 115).

[**default**] Empty sequence; i.e. all the transports available to the entity.

[**range**] Any sequence of non-null, non-empty strings.

8.285.3.2 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. 1714).

[**range**] [0,0xffffffff]

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.participant_id (p. 1714).

8.286 TransportUnicastSettingsSeq Class Reference

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportUnicastSettings_t** (p. 1608) `>`.

Inherits ArraySequence.

8.286.1 Detailed Description

Declares IDL `sequence<` **com.rti.dds.infrastructure.TransportUnicastSettings_t** (p. 1608) `>`.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

com.rti.dds.infrastructure.TransportUnicastSettings_t (p. 1608)

8.287 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** (p. 290) or to modify types you define yourself at runtime.

Inherits java.io.Serializable.

Public Member Functions

- ^ **TCKind kind** ()

  *Gets the **TCKind** (p. 1526) value of a type code.*

- ^ boolean **equal** (**TypeCode** tc)

  *Compares two **TypeCode** (p. 1611) objects for equality.*

- ^ boolean **equals** (Object tc)

  *Compares two **TypeCode** (p. 1611) objects for equality.*

- ^ 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 **TypeCode** (p. 1611) 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 **TypeCode** (p. 1611) object describes.*

- ^ **TypeCode concrete_base_type** () throws BadKind

  *Returns the **TypeCode** (p. 1611) that describes the concrete base type of the value type that this **TypeCode** (p. 1611) object describes.*

- ^ **TypeCode content_type** () throws BadKind

  *Returns the **TypeCode** (p. 1611) object representing the type for the members of the object described by this **TypeCode** (p. 1611) 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 **TypeCode** (p. 1611) object describing the type of the member identified by the given index.*

- int **member_id** (int index) throws BadKind,Bounds

  Returns the ID of a sparse type code 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 **TypeCode** (p. 1611) in a pseudo-IDL notation.*

- int **add_member** (String name, int id, **TypeCode** tc, byte member_flags) throws BadKind,BadMemberName,BadMemberId

  *Add a new member to this **TypeCode** (p. 1611).*

- 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

  *Add a new member to this **TypeCode** (p. 1611).*

- int **add_member_to_enum** (String name, int ordinal) throws BadKind,BadMemberName

  *Add a new enumerated constant to this enum **TypeCode** (p. 1611).*

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 **TypeCode** (p. 1611) member ID.*

- static final int **INDEX_INVALID**

  *A sentinel indicating an invalid **TypeCode** (p. 1611) 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.

8.287.1 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** (p. 290) or to modify types you define yourself at runtime.

You create **TypeCode** (p. 1611) objects using the **TypeCodeFactory** (p. 1641) 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 **TypeCode** (p. 1611) object concurrently from multiple threads is *unsafe*. Modifying a **TypeCode** (p. 1611) from a single thread while concurrently reading the state of that **TypeCode** (p. 1611) from another thread is also *unsafe*. However, reading the state of a **TypeCode** (p. 1611) concurrently from multiple threads, without any modification, is *safe*.

See also:

`http://java.sun.com/javase/6/docs/api/org/omg/CORBA/TypeCode.html`

8.287.2 Member Function Documentation

8.287.2.1 TCKind kind ()

Gets the **TCKind** (p. 1526) value of a type code.

Retrieves the kind of this **TypeCode** (p. 1611) object. The kind of a type code determines which **TypeCode** (p. 1611) methods may legally be invoked on it.

MT Safety:

SAFE.

Returns:

The type code kind.

8.287.2.2 boolean equal (TypeCode *tc*)

Compares two **TypeCode** (p. 1611) objects for equality.

MT Safety:

SAFE.

Parameters:

tc <<*in*>> *(p. 271)* Type code that will be compared with this **TypeCode** (p. 1611).

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM *(p. 396)* if tc is null.

Returns:

true if the type codes are equal. Otherwise, false.

See also:

`http://java.sun.com/javase/6/docs/api/org/omg/CORBA/TypeCode.html#equal(org.omg.COR`

8.287.2.3 boolean equals (Object *tc*)

Compares two **TypeCode** (p. 1611) objects for equality.

MT Safety:

SAFE.

Parameters:

tc <<*in*>> *(p. 271)* Type code that will be compared with this **TypeCode** (p. 1611).

Returns:

true if the type codes are equal. Otherwise, false.

See also:

`http://java.sun.com/javase/6/docs/api/java/lang/Object.html#equals(java.lang.Object)`

8.287.2.4 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 **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529), **TCKind.TK_STRING** (p. 1529) or **TCKind.TK_WSTRING** (p. 1530).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

The bound for strings and sequences, or the number of elements for arrays if no errors.

8.287.2.5 String name () throws BadKind

Retrieves the simple name identifying this **TypeCode** (p. 1611) object within its enclosing scope.

Precondition:

self kind is **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529), **TCKind.TK_ENUM** (p. 1529), **TCKind.TK_VALUE** (p. 1530), **TCKind.TK_SPARSE** (p. 1530) or **TCKind.TK_ALIAS** (p. 1529).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

Name of the type code if no errors.

8.287.2.6 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 **TCKind.TK_ALIAS** (p. 1529).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

true if an alias is a pointer to the aliased type. Otherwise, false.

8.287.2.7 short type_modifier () throws BadKind

Returns a constant indicating the modifier of the value type that this **TypeCode** (p. 1611) object describes.

Precondition:

self kind is **TCKind.TK_VALUE** (p. 1530).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

One of the following type modifiers: **VM_NONE** (p. 1693), **VM_ABSTRACT** (p. 1691), **VM_CUSTOM** (p. 1692) or **VM_TRUNCATABLE** (p. 1694).

8.287.2.8 TypeCode concrete_base_type () throws BadKind

Returns the **TypeCode** (p. 1611) that describes the concrete base type of the value type that this **TypeCode** (p. 1611) object describes.

Precondition:

self kind is **TCKind.TK_VALUE** (p. 1530) or **TCKind.TK_SPARSE** (p. 1530).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

TypeCode (p. 1611) that describes the concrete base type or null if there is no a concrete base type.

8.287.2.9 TypeCode content_type () throws BadKind

Returns the **TypeCode** (p. 1611) object representing the type for the members of the object described by this **TypeCode** (p. 1611) object.

For sequences and arrays, it returns the element type. For aliases, it returns the original type.

Precondition:

self kind is **TCKind.TK_ARRAY** (p. 1529), **TCKind.TK_SEQUENCE** (p. 1529) or **TCKind.TK_ALIAS** (p. 1529).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

A **TypeCode** (p. 1611) object representing the element type for sequences and arrays, and the original type for aliases.

8.287.2.10 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 **TCKind.TK_ARRAY** (p. 1529).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

Number of dimensions if no errors.

8.287.2.11 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 **TCKind.TK_ARRAY** (p. 1529).
Dimension index in the interval [0,(dimensions count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Dimension index in the interval [0,(dimensions count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

Requested dimension if no errors.

8.287.2.12 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.

8.287.2.13 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 **TypeCode** (p. 1611) objects.

Precondition:

self kind is **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529), **TCKind.TK_ENUM** (p. 1529), **TCKind.TK_VALUE** (p. 1530) or **TCKind.TK_SPARSE** (p. 1530).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

The number of members constituting the type described by this **TypeCode** (p. 1611) object if no errors.

8.287.2.14 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 **TypeCode** (p. 1611) objects.

Precondition:

self kind is **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529), **TCKind.TK_ENUM** (p. 1529), **TCKind.TK_VALUE** (p. 1530) or **TCKind.TK_SPARSE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>> (p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

Name of the member if no errors.

8.287.2.15 TypeCode member_type (int *index*) throws BadKind,Bounds

Retrieves the **TypeCode** (p. 1611) 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 **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529), **TCKind.TK_VALUE** (p. 1530) or **TCKind.TK_SPARSE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>> (p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

The **TypeCode** (p. 1611) object describing the member at the given index if no errors.

8.287.2.16 int member_id (int *index*) throws BadKind,Bounds

Returns the ID of a sparse type code member identified by the given index.

The method can be invoked on sparse **TypeCode** (p. 1611) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition:

self kind is **TCKind.TK_SPARSE** (p. 1530).
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>> (p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

ID of the member if no errors.

8.287.2.17 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 **TypeCode** (p. 1611) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition:

self kind is **TCKind.TK_UNION** (p. 1529).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

Number of labels if no errors.

8.287.2.18 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 **TypeCode** (p. 1611) objects.

Precondition:

self kind is **TCKind.TK_UNION** (p. 1529).
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. 271)* Member index.

label_index *<<in>>* *(p. 271)* Label index.

Exceptions:

com.rti.dds.infrastructure.BadKind *(*p. ***398****)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(*p. ***411****)* - if the index parameter/s are out of range.

Returns:

The evaluated value of the label if no errors.

8.287.2.19 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 **TypeCode** (p. 1611) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition:

self kind is **TCKind.TK_ENUM** (p. 1529).
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<**in**>>* *(*p. *271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(*p. ***398****)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(*p. ***411****)* - if the index parameter/s are out of range.

Returns:

Ordinal that corresponds to the index-th enumerator if no errors.

8.287.2.20 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 **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_VALUE** (p. 1530) or **TCKind.TK_SPARSE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

true if the member is a key. Otherwise, false.

8.287.2.21 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.

Which fields are required depends on the **TCKind** (p. 1526) of the type. For example, in a type of kind **TCKind.TK_SPARSE** (p. 1530), key fields are required. In **TCKind.TK_STRUCT** (p. 1529) and **TCKind.TK_VALUE** (p. 1530) types, all fields are required.

MT Safety:

SAFE.

8.287.2.22 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 **TCKind.TK_STRUCT** (p. 1529), **TCKind.TK_UNION** (p. 1529) or **TCKind.TK_VALUE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Index of the member for which type information is begin requested.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

true if the member is a pointer. Otherwise, false.

8.287.2.23 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 **TCKind.TK_STRUCT** (p. 1529) or **TCKind.TK_VALUE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

true if the member is a bitfield. Otherwise, false.

8.287.2.24 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 **TCKind.TK_STRUCT** (p. 1529) or **TCKind.TK_VALUE** (p. 1530).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

The number of bits of the bitfield or **TypeCode.NOT_BITFIELD** (p. 1640) if the member is not a bitfield.

8.287.2.25 short member_visibility (int *index*) throws BadKind,Bounds

Returns the constant that indicates the visibility of the index-th member.

Precondition:

self kind is **TCKind.TK_VALUE** (p. 1530). The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>> (p. 271)* Member index in the interval [0,(member count-1)].

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

com.rti.dds.infrastructure.Bounds *(p. 411)* - if the index parameter/s are out of range.

Returns:

One of the following constants: **PRIVATE_MEMBER** (p. 1244) or **PUBLIC_MEMBER** (p. 1263).

8.287.2.26 TypeCode discriminator_type () throws BadKind

Returns the discriminator type code.

The method discriminator_type can be invoked only on union **TypeCode** (p. 1611) objects.

Precondition:

self kind is **TCKind.TK_UNION** (p. 1529).

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

TypeCode (p. 1611) object describing the discriminator of the union type if no errors.

8.287.2.27 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 **TypeCode** (p. 1611) objects.

Precondition:

self kind is **TCKind.TK_UNION** (p. 1529)

MT Safety:

SAFE.

Exceptions:

com.rti.dds.infrastructure.BadKind *(p. 398)* if the method is invoked on an inappropriate kind of **TypeCode** (p. 1611) object.

Returns:

The index of the default member, or -1 if there is no default member.

8.287.2.28 int find_member_by_id (int *id*) throws BadKind

Get the index of the member of the given ID.

MT Safety:

SAFE.

8.287.2.29 int find_member_by_name (String *name*) throws BadKind

Get the index of the member of the given name.

MT Safety:

SAFE.

8.287.2.30 void print_IDL (int *indent*)

Prints a **TypeCode** (p. 1611) in a pseudo-IDL notation.

MT Safety:

SAFE.

Parameters:

indent *<<in>> (p. 271)* Indent.

8.287.2.31 int add_member (String *name*, int *id*, TypeCode *tc*, byte *member_flags*) throws BadKind,BadMemberName,BadMemberId

Add a new member to this **TypeCode** (p. 1611).

This method is applicable to **TypeCode** (p. 1611) objects representing structures (**TCKind.TK_STRUCT** (p. 1529)), value types (**TCKind.TK_VALUE** (p. 1530)), sparse value types (**TCKind.TK_SPARSE** (p. 1530)), and unions (**TCKind.TK_UNION** (p. 1529)). To add a constant to an enumeration, see **TypeCode.add_member_to_enum** (p. 1634).

Modifying a **TypeCode** (p. 1611) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 170) 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",
     // If the type is sparse, specify an ID. Otherwise, use this sentinel:
     TypeCode.MEMBER_ID_INVALID,
     TypeCode.TC_LONG,
     // New field is not a key:
     TypeCode.NONKEY_REQUIRED_MEMBER);
```

```
// Sequence of 10 or fewer integers:
myTypeCode.add_member(
    "myFieldName",
    // If the type is sparse, specify an ID. Otherwise, use this sentinel:
    TypeCode.MEMBER_ID_INVALID,
    TypeCodeFactory.get_instance().create_sequence_tc(10, TypeCode.TC_LONG),
    // New field is not a key:
    TypeCode.NONKEY_REQUIRED_MEMBER);
```

MT Safety:

UNSAFE.

Parameters:

name *<<in>>* *(p. 271)* The name of the new member.

id *<<in>>* *(p. 271)* The ID of the new member. This should only be specified for members of kind **TCKind.TK_SPARSE** (p. 1530) and **TCKind.TK_UNION** (p. 1529); otherwise, it should be **TypeCode.MEMBER_ID_INVALID** (p. 1638).

tc *<<in>>* *(p. 271)* The type of the new member. You can get or create this **TypeCode** (p. 1611) with the **TypeCodeFactory** (p. 1641).

member_flags *<<in>>* *(p. 271)* 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:

TypeCode.add_member (p. 1632)
TypeCode.add_member_to_enum (p. 1634)
TypeCodeFactory (p. 1641)
TypeCode.NONKEY_MEMBER (p. 1638)
TypeCode.KEY_MEMBER (p. 1639)
TypeCode.NONKEY_REQUIRED_MEMBER (p. 1639)

8.287.2.32 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

Add a new member to this **TypeCode** (p. 1611).

Modifying a **TypeCode** (p. 1611) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 170) APIs.

MT Safety:

UNSAFE.

Parameters:

name *<<in>> (p. 271)* The name of the new member.

id *<<in>> (p. 271)* The ID of the new member. This should only be specified for members of kind **TCKind.TK_SPARSE** (p. 1530) and **TCKind.TK_UNION** (p. 1529); otherwise, it should be **TypeCode.MEMBER_ID_INVALID** (p. 1638).

tc *<<in>> (p. 271)* The type of the new member. You can get or create this **TypeCode** (p. 1611) with the **TypeCodeFactory** (p. 1641).

member_flags *<<in>> (p. 271)* Indicates whether the member is part of the key and whether it is required.

visibility *<<in>> (p. 271)* Whether the new member is public or private. Non-public members are only relevant for types of kind **TCKind.TK_VALUE** (p. 1530) and **TCKind.TK_SPARSE** (p. 1530).

is_pointer *<<in>> (p. 271)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.

bits *<<in>> (p. 271)* The number of bits, if this new member is a bit field, or **TypeCode.NOT_BITFIELD** (p. 1640).

Returns:

The zero-based index of the new member relative to any other members that previously existed.

See also:

TypeCode.add_member (p. 1632)
TypeCodeFactory (p. 1641)
TypeCode.NONKEY_MEMBER (p. 1638)
TypeCode.KEY_MEMBER (p. 1639)
TypeCode.NONKEY_REQUIRED_MEMBER (p. 1639)

8.287.2.33 int add_member_to_enum (String *name*, int *ordinal*) throws BadKind,BadMemberName

Add a new enumerated constant to this enum **TypeCode** (p. 1611).

This method is applicable to **TypeCode** (p. 1611) objects representing enumerations (**TCKind.TK_ENUM** (p. 1529)). To add a field to a structured type, see **TypeCode.add_member_to_enum** (p. 1634).

Modifying a **TypeCode** (p. 1611) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 170) APIs.

MT Safety:

UNSAFE.

Parameters:

name *<<in>> (p. 271)* The name of the new member. This string must be unique within this type and must not be null.

ordinal *<<in>> (p. 271)* 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:

TypeCode.add_member (p. 1632)
TypeCode.add_member (p. 1632)
TypeCodeFactory (p. 1641)

8.287.3 Member Data Documentation

8.287.3.1 final TypeCode TC_NULL [static]

Basic null type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.2 final TypeCode TC_SHORT [static]

Basic 16-bit signed integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.3 final TypeCode TC_LONG [static]

Basic 32-bit signed integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.4 final TypeCode TC_USHORT [static]

Basic unsigned 16-bit integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.5 final TypeCode TC_ULONG [static]

Basic unsigned 32-bit integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.6 final TypeCode TC_FLOAT [static]

Basic 32-bit floating point type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.7 final TypeCode TC_DOUBLE [static]

Basic 64-bit floating point type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.8 final TypeCode TC_BOOLEAN [static]

Basic Boolean type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.9 final TypeCode TC_CHAR [static]

Basic single-byte character type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.10 final TypeCode TC_OCTET [static]

Basic octet/byte type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.11 final TypeCode TC_LONGLONG [static]

Basic 64-bit integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.12 final TypeCode TC_ULONGLONG [static]

Basic unsigned 64-bit integer type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.13 final TypeCode TC_LONGDOUBLE [static]

Basic 128-bit floating point type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.14 final TypeCode TC_WCHAR [static]

Basic four-byte character type.

See also:

TypeCodeFactory.get_primitive_tc (p. 1650)

8.287.3.15 final int MEMBER_ID_INVALID [static]

A sentinel indicating an invalid **TypeCode** (p. 1611) member ID.

8.287.3.16 final int INDEX_INVALID [static]

A sentinel indicating an invalid **TypeCode** (p. 1611) member index.

8.287.3.17 final byte NONKEY_MEMBER [static]

A flag indicating that a type member is optional and not part of the key.

Only sparse value types (i.e. types of **TCKind** (p. 1526) **TCKind.TK_SPARSE** (p. 1530)) support this flag. Non-key members of other type kinds should use the flag **TypeCode.NONKEY_REQUIRED_MEMBER** (p. 1639).

If a type is used with the **Dynamic Data** (p. 170) facility, a **com.rti.dds.dynamicdata.DynamicData** (p. 780) sample of the type will only contain a value for a **TypeCode.NONKEY_MEMBER** (p. 1638) field if one has been explicitly set (see, for example, DynamicData.set_int). The middleware will *not* assume any default value.

See also:

TypeCode.KEY_MEMBER (p. 1639)
TypeCode.NONKEY_REQUIRED_MEMBER (p. 1639)
TypeCode.KEY_MEMBER (p. 1639)

TypeCode.add_member (p. 1632)
TypeCode.add_member (p. 1632)
TypeCode.is_member_key (p. 1627)
TypeCode.is_member_required (p. 1627)
StructMember.is_key (p. 1477)
ValueMember.is_key (p. 1684)

8.287.3.18 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. 170) facility, all **com.rti.dds.dynamicdata.DynamicData** (p. 780) samples of the type will contain a value for all **TypeCode.KEY_MEMBER** (p. 1639) fields, even if the type is a sparse value type (i.e. of kind **TCKind.TK_SPARSE** (p. 1530)). If you do not set a value of the member explicitly (see, for example, 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:

TypeCode.NONKEY_REQUIRED_MEMBER (p. 1639)
TypeCode.NONKEY_MEMBER (p. 1638)
TypeCode.add_member (p. 1632)
TypeCode.add_member (p. 1632)
TypeCode.is_member_key (p. 1627)
TypeCode.is_member_required (p. 1627)
StructMember.is_key (p. 1477)
ValueMember.is_key (p. 1684)

8.287.3.19 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. 170) facility, all **com.rti.dds.dynamicdata.DynamicData** (p. 780) samples of the type will contain a value for all **TypeCode.NONKEY_REQUIRED_MEMBER** (p. 1639) fields, even if the type is a sparse value type (i.e. of kind **TCKind.TK_SPARSE** (p. 1530)). If you do not set a value of the member explicitly (see, for example, 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:

TypeCode.KEY_MEMBER (p. 1639)
TypeCode.NONKEY_MEMBER (p. 1638)
TypeCode.KEY_MEMBER (p. 1639)
TypeCode.add_member (p. 1632)
TypeCode.add_member (p. 1632)
TypeCode.is_member_key (p. 1627)
TypeCode.is_member_required (p. 1627)
StructMember.is_key (p. 1477)
ValueMember.is_key (p. 1684)

8.287.3.20 final short NOT_BITFIELD [static]

Indicates that a member of a type is not a bitfield.

8.288 TypeCodeFactory Class Reference

A singleton factory for creating, copying, and deleting data type definitions dynamically.

Public Member Functions

- **TypeCode create_struct_tc** (String name, **StructMember**[] members) throws BAD_PARAM

  *Constructs a **TCKind.TK_STRUCT** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_value_tc** (String name, short type_modifier, **TypeCode** concrete_base, **ValueMember**[] members) throws BAD_PARAM

  *Constructs a **TCKind.TK_VALUE** (p. 1530) **TypeCode** (p. 1611).*

- **TypeCode create_sparse_tc** (String name, short type_modifier, **TypeCode** concrete_base) throws BAD_PARAM

  *Constructs a **TCKind.TK_SPARSE** (p. 1530) **TypeCode** (p. 1611).*

- **TypeCode create_union_tc** (String name, **TypeCode** discriminator_type, int default_index, **UnionMember**[] members) throws BAD_PARAM

  *Constructs a **TCKind.TK_UNION** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_enum_tc** (String name, **EnumMember**[] members) throws BAD_PARAM

  *Constructs a **TCKind.TK_ENUM** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_alias_tc** (String name, **TypeCode** original_type, boolean is_pointer) throws BAD_PARAM

  *Constructs a **TCKind.TK_ALIAS** (p. 1529) (typedef) **TypeCode** (p. 1611).*

- **TypeCode create_string_tc** (int bound) throws BAD_PARAM

  *Constructs a **TCKind.TK_STRING** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_wstring_tc** (int bound) throws BAD_PARAM

  *Constructs a **TCKind.TK_WSTRING** (p. 1530) **TypeCode** (p. 1611).*

- **TypeCode create_sequence_tc** (int bound, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **TCKind.TK_SEQUENCE** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_array_tc** (int[] dimensions, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **TCKind.TK_ARRAY** (p. 1529) **TypeCode** (p. 1611).*

- **TypeCode create_array_tc** (int length, **TypeCode** element_type) throws BAD_PARAM

  *Constructs a **TCKind.TK_ARRAY** (p. 1529) **TypeCode** (p. 1611) for a single-dimensional array.*

- **TypeCode clone_tc** (**TypeCode** tc)

  *Creates and returns a copy of the input **TypeCode** (p. 1611).*

- void **delete_tc** (**TypeCode** tc)

  *Deletes the input **TypeCode** (p. 1611).*

- **TypeCode get_primitive_tc** (**TCKind** kind) throws BAD_PARAM

  *Get the **TypeCode** (p. 1611) for a primitive type (integers, floating point values, etc.) identified by the given **TCKind** (p. 1526).*

Static Public Member Functions

- static final **TypeCodeFactory get_instance** ()

  Gets the singleton instance of this class.

8.288.1 Detailed Description

A singleton factory for creating, copying, and deleting data type definitions dynamically.

You can access the singleton with the **TypeCodeFactory.get_instance** (p. 1644) 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 **TypeCode** (p. 1611) class and then register it with the **Dynamic Data** (p. 170) 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.

- ^ **TypeCodeFactory.get_primitive_tc** (p. 1650)

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.

- ^ **TypeCodeFactory.create_string_tc** (p. 1647)
- ^ **TypeCodeFactory.create_wstring_tc** (p. 1648)
- ^ **TypeCodeFactory.create_array_tc** (p. 1649)
- ^ **TypeCodeFactory.create_array_tc** (p. 1649)
- ^ **TypeCodeFactory.create_sequence_tc** (p. 1648)

Creating definitions for structured types:

Structured types include structures, value types, sparse value types, and unions.

- ^ **TypeCodeFactory.create_struct_tc** (p. 1644)
- ^ **TypeCodeFactory.create_value_tc** (p. 1644)
- ^ **TypeCodeFactory.create_sparse_tc** (p. 1645)
- ^ **TypeCodeFactory.create_union_tc** (p. 1646)

Creating definitions for other types:

The type system also supports enumerations and aliases (i.e. `typedef`s in C and C++).

- ^ **TypeCodeFactory.create_enum_tc** (p. 1646)
- ^ **TypeCodeFactory.create_alias_tc** (p. 1647)

Deleting type definitions:

When you're finished using a type definition, you should delete it. (*Note* that you only need to delete a **TypeCode** (p. 1611) that you *created*; if you got the object from **TypeCodeFactory.get_primitive_tc** (p. 1650), you must *not* delete it.)

- ^ **TypeCodeFactory.delete_tc** (p. 1650)

Copying type definitions:

You can also create deep copies of type definitions:

- ^ **TypeCodeFactory.clone_tc** (p. 1649)

8.288.2 Member Function Documentation

8.288.2.1 static final TypeCodeFactory get_instance () [static]

Gets the singleton instance of this class.

Returns:

The **TypeCodeFactory** (p. 1641) instance if no errors. Otherwise, null.

8.288.2.2 TypeCode create_struct_tc (String *name*, StructMember[] *members*) throws BAD_PARAM

Constructs a **TCKind.TK_STRUCT** (p. 1529) **TypeCode** (p. 1611).

Parameters:

name *<<in>>* *(p. 271)* Name of the struct type. Cannot be null.

members *<<in>>* *(p. 271)* Initial members of the structure. This list may be empty (that is, 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.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a struct.

8.288.2.3 TypeCode create_value_tc (String *name*, short *type_modifier*, TypeCode *concrete_base*, ValueMember[] *members*) throws BAD_PARAM

Constructs a **TCKind.TK_VALUE** (p. 1530) **TypeCode** (p. 1611).

Parameters:

name *<<in>>* *(p. 271)* Name of the value type. Cannot be null.

type_modifier *<<in>>* *(p. 271)* One of the value type modifier constants: **VM_NONE** (p. 1693), **VM_CUSTOM** (p. 1692), **VM_ABSTRACT** (p. 1691) or **VM_TRUNCATABLE** (p. 1694).

concrete_base *<<in>>* *(p. 271)* **TypeCode** (p. 1611) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.

members *<<in>>* *(p. 271)* 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.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a value.

8.288.2.4 TypeCode create_sparse_tc (String *name*, short *type_modifier*, TypeCode *concrete_base*) throws BAD_PARAM

Constructs a **TCKind.TK_SPARSE** (p. 1530) **TypeCode** (p. 1611).

A sparse value type is similar to other value types but with one major difference: not all members need to be present in every sample.

It is not possible to generate code for sparse value types; they must be created at runtime using these APIs. You will interact with samples of sparse types using the **Dynamic Data** (p. 170) APIs.

Parameters:

name *<<in>>* *(p. 271)* Name of the value type. Cannot be null.

type_modifier *<<in>>* *(p. 271)* One of the value type modifier constants: **VM_NONE** (p. 1693), **VM_CUSTOM** (p. 1692), **VM_ABSTRACT** (p. 1691) or **VM_TRUNCATABLE** (p. 1694).

concrete_base *<<in>>* *(p. 271)* **TypeCode** (p. 1611) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a value.

8.288.2.5 TypeCode create_union_tc (String *name*, TypeCode *discriminator_type*, int *default_index*, UnionMember[] *members*) throws BAD_PARAM

Constructs a **TCKind.TK_UNION** (p. 1529) **TypeCode** (p. 1611).

Parameters:

name *<<in>> (p. 271)* Name of the union type. Cannot be null.

discriminator_type *<<in>> (p. 271)* Discriminator Type Code. Cannot be null.

default_index *<<in>> (p. 271)* Index of the default member, or -1 if there is no default member.

members *<<in>> (p. 271)* 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.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a union.

8.288.2.6 TypeCode create_enum_tc (String *name*, EnumMember[] *members*) throws BAD_PARAM

Constructs a **TCKind.TK_ENUM** (p. 1529) **TypeCode** (p. 1611).

Parameters:

name *<<in>> (p. 271)* Name of the enum type. Cannot be null.

members *<<in>>* *(p. 271)* 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 **TypeCode.add_member_to_enum** (p. 1634). This argument may be null.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing an enumeration.

8.288.2.7 TypeCode create_alias_tc (String *name*, TypeCode *original_type*, boolean *is_pointer*) throws BAD_PARAM

Constructs a **TCKind.TK_ALIAS** (p. 1529) (typedef) **TypeCode** (p. 1611).

Parameters:

name *<<in>>* *(p. 271)* Name of the alias. Cannot be null.

original_type *<<in>>* *(p. 271)* Aliased type code. Cannot be null.

is_pointer *<<in>>* *(p. 271)* Indicates if the alias is a pointer to the aliased type code.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing an alias.

8.288.2.8 TypeCode create_string_tc (int *bound*) throws BAD_PARAM

Constructs a **TCKind.TK_STRING** (p. 1529) **TypeCode** (p. 1611).

Parameters:

bound *<<in>>* *(p. 271)* Maximum length of the string. It cannot be negative.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a string.

8.288.2.9 TypeCode create_wstring_tc (int *bound*) throws BAD_PARAM

Constructs a **TCKind.TK_WSTRING** (p. 1530) **TypeCode** (p. 1611).

Parameters:

bound *<<in>> (p. 271)* Maximum length of the wide string. It cannot be negative.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a wide string.

8.288.2.10 TypeCode create_sequence_tc (int *bound*, TypeCode *element_type*) throws BAD_PARAM

Constructs a **TCKind.TK_SEQUENCE** (p. 1529) **TypeCode** (p. 1611).

Parameters:

bound *<<in>> (p. 271)* The bound for the sequence (> 0).

element_type *<<in>> (p. 271)* **TypeCode** (p. 1611) object describing the sequence elements.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a sequence.

8.288.2.11 TypeCode create_array_tc (int[] *dimensions*, TypeCode *element_type*) throws BAD_PARAM

Constructs a **TCKind.TK_ARRAY** (p. 1529) **TypeCode** (p. 1611).

Parameters:

dimensions *<<in>>* *(p. 271)* Dimensions of the array. Each dimension has to be greater than 0.

element_type *<<in>>* *(p. 271)* **TypeCode** (p. 1611) describing the array elements. Cannot be null.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a sequence.

8.288.2.12 TypeCode create_array_tc (int *length*, TypeCode *element_type*) throws BAD_PARAM

Constructs a **TCKind.TK_ARRAY** (p. 1529) **TypeCode** (p. 1611) for a single-dimensional array.

Parameters:

length *<<in>>* *(p. 271)* Length of the single-dimensional array.

element_type *<<in>>* *(p. 271)* **TypeCode** (p. 1611) describing the array elements. Cannot be null.

Exceptions:

com.rti.dds.infrastructure.BAD_PARAM. Illegal parameter value.

Returns:

A newly-created **TypeCode** (p. 1611) object describing a sequence.

8.288.2.13 TypeCode clone_tc (TypeCode *tc*)

Creates and returns a copy of the input **TypeCode** (p. 1611).

Parameters:

tc <<***in***>> *(p. 271)* Type code that will be copied. Cannot be null.

Returns:

A clone of tc.

8.288.2.14 void delete_tc (TypeCode *tc*)

Deletes the input **TypeCode** (p. 1611).

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. 271)* Type code that will be deleted. Cannot be null.

8.288.2.15 TypeCode get_primitive_tc (TCKind *kind*) throws BAD_PARAM

Get the **TypeCode** (p. 1611) for a primitive type (integers, floating point values, etc.) identified by the given **TCKind** (p. 1526).

See also:

TypeCode.TC_LONG (p. 1636)
TypeCode.TC_ULONG (p. 1636)
TypeCode.TC_SHORT (p. 1635)
TypeCode.TC_USHORT (p. 1636)
TypeCode.TC_FLOAT (p. 1636)
TypeCode.TC_DOUBLE (p. 1636)
TypeCode.TC_LONGDOUBLE (p. 1638)
TypeCode.TC_OCTET (p. 1637)
TypeCode.TC_BOOLEAN (p. 1637)
TypeCode.TC_CHAR (p. 1637)
TypeCode.TC_WCHAR (p. 1638)

8.289 TypeSupport Interface Reference

*<<**interface**>> (p. 271)* 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::

8.289.1 Detailed Description

*<<**interface**>> (p. 271)* 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.dds.topic.example.FooTypeSupport** (p. 1060), from a description of the type in IDL.

A **TypeSupport** (p. 1651) must be registered using the **com.rti.dds.topic.example.FooTypeSupport.register_type** (p. 1060) operation on this type-specific class before it can be used to create **com.rti.dds.topic.Topic** (p. 1545) objects.

See also:

com.rti.dds.topic.example.FooTypeSupport (p. 1060)
rtiddsgen (p. 290)

8.290 TypeSupportQosPolicy Class Reference

Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

Inheritance diagram for TypeSupportQosPolicy::

Public Attributes

- transient Object **plugin_data**

  Value to pass into the type plugin's de-/serialization function.

8.290.1 Detailed Description

Allows you to attach application-specific values to a DataWriter or DataReader that are passed to the serialization or deserialization routine of the associated data type.

The purpose of this QoS is to allow a user application to pass data to a type plugin's support functions.

Entity:

com.rti.dds.subscription.DataReader (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = N/A

Changeable (p. 98) = **YES** (p. 98)

8.290.2 Usage

This QoS policy allows you to associate a pointer to an object with a **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473). This object pointer is passed to the serialization routine of the data type associated with the **com.rti.dds.publication.DataWriter** (p. 538) or the deserialization routine of the data type associated with the **com.rti.dds.subscription.DataReader** (p. 473).

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. 538) or received by a **com.rti.dds.subscription.DataReader** (p. 473) 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. 538) calls com.rti.dds.topic.example.FooDataWriter.write, and then never again.

8.290.3 Member Data Documentation

8.290.3.1 transient Object plugin_data

Value to pass into the type plugin's de-/serialization function.

[**default**] NULL

8.291 UDPv4Transport Interface Reference

Built-in **transport** (p. 367) plug-in using UDP/IPv4.

Inheritance diagram for UDPv4Transport::

Static Public Attributes

- static final int **BLOCKING_NEVER**

  *Value for **UDPv4Transport.Property_t.send_blocking*** *(p. 1663) to specify non-blocking sockets.*

- static final int **BLOCKING_ALWAYS**

  [default] *Value for **UDPv4Transport.Property_t.send_blocking*** *(p. 1663) to specify blocking sockets.*

Classes

- class **Property_t**

  Configurable IPv4/UDP Transport-Plugin properties.

8.291.1 Detailed Description

Built-in **transport** (p. 367) plug-in using UDP/IPv4.

This **transport** (p. 367) 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.

The user can configure an instance of this plugin to only use unicast or only use multicast, see **UDPv4Transport.Property_t.unicast_enabled** (p. 1660) and **UDPv4Transport.Property_t.multicast_enabled** (p. 1660).

In addition, the user 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 "white" and "black" lists in the base property's fields (**Transport.Property_t.allow_interfaces_list** (p. 1575), **Transport.Property_t.deny_interfaces_list** (p. 1575), **Transport.Property_t.allow_multicast_interfaces_list** (p. 1576), **Transport.Property_t.deny_multicast_interfaces_list** (p. 1576)).

RTI Connext can implicitly create this plugin and register with the **com.rti.dds.domain.DomainParticipant** (p. 629) if this **transport** (p. 367) is specified in **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580).

To specify the properties of the builtin UDPv4 **transport** (p. 367) that is implicitly registered, you can either:

- call **TransportSupport.set_builtin_transport_property** (p. 1603) or
- specify the predefined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) associated with the **com.rti.dds.domain.DomainParticipant** (p. 629). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1655)). Builtin **transport** (p. 367) plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) always overwrite the ones specified through **TransportSupport.set_builtin_transport_property()** (p. 1603). The default value is assumed on any unspecified property.

Note that all properties should be set before the **transport** (p. 367) is implicitly created and registered by RTI Connext. Any properties set after the builtin **transport** (p. 367) is registered will be ignored. See **Built-in Transport Plugins** (p. 216) for details on when a builtin **transport** (p. 367) is registered.

8.291.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) of a **com.rti.dds.domain.DomainParticipant** (p. 629) to configure the builtin UDPv4 **transport** (p. 367) plugin.

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.291.3 Member Data Documentation

8.291.3.1 final int BLOCKING_NEVER [static]

Value for **UDPv4Transport.Property_t.send_blocking** (p. 1663) to specify non-blocking sockets.

8.291.3.2 final int BLOCKING_ALWAYS [static]

[default] Value for **UDPv4Transport.Property_t.send_blocking** (p. 1663) to specify blocking sockets.

Property Name	Description
dds.transport.UDPv4.builtin.parent.address_bit_count	See **Transport.Property_t.address_bit_count** (p. 1573)
dds.transport.UDPv4.builtin.parent.properties_bitmap	See **Transport.Property_t.properties_bitmap** (p. 1574)
dds.transport.UDPv4.builtin.parent.gather_send_buffer_count_max	See **Transport.Property_t.gather_send_buffer_count_max** (p. 1574)
dds.transport.UDPv4.builtin.parent.message_size_max	See **Transport.Property_t.message_size_max** (p. 1574)
dds.transport.UDPv4.builtin.parent.allow_interfaces	See **Transport.Property_t.allow_interfaces_list** (p. 1575) and Transport.Property_t.allow_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366), 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_interfaces	See **Transport.Property_t.deny_interfaces_list** (p. 1575) and Transport.Property_t.deny_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.allow_multicast_interfaces	See **Transport.Property_t.allow_multicast_interfaces_list** (p. 1576) and Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_multicast_interfaces	See **Transport.Property_t.deny_multicast_interfaces_list** (p. 1576) and Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv4.builtin.send_socket_buffer_size	See **UDPv4Transport.Property_t.send_socket_buffer_size** (p. 1660)
dds.transport.UDPv4.builtin.recv_socket_buffer_size	See **UDPv4Transport.Property_t.recv_socket_buffer_size** (p. 1660)

8.292 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** (p. 367) plugin to use unicast for sending and receiving.*

- int **multicast_enabled**

  *Allows the **transport** (p. 367) 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** (p. 367) plugin from putting multicast packets onto the loopback interface.*

- int **ignore_loopback_interface**

  *Prevents the **transport** (p. 367) plugin from using the IP loopback interface.*

- int **ignore_nonup_interfaces**

  *Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as UP by the operating system.*

- int **ignore_nonrunning_interfaces**

*Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as RUNNING by the operating system.*

^ int **no_zero_copy**

*Prevents the **transport** (p. 367) 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.

^ long **transport_priority_mask**

*Set mask for use of **transport** (p. 367) 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.

^ 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.

8.292.1 Detailed Description

Configurable IPv4/UDP Transport-Plugin properties.

The properties in this structure can be modified by the end user to configure the plugin. However, the properties must be set before the plugin is instantiated.

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.292.2 Constructor & Destructor Documentation

8.292.2.1 Property_t ()

Create an empty **UDPv4Transport** (p. 1654) property with default values

8.292.3 Member Data Documentation

8.292.3.1 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 **Transport.Property_t.message_size_max** (p. 1574). The maximum value is operating system-dependent.

8.292.3.2 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 **Transport.Property_t.message_size_max** (p. 1574). The maximum value is operating system-dependent.

8.292.3.3 int unicast_enabled

Allows the **transport** (p. 367) 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 instanced.

8.292.3.4 int multicast_enabled

Allows the **transport** (p. 367) 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 instanced.

8.292.3.5 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** (p. 367) plugin.

See also:

NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT

8.292.3.6 int multicast_loopback_disabled

Prevents the **transport** (p. 367) plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

[NOTE: Windows CE systems do not support multicast loopback. This field is ignored for Windows CE targets.]

8.292.3.7 int ignore_loopback_interface

Prevents the **transport** (p. 367) 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** (p. 367) (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** (p. 367) plugin is available for local traffic, 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** (p. 367).

8.292.3.8 int ignore_nonup_interfaces

Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as UP by the operating system.

The **transport** (p. 367) checks the flags reported by the operating system for each network interface upon initialization. An interface which is not reported as UP will not be used. This property allows the user to configure the **transport** (p. 367) to start using even the interfaces which were not reported as UP.

Two values are allowed:

- **0**: Allow the use of interfaces which were not reported as UP.
- **1**: Do not use interfaces which were not reported as UP.

[default] 1

8.292.3.9 int ignore_nonrunning_interfaces

Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as RUNNING by the operating system.

The **transport** (p. 367) checks the flags reported by the operating system for each network interface upon initialization. An interface which is not reported as UP will not be used. This property allows the same check to be extended to the IFF_RUNNING flag implemented by some operating systems. The RUNNING flag is defined to mean that "all resources are allocated", and may be off if 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** (p. 367) to ignore interfaces that are enabled but not connected to the network.

[**default**] 0 (i.e., do not check RUNNING flag)

8.292.3.10 int no_zero_copy

Prevents the **transport** (p. 367) plugin from doing a zero copy.

By default, this plugin will use the zero copy on OSs that offer it. While this is good for performance, it may sometime tax the OS resources in a manner that cannot be overcome by the application.

The best **example** (p. 366) 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. In case you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off zero copy use.

By default this is set to 0, so RTI Connext will use the zero-copy API if offered by the OS.

8.292.3.11 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 Operating System).
- **NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS IS NOT A SUPPORTED CONFIGURATION AND MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[**default**] NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS.

8.292.3.12 long transport_priority_mask

Set mask for use of **transport** (p. 367) priority field.

This is used in conjunction with **UDPv4Transport.Property_t.transport_priority_mapping_low** (p. 1664) and **UDPv4Transport.Property_t.transport_priority_mapping_high** (p. 1664) to define the mapping from DDS **transport** (p. 367) priority (see **TRANSPORT_PRIORITY** (p. 121)) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit

transport (p. 367) priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For **example** (p. 366), 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** (p. 367) will not set IPv4 TOS for send sockets.

[default] 0.

8.292.3.13 int transport_priority_mapping_low

Set low value of output range to IPv4 TOS.

This is used in conjunction with **UDPv4Transport.Property_t.transport_priority_mask** (p. 1663) and **UDPv4Transport.Property_t.transport_priority_mapping_high** (p. 1664) to define the mapping from DDS **transport** (p. 367) priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0.

8.292.3.14 int transport_priority_mapping_high

Set high value of output range to IPv4 TOS.

This is used in conjunction with **UDPv4Transport.Property_t.transport_priority_mask** (p. 1663) and **UDPv4Transport.Property_t.transport_priority_mapping_low** (p. 1664) to define the mapping from DDS **transport** (p. 367) priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

[default] 0xff.

8.292.3.15 long interface_poll_period

Specifies the period in milliseconds to query for changes in the state of all the interfaces.

The value of this property is ignored if ignore_non_interfaces is 1. If ignore_nonup_interfaces is 0 then the UDPv4 **transport** (p. 367) creates a new thread to query the status of the interfaces. This property specifies the polling period in milliseconds for performing this query.

[**default**] 500 milliseconds.

8.292.3.16 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**] 0.

8.292.3.17 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 **Transport.Property_t.message_size_max** (p. 1574) 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:

Transport.Property_t.message_size_max (p. 1574)

8.293 UDPv6Transport Interface Reference

Built-in **transport** (p. 367) plug-in using UDP/IPv6.

Inheritance diagram for UDPv6Transport::

Static Public Attributes

- static final int **BLOCKING_NEVER**

  *Value for **UDPv6Transport.Property_t.send_blocking*** *(p. 1674) to specify non-blocking sockets.*

- static final int **BLOCKING_ALWAYS**

  [default] *Value for **UDPv6Transport.Property_t.send_blocking*** *(p. 1674) to specify blocking sockets.*

Classes

- class **Property_t**

  Configurable IPv6/UDP Transport-Plugin properties.

8.293.1 Detailed Description

Built-in **transport** (p. 367) plug-in using UDP/IPv6.

This **transport** (p. 367) 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.

The user can configure an instance of this plugin to only use unicast or only use multicast, see **UDPv6Transport.Property_t.unicast_enabled** (p. 1672) and **UDPv6Transport.Property_t.multicast_enabled** (p. 1672).

In addition, the user 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 "white" and "black" lists in the base property's fields (**Transport.Property_t.allow_interfaces_list** (p. 1575), **Transport.Property_t.deny_interfaces_list** (p. 1575), **Transport.Property_t.allow_multicast_interfaces_list** (p. 1576), **Transport.Property_t.deny_multicast_interfaces_list** (p. 1576)).

RTI Connext can implicitly create this plugin and register it with the **com.rti.dds.domain.DomainParticipant** (p. 629) if this **transport** (p. 367) is specified in the **com.rti.dds.infrastructure.TransportBuiltinQosPolicy** (p. 1580).

To specify the properties of the builtin UDPv6 **transport** (p. 367) that is implicitly registered, you can either:

- call **TransportSupport.set_builtin_transport_property** (p. 1603) or
- specify the predefined property names in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) associated with the **com.rti.dds.domain.DomainParticipant** (p. 629). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1667)). Builtin **transport** (p. 367) plugin properties specified in **com.rti.dds.infrastructure.PropertyQosPolicy** (p. 1252) always overwrite the ones specified through **TransportSupport.set_builtin_transport_property()** (p. 1603). The default value is assumed on any unspecified property.

Note that all properties should be set before the **transport** (p. 367) is implicitly created and registered by RTI Connext. Any properties that are set after the builtin **transport** (p. 367) is registered will be ignored. See **Built-in Transport Plugins** (p. 216) for details on when a builtin **transport** (p. 367) is registered.

8.293.2 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. 1252) of a **com.rti.dds.domain.DomainParticipant** (p. 629) to configure the builtin UDPv6 **transport** (p. 367) plugin.

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.293.3 Member Data Documentation

8.293.3.1 final int BLOCKING_NEVER [static]

Value for **UDPv6Transport.Property_t.send_blocking** (p. 1674) to specify non-blocking sockets.

8.293.3.2 final int BLOCKING_ALWAYS [static]

[default] Value for **UDPv6Transport.Property_t.send_blocking** (p. 1674) to specify blocking sockets.

Property Name	Description
dds.transport.UDPv6.builtin.parent.address_bit_count	See **Transport.Property_t.address_bit_count** (p. 1573)
dds.transport.UDPv6.builtin.parent.properties_bitmap	See **Transport.Property_t.properties_bitmap** (p. 1574)
dds.transport.UDPv6.builtin.parent.gather_send_buffer_count_max	See **Transport.Property_t.gather_send_buffer_count_max** (p. 1574)
dds.transport.UDPv6.builtin.parent.message_size_max	See **Transport.Property_t.message_size_max** (p. 1574)
dds.transport.UDPv6.builtin.parent.allow_interfaces	See **Transport.Property_t.allow_interfaces_list** (p. 1575) and Transport.Property_t.allow_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_interfaces	See **Transport.Property_t.deny_interfaces_list** (p. 1575) and Transport.Property_t.deny_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.allow_multicast_interfaces	See **Transport.Property_t.allow_multicast_interfaces_list** (p. 1576) and Transport.Property_t.allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv6.builtin.parent.deny_multicast_interfaces	See **Transport.Property_t.deny_multicast_interfaces_list** (p. 1576) and Transport.Property_t.deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For **example** (p. 366): 127.0.0.1,eth0
dds.transport.UDPv6.builtin.send_socket_buffer_size	See **UDPv6Transport.Property_t.send_socket_buffer_size** (p. 1671)
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See **UDPv6Transport.Property_t.recv_socket_buffer_size** (p. 1672)

8.294 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** (p. 367) plugin to use unicast for sending and receiving.*

- int **multicast_enabled**

  *Allows the **transport** (p. 367) 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** (p. 367) plugin from putting multicast packets onto the loopback interface.*

- int **ignore_loopback_interface**

  *Prevents the **transport** (p. 367) plugin from using the IP loopback interface.*

- int **ignore_nonrunning_interfaces**

  *Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as RUNNING by the operating system.*

- int **no_zero_copy**

*Prevents the **transport** (p. 367) 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** (p. 367) will process IPv4 addresses.*

- long **transport_priority_mask**

  *Set mask for use of **transport** (p. 367) 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.

8.294.1 Detailed Description

Configurable IPv6/UDP Transport-Plugin properties.

The properties in this structure can be modified by the end user to configure the plugin. However, the properties must be set before the plugin is instantiated.

See also:

TransportSupport.set_builtin_transport_property() (p. 1603)

8.294.2 Constructor & Destructor Documentation

8.294.2.1 Property_t ()

Create an empty **UDPv6Transport** (p. 1666) property with default values

8.294.3 Member Data Documentation

8.294.3.1 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 **Transport.Property_t.message_size_max** (p. 1574). The maximum value is operating system-dependent.

8.294.3.2 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 **Transport.Property_t.message_size_max** (p. 1574). The maximum value is operating system-dependent.

8.294.3.3 int unicast_enabled

Allows the **transport** (p. 367) 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 instanced.

8.294.3.4 int multicast_enabled

Allows the **transport** (p. 367) 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 instanced.

8.294.3.5 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** (p. 367) plugin.

See also:

NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT

8.294.3.6 int multicast_loopback_disabled

Prevents the **transport** (p. 367) plugin from putting multicast packets onto the loopback interface.

If multicast loopback is disabled (this value is set to 1), then when sending multicast packets, RTI Connext will *not* put a copy of the packets on the loopback interface. This prevents applications on the same node (including itself) from receiving those packets.

This value is set to 0 by default, meaning multicast loopback is *enabled*.

Disabling multicast loopback (setting this value to 1) may result in minor performance gains when using multicast.

8.294.3.7 int ignore_loopback_interface

Prevents the **transport** (p. 367) 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** (p. 367) (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** (p. 367) (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** (p. 367) plugin is available for local traffic, 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** (p. 367).

8.294.3.8 int ignore_nonrunning_interfaces

Prevents the **transport** (p. 367) plugin from using a network interface that is not reported as RUNNING by the operating system.

The **transport** (p. 367) checks the flags reported by the operating system for each network interface upon initialization. An interface which is not reported as UP will not be used. This property allows the same check to be extended to the IFF_RUNNING flag implemented by some operating systems. The RUNNING

flag is defined to mean that "all resources are allocated", and may be off if 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** (p. 367) to ignore interfaces that are enabled but not connected to the network.

[default] 0 (i.e., do not check RUNNING flag)

8.294.3.9 int no_zero_copy

Prevents the **transport** (p. 367) plugin from doing zero copy.

By default, this plugin will use the zero copy on OSs 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** (p. 366) is if the hardware/device driver lends the buffer to the application itself. If the application does not return the loaned buffers soon enough, the node may error or malfunction. If you cannot reconfigure the H/W, device driver, or the OS to allow the zero copy feature to work for your application, you may have no choice but to turn off the use of zero copy.

By default this is set to 0, so RTI Connext will use the zero copy API if offered by the OS.

8.294.3.10 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 Operating System).
- **NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER:** Sockets are modified to make them non-blocking. THIS IS NOT A SUPPORTED CONFIGURATION AND MAY CAUSE SIGNIFICANT PERFORMANCE PROBLEMS.

[default] NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS.

8.294.3.11 int enable_v4mapped

Specify whether UDPv6 **transport** (p. 367) 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** (p. 367) within the same domain participant.

[**default**] 0.

8.294.3.12 long transport_priority_mask

Set mask for use of **transport** (p. 367) priority field.

If **transport** (p. 367) priority mapping is supported on the platform, this mask is used in conjunction with **UDPv6Transport.Property_t.transport_priority_mapping_low** (p. 1675) and **UDPv6Transport.Property_t.transport_priority_mapping_high** (p. 1676) to define the mapping from DDS **transport** (p. 367) priority (see **TRANSPORT_PRIORITY** (p. 121)) to the IPv6 TCLASS field. Defines a contiguous region of bits in the 32-bit **transport** (p. 367) priority value that is used to generate values for the IPv6 TCLASS field on an outgoing socket. (See the *Platform Notes* to find out if the **transport** (p. 367) priority is supported on a specific platform.)

For **example** (p. 366), 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** (p. 367) will not set IPv6 TCLASS for send sockets.

[**default**] 0.

8.294.3.13 int transport_priority_mapping_low

Set low value of output range to IPv6 TCLASS.

This is used in conjunction with **UDPv6Transport.Property_t.transport_priority_mask** (p. 1675) and **UDPv6Transport.Property_t.transport_priority_mapping_high** (p. 1676) to define the mapping from DDS **transport** (p. 367) priority to the IPv6 TCLASS field. Defines the low value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[**default**] 0.

8.294.3.14 int transport_priority_mapping_high

Set high value of output range to IPv6 TCLASS.

This is used in conjunction with **UDPv6Transport.Property_t.transport_priority_mask** (p. 1675) and **UDPv6Transport.Property_t.transport_priority_mapping_low** (p. 1675) to define the mapping from DDS **transport** (p. 367) priority to the IPv6 TCLASS field. Defines the high value of the output range for scaling.

Note that IPv6 TCLASS is generally an 8-bit value.

[default] 0xff.

8.295 Union Class Reference

8.295.1 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

Version:

Revision

1.2

Date

2006/09/24 08:28:59

8.296 UnionMember Class Reference

A description of a member of a union.

Inherits java.io.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.

8.296.1 Detailed Description

A description of a member of a union.

See also:

TypeCodeFactory.create_union_tc (p. 1646)

8.296.2 Constructor & Destructor Documentation

8.296.2.1 UnionMember (String *name*, boolean *is_pointer*, int[] *labels*, TypeCode *type*)

Constructs a **UnionMember** (p. 1678) object initialized with the given values.

8.296.3 Member Data Documentation

8.296.3.1 String name

The name of the union member.

Cannot be null.

8.296.3.2 boolean is_pointer

Indicates whether the union member is a pointer or not.

8.296.3.3 int [] labels

The labels of the union member.

Each union member should contain at least one label. If the union discriminator type is not int the label value should be evaluated to an integer value. For instance, 'a' would be evaluated to 97.

8.296.3.4 TypeCode type

The type of the union member.

Cannot be null.

8.297 UserDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Inheritance diagram for UserDataQosPolicy::

Public Attributes

- final **ByteSeq value**

  a sequence of octets

8.297.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 153) during discovery.

Entity:

com.rti.dds.domain.DomainParticipant (p. 629), **com.rti.dds.subscription.DataReader** (p. 473), **com.rti.dds.publication.DataWriter** (p. 538)

Properties:

RxO (p. 97) = NO;
Changeable (p. 98) = **YES** (p. 98)

See also:

com.rti.dds.domain.DomainParticipant.get_builtin_subscriber (p. 684)

8.297.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **com.rti.dds.infrastructure.Entity** (p. 912) 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. 686), **com.rti.dds.domain.DomainParticipant.ignore_publication** (p. 688), **com.rti.dds.domain.DomainParticipant.ignore_subscription** (p. 689), and **com.rti.dds.domain.DomainParticipant.ignore_topic** (p. 687), 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. 753), **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPolicy.** **user_data_max_length** (p. 754), and **com.rti.dds.infrastructure.DomainParticipantResourceLimitsQosPol** **user_data_max_length** (p. 754).

8.297.3 Member Data Documentation

8.297.3.1 final ByteSeq value

a sequence of octets

[**default**] empty (zero-length)

[**range**] Octet sequence of length [0,max_length]

8.298 UserException Class Reference

User exception.

Inheritance diagram for UserException::

8.298.1 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`

8.299 ValueMember Class Reference

A description of a member of a value type.

Inherits java.io.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.

8.299.1 Detailed Description

A description of a member of a value type.

See also:

TypeCodeFactory.create_value_tc (p. 1644)

8.299.2 Constructor & Destructor Documentation

8.299.2.1 ValueMember (String *name*, boolean *is_pointer*, short *bits*, boolean *is_key*, short *access*, TypeCode *type*)

Constructs a **ValueMember** (p. 1683) object initialized with the given values.

8.299.3 Member Data Documentation

8.299.3.1 String name

The name of the value member.

Cannot be null.

8.299.3.2 TypeCode type

The type of the value member.

Cannot be null.

8.299.3.3 boolean is_pointer

Indicates whether the value member is a pointer or not.

8.299.3.4 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 **TypeCode.NOT_BITFIELD** (p. 1640).

8.299.3.5 boolean is_key

Indicates if the value member is a key member or not.

8.299.3.6 short access

The type of access (public, private) for the value member.

It can take the values: **PRIVATE_MEMBER** (p. 1244) or **PUBLIC_MEMBER** (p. 1263).

8.300 VendorId_t Class Reference

<<*eXtension*>> *(p. 270)* 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.

8.300.1 Detailed Description

<<*eXtension*>> *(p. 270)* Type used to represent the vendor of the service implementing the RTPS protocol.

8.300.2 Constructor & Destructor Documentation

8.300.2.1 VendorId_t ()

Constructor.

8.300.3 Member Data Documentation

8.300.3.1 final VendorId_t UNKNOWN [static]

The ID used when the vendor of the service implementing the RTPS protocol is not known.

8.300.3.2 final int LENGTH_MAX = 2 [static]

Length of vendor id.

8.300.3.3 final byte [] vendorId = new byte[LENGTH_MAX]

The vendor Id.

8.301 Version Class Reference

<<*interface*>> *(p. 271)* 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.

8.301.1 Detailed Description

<<*interface*>> *(p. 271)* 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.

8.301.2 Member Function Documentation

8.301.2.1 static Version get_instance () [static]

Get the singleton instance of this type.

8.301.2.2 ProductVersion_t get_product_version ()

Get the RTI Connext product version.

8.301.2.3 LibraryVersion_t get_java_api_version ()

Get the version of the Java API library.

8.301.2.4 LibraryVersion_t get_c_api_version ()

Get the version of the C API library.

8.301.2.5 LibraryVersion_t get_core_version ()

Get the version of the core library.

8.301.2.6 String toString ()

Get this version in string form.

Combine all of the constituent library versions into a single string.

8.302 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 ***ViewStateKind.NEW_VIEW_STATE*** *(p. 1690) | **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690).*

8.302.1 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. 473). The view state can be either:

- **ViewStateKind.NEW_VIEW_STATE** (p. 1690) indicates that either this is the first time that the **com.rti.dds.subscription.DataReader** (p. 473) has ever accessed samples of that instance, or else that the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1410) and the **com.rti.dds.subscription.SampleInfo.no_writers_generation_count** (p. 1411).

- **ViewStateKind.NOT_NEW_VIEW_STATE** (p. 1690) indicates that the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1404) is a snapshot of the view state of the instance relative to the **com.rti.dds.subscription.DataReader** (p. 473) 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. 473), the middleware can reclaim all local resources regarding the instance. Future samples will be treated as "never seen."

8.302.2 Member Data Documentation

8.302.2.1 final int NEW_VIEW_STATE = 0x0001 << 0 [static]

New instance. This latest generation of the instance has not previously been accessed.

8.302.2.2 final int NOT_NEW_VIEW_STATE = 0x0001 << 1 [static]

Not a new instance. This latest generation of the instance has previously been accessed.

8.303 VM_ABSTRACT Class Reference

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

Static Public Attributes

- ^ static final short **VALUE**

8.303.1 Detailed Description

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

An abstract value type may not be instantiated.

8.303.2 Member Data Documentation

8.303.2.1 final short VALUE [static]

Constant value.

8.304 VM_CUSTOM Class Reference

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

Static Public Attributes

- ^ static final short **VALUE**

8.304.1 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.

8.304.2 Member Data Documentation

8.304.2.1 final short VALUE [static]

Constant value.

8.305 VM_NONE Class Reference

Constant used to indicate that a value type has no modifiers.

Static Public Attributes

- ^ static final short **VALUE**

8.305.1 Detailed Description

Constant used to indicate that a value type has no modifiers.

8.305.2 Member Data Documentation

8.305.2.1 final short VALUE [static]

Constant value.

8.306 VM_TRUNCATABLE Class Reference

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

Static Public Attributes

- ^ static final short **VALUE**

8.306.1 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.

8.306.2 Member Data Documentation

8.306.2.1 final short VALUE [static]

Constant value.

8.307 WaitSet Class Reference

<<*interface*>> *(p. 271)* Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.Condition** (p. 451) objects has a `trigger_value` of true or else until the timeout expires.

Inherits AbstractNativeObject.

Public Member Functions

- **WaitSet** ()

  Default no-argument constructor.

- **WaitSet** (**WaitSetProperty_t** prop)

  <<**eXtension**>> (p. 270) *Constructor for a **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. 451) to the **com.rti.dds.infrastructure.WaitSet** (p. 1695).*

- void **detach_condition** (**Condition** cond)

  *Detaches a **com.rti.dds.infrastructure.Condition** (p. 451) from the **com.rti.dds.infrastructure.WaitSet** (p. 1695).*

- void **get_conditions** (**ConditionSeq** attached_conditions)

  *Retrieves the list of attached **com.rti.dds.infrastructure.Condition** (p. 451) (s).*

- void **set_property** (**WaitSetProperty_t** prop)

  <<**eXtension**>> (p. 270) *Sets the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1705), to configure the associated **com.rti.dds.infrastructure.WaitSet** (p. 1695) to return after one or more trigger events have occurred.*

- void **get_property** (**WaitSetProperty_t** prop)

  <<**eXtension**>> (p. 270) *Retrieves the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1705) configuration of the associated **com.rti.dds.infrastructure.WaitSet** (p. 1695).*

- void **delete** ()

  Destructor.

8.307.1 Detailed Description

<<***interface***>> *(p. 271)* Allows an application to wait until one or more of the attached **com.rti.dds.infrastructure.Condition** (p. 451) objects has a `trigger_value` of true or else until the timeout expires.

8.307.2 Usage

com.rti.dds.infrastructure.Condition (p. 451) (s) (in conjunction with wait-sets) provide an alternative mechanism to allow the middleware to communicate communication status changes (including arrival of data) to the application.

This mechanism is wait-based. Its general use pattern is as follows:

- The application indicates which relevant information it wants to get by creating **com.rti.dds.infrastructure.Condition** (p. 451) objects (**com.rti.dds.infrastructure.StatusCondition** (p. 1452), **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324)) and attaching them to a **com.rti.dds.infrastructure.WaitSet** (p. 1695).

- It then waits on that **com.rti.dds.infrastructure.WaitSet** (p. 1695) until the `trigger_value` of one or several **com.rti.dds.infrastructure.Condition** (p. 451) objects become true.

- It then uses the result of the wait (i.e., `active_conditions`, the list of **com.rti.dds.infrastructure.Condition** (p. 451) objects with `trigger_value` == true) to actually get the information:

  - by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917) and then `get_<communication_status>()` on the relevant **com.rti.dds.infrastructure.Entity** (p. 912), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1452) and the status changes, refer to plain communication status;
  - by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917) and then **com.rti.dds.subscription.Subscriber.get_datareaders**

(p. 1491) on the relevant **com.rti.dds.subscription.Subscriber** (p. 1478) (and then com.rti.dds.topic.example.FooDataReader.read() or com.rti.dds.topic.example.FooDataReader.take on the returned **com.rti.dds.subscription.DataReader** (p. 473) objects), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1452) and the status changes refers to **StatusKind.DATA_ON_READERS_STATUS** (p. 1460);

- by calling **com.rti.dds.infrastructure.Entity.get_status_changes** (p. 917) and then com.rti.dds.topic.example.FooDataReader.read() or com.rti.dds.topic.example.FooDataReader.take on the relevant **com.rti.dds.subscription.DataReader** (p. 473), if the condition is a **com.rti.dds.infrastructure.StatusCondition** (p. 1452) and the status changes refers to **StatusKind.DATA_AVAILABLE_STATUS** (p. 1460);
- by calling directly com.rti.dds.topic.example.FooDataReader.read_w_condition or com.rti.dds.topic.example.FooDataReader.take_w_condition on a **com.rti.dds.subscription.DataReader** (p. 473) with the **com.rti.dds.infrastructure.Condition** (p. 451) as a parameter if it is a **com.rti.dds.subscription.ReadCondition** (p. 1326) or a **com.rti.dds.subscription.QueryCondition** (p. 1324).

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. 451) objects), **com.rti.dds.infrastructure.Condition** (p. 451) objects are meant to embed all that is needed to react properly when enabled. In particular, **com.rti.dds.infrastructure.Entity** (p. 912)-related conditions are related to exactly one **com.rti.dds.infrastructure.Entity** (p. 912) and cannot be shared.

The blocking behavior of the **com.rti.dds.infrastructure.WaitSet** (p. 1695) is illustrated below.

The result of a **WaitSet.wait** (p. 1701) operation depends on the state of the **com.rti.dds.infrastructure.WaitSet** (p. 1695), which in turn depends on whether at least one attached **com.rti.dds.infrastructure.Condition** (p. 451) has a `trigger_value` of true. If the wait operation is called on **com.rti.dds.infrastructure.WaitSet** (p. 1695) with state BLOCKED, it will block the calling thread. If wait is called on a **com.rti.dds.infrastructure.WaitSet** (p. 1695) with state UNBLOCKED, it will return immediately. In addition, when the

com.rti.dds.infrastructure.WaitSet (p. 1695) transitions from BLOCKED to UNBLOCKED it wakes up any threads that had called wait on it.

A key aspect of the **com.rti.dds.infrastructure.Condition** (p. 451)/com.**rti.dds.infrastructure.WaitSet** (p. 1695) mechanism is the setting of the `trigger_value` of each **com.rti.dds.infrastructure.Condition** (p. 451).

8.307.3 Trigger State of a com.rti.dds.infrastructure.StatusCondition

The `trigger_value` of a **com.rti.dds.infrastructure.StatusCondition** (p. 1452) is the boolean OR of the ChangedStatusFlag of all the communication statuses (see **Status Kinds** (p. 106)) to which it is sensitive. That is, `trigger_value` == false only if all the values of the ChangedStatusFlags are false.

The sensitivity of the **com.rti.dds.infrastructure.StatusCondition** (p. 1452) 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. 1453) operation.

8.307.4 Trigger State of a com.rti.dds.subscription.ReadCondition

Similar to the **com.rti.dds.infrastructure.StatusCondition** (p. 1452), a **com.rti.dds.subscription.ReadCondition** (p. 1326) also has a `trigger_value` that determines whether the attached **com.rti.dds.infrastructure.WaitSet** (p. 1695) is BLOCKED or UNBLOCKED. However, unlike the **com.rti.dds.infrastructure.StatusCondition** (p. 1452), the `trigger_value` of the **com.rti.dds.subscription.ReadCondition** (p. 1326) is tied to the presence of *at least a sample* managed by RTI Connext with **com.rti.dds.subscription.SampleStateKind** (p. 1430) and **com.rti.dds.subscription.ViewStateKind** (p. 1689) matching those of the **com.rti.dds.subscription.ReadCondition** (p. 1326). Furthermore, for the **com.rti.dds.subscription.QueryCondition** (p. 1324) to have a `trigger_value` == true, the data associated with the sample must be such that the `query_expression` evaluates to true.

The fact that the trigger_value of a **com.rti.dds.subscription.ReadCondition** (p. 1326) depends on the presence of samples on the associated **com.rti.dds.subscription.DataReader** (p. 473) implies that a single take operation can potentially change the `trigger_value` of several **com.rti.dds.subscription.ReadCondition** (p. 1326) or **com.rti.dds.subscription.QueryCondition** (p. 1324) conditions. For example, if all samples are taken, any **com.rti.dds.subscription.ReadCondition**

(p. 1326) and **com.rti.dds.subscription.QueryCondition** (p. 1324) conditions associated with the **com.rti.dds.subscription.DataReader** (p. 473) 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. 1695) 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. 1695), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwakeup' the **WaitSet** (p. 1695) as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. 1695) objects and taking data associated with the same **com.rti.dds.subscription.DataReader** (p. 473) entity.

To elaborate further, consider the following example: A **com.rti.dds.subscription.ReadCondition** (p. 1326) that has a `sample_state_mask` = {SampleStateKind.NOT_READ_SAMPLE_STATE} will have `trigger_value` of true whenever a new sample arrives and will transition to false as soon as all the newly-arrived samples are either read (so their sample state changes to READ) or taken (so they are no longer managed by RTI Connext). However if the same **com.rti.dds.subscription.ReadCondition** (p. 1326) had a `sample_state_mask` = { SampleStateKind.READ_SAMPLE_STATE, SampleStateKind.NOT_READ_SAMPLE_STATE }, then the `trigger_value` would only become false once all the newly-arrived samples are taken (it is not sufficient to read them as that would only change the sample state to READ), which overlaps the mask on the **com.rti.dds.subscription.ReadCondition** (p. 1326).

8.307.5 Trigger State of a com.rti.dds.infrastructure.GuardCondition

The `trigger_value` of a **com.rti.dds.infrastructure.GuardCondition** (p. 1066) is completely controlled by the application via the operation **com.rti.dds.infrastructure.GuardCondition.set_trigger_value** (p. 1067).

Important: The **com.rti.dds.infrastructure.WaitSet** (p. 1695) allocates native resources. When **com.rti.dds.infrastructure.WaitSet** (p. 1695) is no longer being used, user should call **WaitSet.delete** (p. 1703) explicitly to properly cleanup all native resources.

See also:

Status Kinds (p. 106)

com.rti.dds.infrastructure.StatusCondition (p. 1452),
com.rti.dds.infrastructure.GuardCondition (p. 1066)
com.rti.dds.infrastructure.Listener (p. 1154)

8.307.6 Constructor & Destructor Documentation

8.307.6.1 WaitSet ()

Default no-argument constructor.

Construct a new **com.rti.dds.infrastructure.WaitSet** (p. 1695).

Important: The **com.rti.dds.infrastructure.WaitSet** (p. 1695) allocates native resources. When the **com.rti.dds.infrastructure.WaitSet** (p. 1695) is no longer being used, users should call **WaitSet.delete** (p. 1703) explicitly to properly clean up all native resources.

Exceptions:

RETCODE_OUT_OF_RESOURCES *(p. **1370**)* if a new **com.rti.dds.infrastructure.WaitSet** (p. 1695) could not be allocated.

8.307.6.2 WaitSet (WaitSetProperty_t *prop*)

*<<**eXtension**>>* *(p. 270)* Constructor for a **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. 1695).

Important: The **com.rti.dds.infrastructure.WaitSet** (p. 1695) allocates native resources. When the **com.rti.dds.infrastructure.WaitSet** (p. 1695) is no longer being used, users should call **WaitSet.delete** (p. 1703) explicitly to properly clean up all native resources.

Parameters:

prop *<<**in**>> (p. 271)* Property of wait set controlling when the wait set should be woken up

Exceptions:

RETCODE_OUT_OF_RESOURCES *(p. **1370**)* if a new **com.rti.dds.infrastructure.WaitSet** (p. 1695) could not be allocated.

8.307.7 Member Function Documentation

8.307.7.1 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. 1695) have a `trigger_value` of true, the wait operation will block suspending the calling thread.

The result of the wait operation is the list of all the attached conditions that have a `trigger_value` of true (i.e., the conditions that unblocked the wait).

Note: The resolution of the `timeout` period is constrained by the resolution of the system clock.

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. 451) objects is true, wait will return with the return code **RETCODE_TIMEOUT** (p. 1372). In this case, the resulting list of conditions will be empty.

RETCODE_TIMEOUT (p. 1372) will *not* be returned when the `timeout` duration is exceeded if attached **com.rti.dds.infrastructure.Condition** (p. 451) objects are true, or in the case of a **com.rti.dds.infrastructure.WaitSet** (p. 1695) waiting for more than one trigger event, if one or more trigger events have occurred.

It is not allowable for for more than one application thread to be waiting on the same **com.rti.dds.infrastructure.WaitSet** (p. 1695). If the wait operation is invoked on a **com.rti.dds.infrastructure.WaitSet** (p. 1695) that already has a thread blocking on it, the operation will return immediately with the value **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

Parameters:

active_conditions *<<inout>>* *(p. 271)* a valid non-null **com.rti.dds.infrastructure.ConditionSeq** (p. 452) object. Note that RTI Connext will not allocate a new object if `active_conditions` is null; the method will return **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

timeout *<<in>>* *(p. 271)* a wait timeout

Exceptions:

One of the **Standard Return Codes** (p. 104) or **RETCODE_PRECONDITION_NOT_MET** (p. 1371) or **RETCODE_TIMEOUT** (p. 1372).

8.307.7.2 void attach_condition (Condition *cond*)

Attaches a **com.rti.dds.infrastructure.Condition** (p. 451) to the **com.rti.dds.infrastructure.WaitSet** (p. 1695).

It is possible to attach a **com.rti.dds.infrastructure.Condition** (p. 451) on a **com.rti.dds.infrastructure.WaitSet** (p. 1695) that is currently being waited upon (via the wait operation). In this case, if the **com.rti.dds.infrastructure.Condition** (p. 451) has a `trigger_value` of true, then attaching the condition will unblock the **com.rti.dds.infrastructure.WaitSet** (p. 1695).

Parameters:

cond *<<in>> (p. 271)* Contition to be attached.

Exceptions:

One of the **Standard Return Codes** (p. 104), or **RETCODE_OUT_OF_RESOURCES** (p. 1370).

8.307.7.3 void detach_condition (Condition *cond*)

Detaches a **com.rti.dds.infrastructure.Condition** (p. 451) from the **com.rti.dds.infrastructure.WaitSet** (p. 1695).

If the **com.rti.dds.infrastructure.Condition** (p. 451) was not attached to the **com.rti.dds.infrastructure.WaitSet** (p. 1695) the operation will return **RETCODE_BAD_PARAMETER** (p. 1363).

Parameters:

cond *<<in>> (p. 271)* **Condition** (p. 451) to be detached.

Exceptions:

One of the **Standard Return Codes** (p. 104), or **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

8.307.7.4 void get_conditions (ConditionSeq *attached_conditions*)

Retrieves the list of attached **com.rti.dds.infrastructure.Condition** (p. 451) (s).

Parameters:

attached_conditions *<<inout>>* *(p. 271)* a **com.rti.dds.infrastructure.ConditionSeq** (p. 452) object where the list of attached conditions will be returned

Exceptions:

One of the **Standard Return Codes** (p. 104), or **RETCODE_PRECONDITION_NOT_MET** (p. 1371).

8.307.7.5 void set_property (WaitSetProperty_t *prop*)

<<eXtension>> *(p. 270)* Sets the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1705), to configure the associated **com.rti.dds.infrastructure.WaitSet** (p. 1695) to return after one or more trigger events have occurred.

Parameters:

prop *<<in>>* *(p. 271)*

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.307.7.6 void get_property (WaitSetProperty_t *prop*)

<<eXtension>> *(p. 270)* Retrieves the **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1705) configuration of the associated **com.rti.dds.infrastructure.WaitSet** (p. 1695).

Parameters:

prop *<<out>>* *(p. 271)*

Exceptions:

One of the **Standard Return Codes** (p. 104)

8.307.7.7 void delete ()

Destructor.

Releases the resources asociated with this **com.rti.dds.infrastructure.WaitSet** (p. 1695).

Calling this method multiple times on the same **com.rti.dds.infrastructure.WaitSet** (p. 1695) is safe; subsequent deletions will have no effect.

8.308 WaitSetProperty_t Class Reference

<<*eXtension*>> *(*p. *270)* Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. *1695) 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. *1695) to awaken.*

8.308.1 Detailed Description

<<*eXtension*>> *(*p. *270)* Specifies the **com.rti.dds.infrastructure.WaitSet** (p. 1695) behavior for multiple trigger events.

In simple use, a **com.rti.dds.infrastructure.WaitSet** (p. 1695) returns when a single trigger event occurs on one of its attached **com.rti.dds.infrastructure.Condition** (p. 451) (s), or when the `timeout` maximum wait duration specified in the **WaitSet.wait** (p. 1701) call expires.

The **com.rti.dds.infrastructure.WaitSetProperty_t** (p. 1705) allows configuration of the waiting behavior of a **com.rti.dds.infrastructure.WaitSet** (p. 1695). 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. 1695) 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 **WaitSet.wait** (p. 1701) call continues to apply.

Entity:

com.rti.dds.infrastructure.WaitSet (p. 1695)

Properties:

RxO (p. 97) = N/A

Changeable (p. 98) = **YES** (p. 98)

8.308.2 Member Data Documentation

8.308.2.1 int max_event_count

Maximum number of trigger events to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1695) to awaken.

The **com.rti.dds.infrastructure.WaitSet** (p. 1695) will wait until up to `max_event_count` trigger events have occurred before returning. The **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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

8.308.2.2 final Duration_t max_event_delay

Maximum delay from occurrence of first trigger event to cause a **com.rti.dds.infrastructure.WaitSet** (p. 1695) to awaken.

The **com.rti.dds.infrastructure.WaitSet** (p. 1695) 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. 1695).

Note that **RETCODE_TIMEOUT** (p. 1372) is *not* returned if `max_event_delay` is exceeded. **RETCODE_TIMEOUT** (p. 1372) is returned only if the `timeout` duration expires before any trigger events occur.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE

8.309 WcharSeq Class Reference

Instantiates com.rti.dds.util.Sequence (p. 1432) < 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.

8.309.1 Detailed Description

Instantiates com.rti.dds.util.Sequence (p. 1432) < char >.

Instantiates:

<<*generic*>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

char
com.rti.dds.util.Sequence (p. 1432)

8.309.2 Constructor & Destructor Documentation

8.309.2.1 WcharSeq ()

Constructs an empty sequence of wide characters with an initial maximum of zero.

8.309.2.2 WcharSeq (int *initialMaximum*)

Constructs an empty sequence of wide characters with the given initial maximum.

8.309.2.3 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

8.310 WireProtocolQosPolicy Class Reference

Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 629).

Inheritance diagram for WireProtocolQosPolicy::

Public Attributes

- int **participant_id**

  *A value used to distinguish among different participants belonging to the same **domain** (p. 317) 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** (p. 317) participant.*

- int **rtps_app_id**

  *The RTPS App ID of the **domain** (p. 317) participant.*

- int **rtps_instance_id**

  *The RTPS Instance ID of the **com.rti.dds.domain.DomainParticipant** (p. 629).*

- 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_IP**

  Kind of auto mechanism used to calculate the GUID prefix.

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.

8.310.1 Detailed Description

Specifies the wire-protocol-related attributes for the **com.rti.dds.domain.DomainParticipant** (p. 629).

Entity:

com.rti.dds.domain.DomainParticipant (p. 629)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **NO** (p. 98)

8.310.2 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. 571) and **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy** (p. 504) configure RTPS and reliability properties on a per **com.rti.dds.publication.DataWriter** (p. 538) or **com.rti.dds.subscription.DataReader** (p. 473) 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. 629) 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. 629) must have a unique value for this prefix relative to all other participants in its **domain** (p. 317). To make it easier to control how this 96-bit value is generated, RTI Connext divides it into three integers: a *host ID*, the value of which is based on the identity of the identity of the machine on which the participant is executing; an *application ID*, the value of which is based on the process or task in which the particpant is contained; and an *instance ID*, which identifies the participant itself.

This QoS policy provides you with a choice of algorithms for generating these values automatically. In case none of these algorithms suit your needs, you may also choose to specify some or all of them yourself.

The following three fields:

- **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_host_id** (p. 1715)
- **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1716)
- **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1716)

comprise the GUID prefix and by default are set to **com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID** (p. 1714). The meaning of this flag depends on the value assigned to the **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) field.

Depending on the **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) value, there are two different scenarios:

1. In the default and most common scenario, **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) is set to **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_IP** (p. 132). Doing so, each field is interpreted 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.
   - **rtps_app_id:** The process (or task) ID is assigned.
   - **rtps_instance_id:** A counter is assigned that is incremented per new participant.

   NOTE: If the IP assigned to the interface is not unique within the network (for instance, if it is not configured), it is possible that the GUID (specifically, the rtps_host_id portion) may also not be unique.
2. In this situation, RTI Connext provides a different value for rtps_auto_id_kind: **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 133). As the name suggests, this alternative mechanism uses the MAC address instead of the IPv4 address. Since the MAC address size is up to 64 bits, the logical mapping of the host information, the application ID, and the instance identifiers has to change.

   Note to Solaris Users: To use **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC** (p. 133), you must run the RTI Connext application while logged in as root.

Using **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_-AUTO_ID_FROM_MAC** (p. 133), the default value of each field is interpreted 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. In both scenarios, you can change the value of each field independently.

If **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_-AUTO_ID_FROM_MAC** (p. 133) is used, the rtps_instance_id has been logically split into two parts: 24 bits for the process/task ID and 8 bits for the per new participant counter. To give to users the ability to manually set the two parts independently, a bit-field mechanism has been introduced for the rtps_instance_id field when it is used in combination with **com.rti.dds.infrastructure.WireProtocolQosPolicyAutoKind.RTPS_-AUTO_ID_FROM_MAC** (p. 133). If one of the two parts is set to 0, only this part will be handled by RTI Connext and you will be able to handle the other one manually.

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. 133).

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. 1709) using one of the options shown below.

Third, create the **com.rti.dds.domain.DomainParticipant** (p. 629) 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. 133) 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. 133) 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. 133) 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.

8.310.3 Member Data Documentation

8.310.3.1 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. 1715), **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1716), **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1716), **com.rti.dds.infrastructure.DataWriterProtocolQosPolicy.rtps_object_id** (p. 573) or **com.rti.dds.infrastructure.DataReaderProtocolQosPolicy.rtps_object_id** (p. 506) 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. 679), **com.rti.dds.publication.DataWriter.get_qos** (p. 544) or **com.rti.dds.subscription.DataReader.get_qos** (p. 482) will never have this value.

QoS:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_host_id (p. 1715) **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_app_id** (p. 1716) **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_instance_id** (p. 1716)

8.310.3.2 int participant_id

A value used to distinguish among different participants belonging to the same **domain** (p. 317) 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. 522) or **com.rti.dds.publication.DataWriterQos.unicast** (p. 593)).

For more information on port mapping, please refer to **com.rti.dds.infrastructure.RtpsWellKnownPorts_t** (p. 1396).

Each **com.rti.dds.domain.DomainParticipant** (p. 629) in the same **domain** (p. 317) 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. 1396).

See also:

com.rti.dds.infrastructure.Entity.enable() (p. 915)

8.310.3.3 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**] **RtpsWellKnownPorts_t.INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 131)

8.310.3.4 int rtps_host_id

The RTPS Host ID of the **domain** (p. 317) participant.

A machine/operating system specific host ID that is unique in the **domain** (p. 317).

[**default**] **com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID** (p. 1714). The default value is interpreted as follows:

If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) is equal to *RTPS_AUTO_ID_FROM_IP* (the default value), 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. 1717) is equal to *RTPS_AUTO_ID_FROM_MAC*, 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.

[range] [0,0xffffffff]

8.310.3.5 int rtps_app_id

The RTPS App ID of the **domain** (p. 317) 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** (p. 317) can distinguish between them.

[default] **com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID** (p. 1714). The default value is interpreted as follows:

If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) is equal to *RTPS_AUTO_ID_FROM_IP* (default value for this field) the value will be the process (or task) ID.

If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) is equal to *RTPS_AUTO_ID_FROM_MAC* the value will be the last 32 bits of the MAC address assigned to the *first* up and running interface of the host machine. **[range]** [0,0xffffffff]

8.310.3.6 int rtps_instance_id

The RTPS Instance ID of the **com.rti.dds.domain.DomainParticipant** (p. 629).

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** (p. 317) can distinguish between them.

[default] **com.rti.dds.infrastructure.WireProtocolQosPolicy.RTPS_AUTO_ID** (p. 1714). The default value is interpreted as follows:

If **com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind** (p. 1717) is equal to *RTPS_AUTO_ID_FROM_IP* (the default value), 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. 1717) is equal to *RTPS_AUTO_ID_FROM_MAC*, 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.

[**range**] [0,0xffffffff] **NOTE:** If you use *DDS_RTPS_AUTO_ID_FROM_MAC* 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`)

8.310.3.7 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. 1396) will be detected at this time, and the enable operation will fail.

[**default**] **RtpsReservedPortKind.MASK_DEFAULT** (p. 129)

8.310.3.8 WireProtocolQosPolicyAutoKind rtps_auto_id_kind = WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_IP

Kind of auto mechanism used to calculate the GUID prefix.

[**default**] RTPS_AUTO_ID_FROM_IP

8.311 WireProtocolQosPolicyAutoKind Class Reference

Kind of auto mechanism used to calculate the GUID prefix.

Inheritance diagram for WireProtocolQosPolicyAutoKind::

Static Public Attributes

- static final **WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_IP** = new **WireProtocolQosPolicyAutoKind**("RTPS_AUTO_ID_FROM_IP", 0)

  Kind of auto mechanism used to calculate the GUID prefix.

- static final **WireProtocolQosPolicyAutoKind RTPS_AUTO_ID_FROM_MAC** = new **WireProtocolQosPolicyAutoKind**("RTPS_AUTO_ID_FROM_MAC", 1)

  Kind of auto mechanism used to calculate the GUID prefix.

8.311.1 Detailed Description

Kind of auto mechanism used to calculate the GUID prefix.

See also:

com.rti.dds.infrastructure.WireProtocolQosPolicy.rtps_auto_id_kind (p. 1717)

8.312 WriteParams_t Class Reference

<<*eXtension*>> *(p. 270)* Input parameters for writing with com.rti.dds.topic.example.FooDataWriter.write_w_params, com.rti.dds.topic.example.FooDataWriter.dispose_w_params, com.rti.dds.topic.example.FooDataWriter.register_instance_w_params, com.rti.dds.topic.example.FooDataWriter.unregister_instance_w_params

Inherits Struct.

Public Member Functions

- **WriteParams_t** ()

  Construct a new ***WriteParams_t*** *(p. 1719).*

- **WriteParams_t** (**SampleIdentity_t identity**, **Time_t source_timestamp**, **Cookie_t cookie**, **InstanceHandle_t handle**)

  Construct a new ***WriteParams_t*** *(p. 1719) with the given members.*

Public Attributes

- final **SampleIdentity_t identity**

  Identity of the sample.

- final **Time_t source_timestamp**

  Source timestamp upon write.

- final **Cookie_t cookie** = new **Cookie_t**()

  Octet sequence identifying written data sample.

- final **InstanceHandle_t handle** = **InstanceHandle_t.HANDLE_NIL**

  Instance handle.

8.312.1 Detailed Description

<<*eXtension*>> *(p. 270)* Input parameters for writing with com.rti.dds.topic.example.FooDataWriter.write_w_params, com.rti.dds.topic.example.FooDataWriter.dispose_w_params, com.rti.dds.topic.example.FooDataWriter.register_instance_w_params, com.rti.dds.topic.example.FooDataWriter.unregister_instance_w_params

8.312.2 Constructor & Destructor Documentation

8.312.2.1 WriteParams_t ()

Construct a new **WriteParams_t** (p. 1719).

8.312.2.2 WriteParams_t (SampleIdentity_t *identity*, Time_t *source_timestamp*, Cookie_t *cookie*, InstanceHandle_t *handle*)

Construct a new **WriteParams_t** (p. 1719) with the given members.

8.312.3 Member Data Documentation

8.312.3.1 final SampleIdentity_t identity

Initial value:

```
new SampleIdentity_t(SampleIdentity_t.AUTO_SAMPLE_IDENTITY)
```

Identity of the sample.

Identifies the sample being written. The identity consist of a pair (Vitual Writer GUID, SequenceNumber).

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. 572).
- 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 you try to write a sample with a sequence number smaller or equal to the last sequence number, the write operation will fail.

[default] **SampleIdentity_t.AUTO_SAMPLE_IDENTITY** (p. 1402).

8.312.3.2 final Time_t source_timestamp

Initial value:

```
new Time_t(Time_t.TIME_INVALID_SEC, Time_t.TIME_INVALID_NSEC)
```

Source timestamp upon write.

Specifies the source timestamp that will be available to the **com.rti.dds.subscription.DataReader** (p. 473) objects by means of the `source_timestamp` attribute within the **com.rti.dds.subscription.SampleInfo** (p. 1404).

[**default**] com.rti.dds.infrastructure.Time_t.INVALID.

8.312.3.3 final Cookie_t cookie = new Cookie_t()

Octet sequence identifying written data sample.

The maximum size of the cookie is configurable using the field **com.rti.dds.infrastructure.DataWriterResourceLimitsQosPolicy.cookie_max_length** (p. 601).

[**default**] Empty sequence (zero-length).

8.312.3.4 final InstanceHandle_t handle = InstanceHandle_t.HANDLE_NIL

Instance handle.

Either the handle returned by a previous call to com.rti.dds.topic.example.FooDataWriter.register_instance, or else the special value **InstanceHandle_t.HANDLE_NIL** (p. 1082).

[**default**] **InstanceHandle_t.HANDLE_NIL** (p. 1082)

8.313 WriterDataLifecycleQosPolicy Class Reference

Controls how a **com.rti.dds.publication.DataWriter** (p. 538) 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. 538) unregisters an instance by means of the unregister operations.*

- final **Duration_t autopurge_unregistered_instances_delay**

  <<**eXtension**>> (p. 270) *Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 538) will maintain information regarding an instance once it has unregistered the instance.*

8.313.1 Detailed Description

Controls how a **com.rti.dds.publication.DataWriter** (p. 538) handles the lifecycle of the instances (keys) that it is registered to manage.

Entity:

com.rti.dds.publication.DataWriter (p. 538)

Properties:

RxO (p. 97) = N/A
Changeable (p. 98) = **YES** (p. 98)

8.313.2 Usage

This policy determines how the **com.rti.dds.publication.DataWriter** (p. 538) 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.DataWriter** (p. 538) or implicitly registered by directly writing the data).

Since the deletion of a DataWriter automatically unregisters all data instances it manages, the setting of the `autodispose_unregistered_instances` flag will only determine whether instances are ultimately disposed when the **com.rti.dds.publication.DataWriter** (p. 538) is deleted either directly by means of the **com.rti.dds.publication.Publisher.delete_datawriter** (p. 1287) operation or indirectly as a consequence of calling **com.rti.dds.publication.Publisher.delete_contained_entities** (p. 1299) or **com.rti.dds.domain.DomainParticipant.delete_contained_entities** (p. 691) that contains the DataWriter.

You may use com.rti.dds.topic.example.FooDataWriter.unregister_instance to indicate that the **com.rti.dds.publication.DataWriter** (p. 538) no longer wants to send data for a **com.rti.dds.topic.Topic** (p. 1545).

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. 538) 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. 1216)), DataWriters may want to relinquish their ownership of a particular instance of the Topic to allow other DataWriters to send updates for the value of that instance regardless of Ownership 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.

8.313.3 Member Data Documentation

8.313.3.1 boolean autodispose_unregistered_instances

Boolean flag that controls the behavior when the **com.rti.dds.publication.DataWriter** (p. 538) unregisters an instance by means of the unregister operations.

- true (default)

  The **com.rti.dds.publication.DataWriter** (p. 538) will dispose the instance each time it is unregistered. The behavior is identical to explicitly calling one of the `dispose` operations on the instance prior to calling the `unregister` operation.

- false

  The **com.rti.dds.publication.DataWriter** (p. 538) will not dispose the instance. The application can still call one of the `dispose` operations prior

to unregistering the instance and accomplish the same effect.

[default] true

8.313.3.2 final Duration_t autopurge_unregistered_instances_delay

<<***eXtension***>> *(p. 270)* Maximum duration for which the **com.rti.dds.publication.DataWriter** (p. 538) will maintain information regarding an instance once it has unregistered the instance.

After this time elapses, the **com.rti.dds.publication.DataWriter** (p. 538) will purge all internal information regarding the instance, including historical samples.

When the duration is zero, the instance is purged as soon as all the samples have been acknowledged by all the live DataReaders.

[default] com.rti.dds.infrastructure.Duration_t.INFINITE (disabled)

[range] [0, 1 year] or com.rti.dds.infrastructure.Duration_t.INFINITE

8.314 WstringSeq Class Reference

Instantiates com.rti.dds.util.Sequence (p. 1432) < 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

8.314.1 Detailed Description

Instantiates com.rti.dds.util.Sequence (p. 1432) < char* >.

Instantiates:

<<***generic***>> *(p. 271)* **com.rti.dds.util.Sequence** (p. 1432)

See also:

char
com.rti.dds.infrastructure.StringSeq (p. 1470)
com.rti.dds.util.Sequence (p. 1432)

8.314.2 Constructor & Destructor Documentation

8.314.2.1 WstringSeq ()

Constructs an empty sequence of wide strings with an initial maximum of zero.

8.314.2.2 WstringSeq (int *initialMaximum*)

Constructs an empty sequence of wide strings with the given initial maximum.

8.314.2.3 WstringSeq (Collection *strings*)

Constructs a new sequence containing the given wide strings.

Parameters:

strings the initial contents of this sequence

8.314.3 Member Function Documentation

8.314.3.1 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)

8.314.3.2 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** (p. 290), 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. 160) 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

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorld.idl]

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

9.2 HelloWorldDataReader.java

9.2.1 User Data Type Support

Files generated by **rtiddsgen** (p. 290) that implement the type specific APIs required by the DDS specification, as described in the **User Data Type Support** (p. 160), where:

- ^ FooTypeSupport = HelloWorldTypeSupport
- ^ FooDataWriter = HelloWorldDataWriter
- ^ FooDataReader = HelloWorldDataReader

9.2.1.1 HelloWorldTypeSupport.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldTypeSupport.java]

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

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext 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.CdrHelper;
import com.rti.dds.domain.DomainParticipant;
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.KeyHash_t;
import com.rti.dds.topic.TypeSupportImpl;
import com.rti.dds.topic.TypeSupportType;
import com.rti.dds.util.Sequence;
import com.rti.dds.topic.DefaultEndpointData;
import com.rti.dds.infrastructure.RETCODE_ERROR;


import com.rti.dds.topic.TypeSupportParticipantInfo;
import com.rti.dds.topic.TypeSupportEndpointInfo;
import com.rti.dds.typecode.TypeCode;
```

```
import com.rti.dds.infrastructure.Copyable;


public class HelloWorldTypeSupport extends TypeSupportImpl {
    // ---------------------------------------------------------------------
    // Private Fields
    // ---------------------------------------------------------------------

    private static final 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 String get_type_name() {
        return _singleton.get_type_nameI();
    }

    public static void register_type(DomainParticipant participant,
                                     String type_name) {
        _singleton.register_typeI(participant, type_name);
    }

    public static void unregister_type(DomainParticipant participant,
                                       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 Object create_data() {
        return new HelloWorld();
    }

    public void destroy_data(Object data) {
        return;
```

```
}

public Object create_key() {
    return new HelloWorld();
}

public void destroy_key(Object key) {
    return;
}


public Object copy_data(Object destination, Object source) {
    HelloWorld typedDst = (HelloWorld) destination;
    HelloWorld typedSrc = (HelloWorld) source;

    return typedDst.copy_from(typedSrc);

}



public long get_serialized_sample_max_size(Object endpoint_data,boolean include_encapsulation,short encapsulation_i
    long origAlignment = currentAlignment;

    long encapsulation_size = currentAlignment;


    if(include_encapsulation) {
      if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
          throw new RETCODE_ERROR("Unsupported encapsulation");
      }

      encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
      encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
      encapsulation_size -= currentAlignment;
      currentAlignment = 0;
      origAlignment = 0;

    }



    currentAlignment +=  CdrPrimitiveType.getStringMaxSizeSerialized(currentAlignment, ((128)) + 1);

    if (include_encapsulation) {
        currentAlignment += encapsulation_size;
    }

    return currentAlignment - origAlignment;
}


public long get_serialized_sample_min_size(Object endpoint_data,boolean include_encapsulation,short encapsulation_i
    long origAlignment = currentAlignment;

    long encapsulation_size = currentAlignment;
```

```
        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }

            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;

        }


        currentAlignment +=  CdrPrimitiveType.getStringMaxSizeSerialized(currentAlignment, 1);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }

        return currentAlignment - origAlignment;
    }


    public long get_serialized_sample_size(
        Object endpoint_data, boolean include_encapsulation,
        short encapsulation_id, long current_alignment,
        Object sample)
    {
        long origAlignment = current_alignment;

        long encapsulation_size = current_alignment;
        HelloWorld typedSrc = (HelloWorld) sample;

        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }


            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= current_alignment;
            current_alignment = 0;
            origAlignment = 0;

        }


        current_alignment += CdrPrimitiveType.getStringSerializedSize(current_alignment, typedSrc.msg);

        if (include_encapsulation) {
            current_alignment += encapsulation_size;
        }

        return current_alignment - origAlignment;
```

```
    }


    public long get_serialized_key_max_size(
        Object endpoint_data,
        boolean include_encapsulation,
        short encapsulation_id,
        long currentAlignment)
    {

        long encapsulation_size = currentAlignment;

        long origAlignment = currentAlignment;


        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }


            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;

        }


    currentAlignment += get_serialized_sample_max_size(
                            endpoint_data,false,encapsulation_id,currentAlignment);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }


        return currentAlignment - origAlignment;
    }


    public void serialize(Object endpoint_data,Object src, CdrOutputStream dst,boolean serialize_encapsulation,
                          short encapsulation_id, boolean serialize_sample, Object endpoint_plugin_qos) {
        int position = 0;


        if(serialize_encapsulation) {
            dst.serializeAndSetCdrEncapsulation(encapsulation_id);;


            position = dst.resetAlignment();

        }


        if(serialize_sample) {
```

```
HelloWorld typedSrc = (HelloWorld) src;

        dst.writeString(typedSrc.msg,(128));

        }


        if (serialize_encapsulation) {
          dst.restoreAlignment(position);
        }

    }


    public void serialize_key(
        Object endpoint_data,
        Object src,
        CdrOutputStream dst,
        boolean serialize_encapsulation,
        short encapsulation_id,
        boolean serialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;

        if (serialize_encapsulation) {
            dst.serializeAndSetCdrEncapsulation(encapsulation_id);


            position = dst.resetAlignment();

        }

        if (serialize_key) {
HelloWorld typedSrc = (HelloWorld) src;

            serialize(endpoint_data, src, dst, false, CdrEncapsulation.CDR_ENCAPSULATION_ID_CDR_BE, true, e

        }


        if (serialize_encapsulation) {
            dst.restoreAlignment(position);
        }

    }


    public Object deserialize_sample(
        Object endpoint_data,
        Object dst,
        CdrInputStream src, boolean deserialize_encapsulation,
        boolean deserialize_sample,
        Object endpoint_plugin_qos)
    {
        int position = 0;
```

```
        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();


            position = src.resetAlignment();

        }

        if(deserialize_sample) {
HelloWorld typedDst = (HelloWorld) dst;

            typedDst.msg = src.readString();

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }


        return dst;
    }



    public Object deserialize_key_sample(
        Object endpoint_data,
        Object dst,
        CdrInputStream src,
        boolean deserialize_encapsulation,
        boolean deserialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;


        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();


            position = src.resetAlignment();

        }

        if(deserialize_key) {
HelloWorld typedDst = (HelloWorld) dst;


        deserialize_sample(endpoint_data, dst, src, false, true, endpoint_plugin_qos);

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
```

```
        }


        return dst;
    }


    public void skip(Object endpoint_data,
                     CdrInputStream src,
                     boolean skip_encapsulation,
                     boolean skip_sample,
                     Object endpoint_plugin_qos)
    {
        int position = 0;


        if (skip_encapsulation) {
            src.skipEncapsulation();


            position = src.resetAlignment();

        }

        if (skip_sample) {

            src.skipString();

        }


        if (skip_encapsulation) {
            src.restoreAlignment(position);
        }

    }

    public Object serialized_sample_to_key(
        Object endpoint_data,
        Object sample,
        CdrInputStream src,
        boolean deserialize_encapsulation,
        boolean deserialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;


        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();

            position = src.resetAlignment();

        }

        if (deserialize_key) {
HelloWorld typedDst = (HelloWorld) sample;
```

```
            deserialize_sample(
                endpoint_data, sample, src, false,
                true, endpoint_plugin_qos);

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }


        return sample;
    }



    // --------------------------------------------------------------------
    // Callbacks
    // --------------------------------------------------------------------

    public Object on_participant_attached(Object registration_data,
                                          TypeSupportParticipantInfo participant_info,
                                          boolean top_level_registration,
                                          Object container_plugin_context,
                                          TypeCode type_code) {
        return super.on_participant_attached(
            registration_data, participant_info, top_level_registration,
            container_plugin_context, type_code);
    }

    public void on_participant_detached(Object participant_data) {
        super.on_participant_detached(participant_data);
    }

    public Object on_endpoint_attached(Object participantData,
                                       TypeSupportEndpointInfo endpoint_info,
                                       boolean top_level_registration,
                                       Object container_plugin_context) {
        return super.on_endpoint_attached(
              participantData,  endpoint_info,
              top_level_registration, container_plugin_context);
    }

    public void on_endpoint_detached(Object endpoint_data) {
        super.on_endpoint_detached(endpoint_data);
    }

    // --------------------------------------------------------------------
    // Protected Methods
    // --------------------------------------------------------------------

    protected DataWriter create_datawriter(long native_writer,
                                           DataWriterListener listener,
                                           int mask) {
```

```
        return new HelloWorldDataWriter(native_writer, listener, mask, this);

    }

    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

    }

    protected HelloWorldTypeSupport(boolean enableKeySupport) {

        super(TYPE_NAME, enableKeySupport,HelloWorldTypeCode.VALUE,HelloWorld.class,TypeSupportType.TST_STR
    }
}
```

9.2.1.2 HelloWorldDataWriter.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldDataWriter.java]

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

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext 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);
    }
}
```

9.2.1.3 HelloWorldDataReader.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldDataReader.java]

```
/*
```

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

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext 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);
    }

}
```

9.3 HelloWorldPublisher.java

9.3.1 RTI Connext Publication Example

The publication example generated by **rtiddsgen** (p. 290). The example has been modified slightly to update the sample value.

9.3.1.1 HelloWorldPublisher.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldPublisher.java]

```
/* HelloWorldPublisher.java

   A publication of data of type HelloWorld

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

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

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

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       java HelloWorldSubscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       java HelloWorldPublisher <domain_id> <sample_count>

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

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

   Example:

       To run the example application on domain <domain_id>:

       Ensure that $(NDDSHOME)/lib/<arch> is on the dynamic library path for
       Java.

        On Unix:
             add $(NDDSHOME)/lib/<arch> to the 'LD_LIBRARY_PATH' environment
             variable

        On Windows:
             add $(NDDSHOME)\lib\<arch> to the 'Path' environment variable
```

```
       Run the Java applications:

        java -Djava.ext.dirs=$(NDDSHOME)/class HelloWorldPublisher <domain_id>

        java -Djava.ext.dirs=$(NDDSHOME)/class HelloWorldSubscriber <domain_id>


modification history
------------ -------
*/

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

import com.rti.dds.domain.*;
import com.rti.dds.infrastructure.*;
import com.rti.dds.publication.*;
import com.rti.dds.topic.*;
import com.rti.ndds.config.*;

// ===========================================================================

public class HelloWorldPublisher {
    // --------------------------------------------------------------------
    // Public Methods
    // --------------------------------------------------------------------

    public static void main(String[] args) {
        // --- Get domain ID --- //
        int domainId = 0;
        if (args.length >= 1) {
            domainId = Integer.valueOf(args[0]).intValue();
        }

        // -- Get max loop count; 0 means infinite loop --- //
        int sampleCount = 0;
        if (args.length >= 2) {
            sampleCount = Integer.valueOf(args[1]).intValue();
        }

        /* Uncomment this to turn on additional logging
        Logger.get_instance().set_verbosity_by_category(
                LogCategory.NDDS_CONFIG_LOG_CATEGORY_API,
                LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        publisherMain(domainId, sampleCount);
    }


    // --------------------------------------------------------------------
    // Private Methods
    // --------------------------------------------------------------------
```

```
// --- Constructors: -----------------------------------------------------

private HelloWorldPublisher() {
    super();
}


// -----------------------------------------------------------------------

private static void publisherMain(int domainId, int sampleCount) {

    DomainParticipant participant = null;
    Publisher publisher = null;
    Topic topic = null;
    HelloWorldDataWriter writer = null;

    try {
        // --- Create participant --- //

        /* To create participant with default QoS,
           use DomainParticipantFactory.DomainParticipantFactory.
                   participant.get_default_publisher_qos() instead */
        participant = DomainParticipantFactory.TheParticipantFactory.
            create_participant(
                domainId, DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null /* listener */, StatusKind.STATUS_MASK_NONE);

        // --- Create publisher --- //

        /* To customize publisher QoS, use
           participant.get_default_publisher_qos() instead */
        publisher = participant.create_publisher(
            DomainParticipant.PUBLISHER_QOS_DEFAULT, null /* listener */,
            StatusKind.STATUS_MASK_NONE);

        // --- Create topic --- //

        /* Register type before creating topic */
        String typeName = HelloWorldTypeSupport.get_type_name();
        HelloWorldTypeSupport.register_type(participant, typeName);

        /* To customize topic QoS, use
           participant.get_default_topic_qos() instead */
        topic = participant.create_topic(
            "Example HelloWorld",
            typeName, DomainParticipant.TOPIC_QOS_DEFAULT,
            null /* listener */, StatusKind.STATUS_MASK_NONE);

        // --- Create writer --- //

        /* To customize data writer QoS, use
           publisher.get_default_datawriter_qos() instead */
        writer = (HelloWorldDataWriter)
            publisher.create_datawriter(
                topic, Publisher.DATAWRITER_QOS_DEFAULT,
                null /* listener */, StatusKind.STATUS_MASK_NONE);
```

```
            // --- Write --- //

            /* Create data sample for writing */
            HelloWorld instance = new HelloWorld();

            InstanceHandle_t instance_handle = InstanceHandle_t.HANDLE_NIL;
            /* For data type that has key, if the same instance is going to be
               written multiple times, initialize the key here
               and register the keyed instance prior to writing */
            //instance_handle = writer.register_instance(instance);

            final long sendPeriodMillis = 4 * 1000; // 4 seconds

            for (int count = 0;
                 (sampleCount == 0) || (count < sampleCount);
                 ++count) {
                System.out.println("Writing HelloWorld, count " + count);

                /* Modify the instance to be written here */
                instance.msg = "Hello World! (" + count + ")";

                /* Write data */
                writer.write(instance, InstanceHandle_t.HANDLE_NIL);
                try {
                    Thread.sleep(sendPeriodMillis);
                } catch (InterruptedException ix) {
                    System.err.println("INTERRUPTED");
                    break;
                }
            }

            //writer.unregister_instance(instance, instance_handle);

        } finally {

            // --- Shutdown --- //

            if(participant != null) {
                participant.delete_contained_entities();

                DomainParticipantFactory.TheParticipantFactory.
                    delete_participant(participant);
            }
            /* RTI Connext provides finalize_instance()
               method for people who want to release memory used by the
               participant factory singleton. Uncomment the following block of
               code for clean destruction of the participant factory
               singleton. */
            //DomainParticipantFactory.finalize_instance();
        }
    }
}
```

9.4 HelloWorldSeq.java

9.4.1 Programming Language Type Description

The following programming language specific type representation is generated by **rtiddsgen** (p. 290) for use in application code, where:

- Foo = HelloWorld
- FooSeq = HelloWorldSeq

9.4.1.1 HelloWorld.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorld.java]

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

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


import com.rti.dds.infrastructure.*;
import com.rti.dds.infrastructure.Copyable;

import java.io.Serializable;
import com.rti.dds.cdr.CdrHelper;


public class HelloWorld implements Copyable, Serializable
{

    public String msg = ""; /* maximum length = (128) */


    public HelloWorld() {

    }


    public HelloWorld(HelloWorld other) {

        this();
        copy_from(other);
    }



    public static Object create() {
        return new HelloWorld();
```

```
}

public boolean equals(Object o) {

    if (o == null) {
        return false;
    }



    if(getClass() != o.getClass()) {
        return false;
    }

    HelloWorld otherObj = (HelloWorld)o;



    if(!msg.equals(otherObj.msg)) {
        return false;
    }

    return true;
}

public int hashCode() {
    int __result = 0;

    __result += msg.hashCode();

    return __result;
}


public Object copy_from(Object src) {


    HelloWorld typedSrc = (HelloWorld) src;
    HelloWorld typedDst = this;

    typedDst.msg = typedSrc.msg;

    return this;
}



public String toString(){
    return toString("", 0);
}


public String toString(String desc, int indent) {
    StringBuffer strBuffer = new StringBuffer();


    if (desc != null) {
```

```
            CdrHelper.printIndent(strBuffer, indent);
            strBuffer.append(desc).append(":\n");
        }


        CdrHelper.printIndent(strBuffer, indent+1);
        strBuffer.append("msg: ").append(msg).append("\n");

        return strBuffer.toString();
    }

}
```

9.4.1.2 HelloWorldSeq.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldSeq.java]

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

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext 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 {
    // -----------------------------------------------------------------------
    // 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);
}


// --- From Copyable: -------------------------------------------------

public Object copy_from(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;
    }

}
```

9.5 HelloWorldSubscriber.java

9.5.1 RTI Connext Subscription Example

The unmodified subscription example generated by **rtiddsgen** (p. 290).

9.5.1.1 HelloWorldPublisher.java

[$(NDDSHOME)/example/JAVA/helloWorld/HelloWorldSubscriber.java]

```
/* HelloWorldSubscriber.java

   A publication of data of type HelloWorld

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

   rtiddsgen -language java -example <arch> .idl

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

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for with the command
       java HelloWorldSubscriber <domain_id> <sample_count>

   (3) Start the publication with the command
       java HelloWorldPublisher <domain_id> <sample_count>

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

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


   Example:

       To run the example application on domain <domain_id>:

       Ensure that $(NDDSHOME)/lib/<arch> is on the dynamic library path for
       Java.

        On UNIX systems:
             add $(NDDSHOME)/lib/<arch> to the 'LD_LIBRARY_PATH' environment
             variable

        On Windows systems:
             add %NDDSHOME%\lib\<arch> to the 'Path' environment variable


       Run the Java applications:
```

```
        java -Djava.ext.dirs=$NDDSHOME/class HelloWorldPublisher <domain_id>

        java -Djava.ext.dirs=$NDDSHOME/class HelloWorldSubscriber <domain_id>


modification history
------------ -------
*/

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

import com.rti.dds.domain.*;
import com.rti.dds.infrastructure.*;
import com.rti.dds.subscription.*;
import com.rti.dds.topic.*;
import com.rti.ndds.config.*;

// ===========================================================================

public class HelloWorldSubscriber {
    // --------------------------------------------------------------------
    // Public Methods
    // --------------------------------------------------------------------

    public static void main(String[] args) {
        // --- Get domain ID --- //
        int domainId = 0;
        if (args.length >= 1) {
            domainId = Integer.valueOf(args[0]).intValue();
        }

        // -- Get max loop count; 0 means infinite loop --- //
        int sampleCount = 0;
        if (args.length >= 2) {
            sampleCount = Integer.valueOf(args[1]).intValue();
        }


        /* Uncomment this to turn on additional logging
        Logger.get_instance().set_verbosity_by_category(
            LogCategory.NDDS_CONFIG_LOG_CATEGORY_API,
            LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        subscriberMain(domainId, sampleCount);
    }



    // --------------------------------------------------------------------
    // Private Methods
    // --------------------------------------------------------------------
```

```
    // --- Constructors: -----------------------------------------------------

    private HelloWorldSubscriber() {
        super();
    }


    // -----------------------------------------------------------------------

    private static void subscriberMain(int domainId, int sampleCount) {

        DomainParticipant participant = null;
        Subscriber subscriber = null;
        Topic topic = null;
        DataReaderListener listener = null;
        HelloWorldDataReader reader = null;

        try {

            // --- Create participant --- //

            /* To customize participant QoS, use
               DomainParticipantFactory.TheParticipantFactory.
               get_default_participant_qos() */

            participant = DomainParticipantFactory.TheParticipantFactory.
                create_participant(
                    domainId, DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                    null /* listener */, StatusKind.STATUS_MASK_NONE);
            if (participant == null) {
                System.err.println("create_participant error\n");
                return;
            }

            // --- Create subscriber --- //

            /* To customize subscriber QoS, use
               participant.get_default_subscriber_qos() */

            subscriber = participant.create_subscriber(
                DomainParticipant.SUBSCRIBER_QOS_DEFAULT, null /* listener */,
                StatusKind.STATUS_MASK_NONE);
            if (subscriber == null) {
                System.err.println("create_subscriber error\n");
                return;
            }

            // --- Create topic --- //

            /* Register type before creating topic */
            String typeName = HelloWorldTypeSupport.get_type_name();
            HelloWorldTypeSupport.register_type(participant, typeName);

            /* To customize topic QoS, use
               participant.get_default_topic_qos() */

            topic = participant.create_topic(
```

```
            "Example HelloWorld",
            typeName, DomainParticipant.TOPIC_QOS_DEFAULT,
            null /* listener */, StatusKind.STATUS_MASK_NONE);
        if (topic == null) {
            System.err.println("create_topic error\n");
            return;
        }

        // --- Create reader --- //

        listener = new HelloWorldListener();

        /* To customize data reader QoS, use
           subscriber.get_default_datareader_qos() */

        reader = (HelloWorldDataReader)
            subscriber.create_datareader(
                topic, Subscriber.DATAREADER_QOS_DEFAULT, listener,
                StatusKind.STATUS_MASK_ALL);
        if (reader == null) {
            System.err.println("create_datareader error\n");
            return;
        }

        // --- Wait for data --- //

        final long receivePeriodSec = 4;

        for (int count = 0;
             (sampleCount == 0) || (count < sampleCount);
             ++count) {
            System.out.println("HelloWorld subscriber sleeping for "
                               + receivePeriodSec + " sec...");
            try {
                Thread.sleep(receivePeriodSec * 1000);  // in millisec
            } catch (InterruptedException ix) {
                System.err.println("INTERRUPTED");
                break;
            }
        }
    } finally {

        // --- Shutdown --- //

        if(participant != null) {
            participant.delete_contained_entities();

            DomainParticipantFactory.TheParticipantFactory.
                delete_participant(participant);
        }
        /* RTI Connext provides the finalize_instance()
           method for users who want to release memory used by the
           participant factory singleton. Uncomment the following block of
           code for clean destruction of the participant factory
           singleton. */
        //DomainParticipantFactory.finalize_instance();
    }
```

```
    }

    // --------------------------------------------------------------------
    // Private Types
    // --------------------------------------------------------------------

    // ====================================================================

    private static class HelloWorldListener extends DataReaderAdapter {

        HelloWorldSeq _dataSeq = new HelloWorldSeq();
        SampleInfoSeq _infoSeq = new SampleInfoSeq();

        public void on_data_available(DataReader reader) {
            HelloWorldDataReader HelloWorldReader =
                (HelloWorldDataReader)reader;

            try {
                HelloWorldReader.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 = (SampleInfo)_infoSeq.get(i);

                    if (info.valid_data) {
                        System.out.println(
                            ((HelloWorld)_dataSeq.get(i)).toString("Received",0));


                    }
                }
            } catch (RETCODE_NO_DATA noData) {
                // No data to process
            } finally {
                HelloWorldReader.return_loan(_dataSeq, _infoSeq);
            }
        }
    }
}
```