RTI Connext DDS

Core Libraries

What's New in Version 6.1.0

Printed in U.S.A. First printing.
April 2021.

Trademarks

Copy and Use Restrictions

Technical Support

Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

Contents

Introduction 1

1 High-Performance WAN Connectivity over UDP that is Secure and Scalable, Using RTI Real-Time WAN Transport and RTI Cloud Discovery Service 1

2 New C# Language Binding Allows Building Multi-Platform Connext DDS Applications for .NET 5 3

3 New Getting Started Guides in Traditional C++, Modern C++, and C# 4

4 Compressed Application Data Using Builtin Support for zlib, LZ4, and bzip2 Algorithms 4

5 Network Capture Utility, Analyzing Network Traffic for DomainParticipants - Works even with Shared Memory and Encrypted Data 5

6 Separate Durability and History Depths, Using New writer_depth Durability QoS 5

7 Coherent Access with Group Presentation QoS: Ensure a Set of Samples Sent from Multiple DataWriters within a Publisher is Received as a Cohesive Unit 6

8 Activity Context in Messages: Identify the Source of a Logged Message More Easily with Added Resources and Activities Information 8

9 New Lost, Rejected, and Dropped Statistics to Better Identify why a Subscribing Application is not Seeing Samples 10

10 New Protocol Status Statistics for DataWriters and DataReaders to more Easily Monitor Fragmented Messages 11

11 Increased Visibility into Connext DDS Applications 13

11.1 QoS 13

11.2 Sample Losses 14

11.3 Errors and Unexpected Behavior 15

11.4 Application State 18

11.5 Backtrace for Fatal Error Debugging 21

11.6 Improvements in Log Messages 23

12 Improved Control over Application Behavior 27

12.1 Instance Lifecycle Management 27

12.2 Logging Usability 30

12.3 Resource Usage Tuning 33

12.4 Improvements in Transport Functionality 35
12.5 New Character Support in Filters 38
13 Language Bindings, APIs, XML Configuration 40
13.1 Print data state (sample, view, instance states) in Modern C++ using new operator<< definitions 40
13.2 Simplify Listener lifecycle management in Modern C++ API 41
13.3 Introduced Remote Procedure Calls (RPC) - Experimental Feature 42
13.4 GetTypeCode from a definition provided in an XML configuration file using the type name 43
13.5 XML fields of type duration have unset tags default to 0 with a warning log message 43
13.6 Simple new component to process new data in a thread pool 44
14 Platform and Build Changes 44
14.1 Use of Core Libraries now supported on these additional platforms 44
14.2 Use of Core Libraries no longer supported on these platforms 45
14.3 Build applications for Linux architectures without using -lnsl flag 45
14.4 Generate build system once, and build Release and Debug configurations using fully supported multiconfiguration generators in FindRTIConnextDDS script 46
14.5 Configure the development environment in Z shell using a new script 46
15 Changes to Defaults 46
15.1 Default for WriterDataLifeCycleQosPolicy.autodispose_unregistered_instances changed to FALSE, now no longer applies during DataWriter deletion 46
15.2 Default value for max_objects_per_thread increased from 1024 to 2048 47
15.3 Default behavior for a Connext DDS application that detects an incorrect property name is now to log an error instead of a warning 47
15.4 Changes to Default Stack size for INTEGRITY Platforms 47
16 Performance Improvements 48
16.1 Improved support for concurrency in data reception events in DataReaders in a DomainParticipant 48
16.2 Improved performance in response to a TopicQuery issued by a DataReader 48
17 Deprecations 48
17.1 Use of refilter field in HISTORY QoS Policy no longer supported 48
17.2 CORBA Compatibility Kit has been removed in this release 49
17.3 RTI Prototyper has been deprecated in this release 49
17.4 Previous release's C# / .NET binding is deprecated 49
17.5 Support for pre-C++11 compilers is deprecated 49
17.6 -legacyPlugin option has been removed 49
17.7 DynamicData::set_buffer and DynamicData::get_estiamted_max_buffer_size APIs have been removed 49
17.8 RTI Secure WAN Transport may be deprecated in a future release 50
17.9 RTI Spreadsheet Add-in for Microsoft Excel is deprecated in this release 50

Introduction

Connext DDS 6.1.0 adds support for geographically distributed systems with a new secure WAN connectivity solution, bandwidth efficiency with data compression, and improved instance resource management capabilities. It also improves ecosystem integration with a new C# language binding based on .NET Standard 2.0.

RTI® Connext® DDS 6.1.0 is a general access release. This document highlights new platforms and improvements in the Core Libraries for 6.1.0.

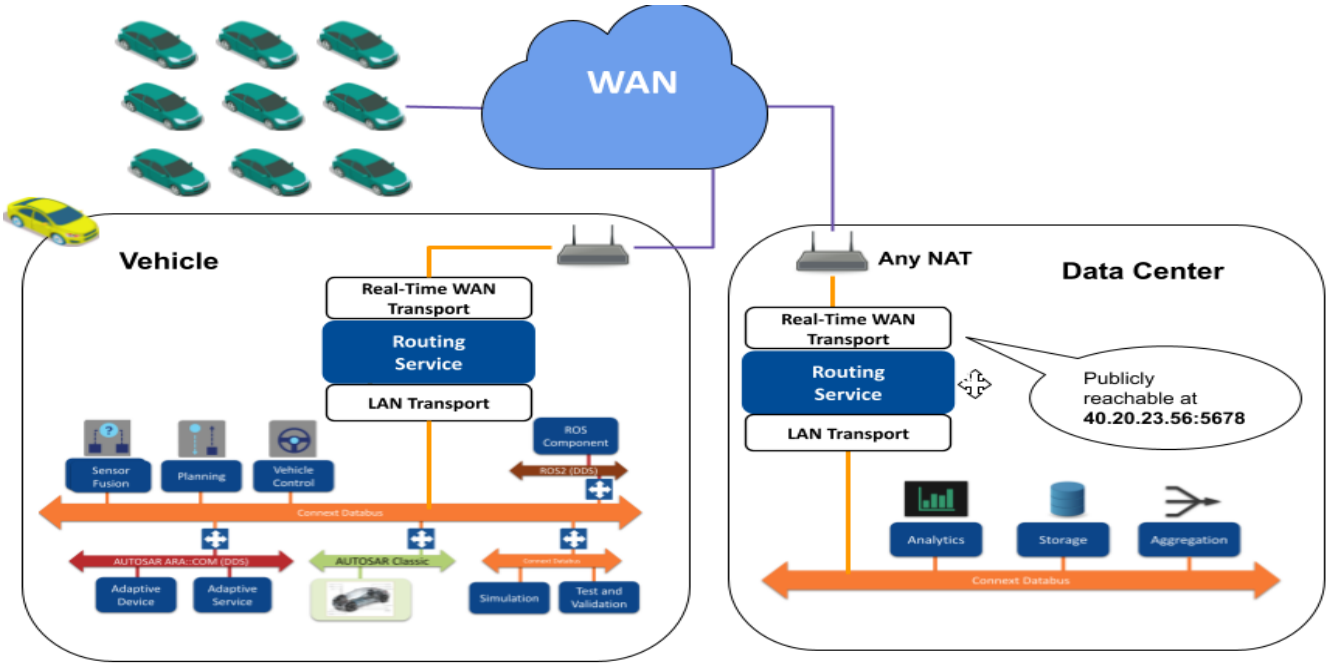
For what's *fixed* in the Core Libraries for 6.1.0, see the RTI Connext DDS Core Libraries Release Notes. For what's new and fixed in other products included in the *Connext* suite, see those products' release notes.

> **Note:** For backward compatibility information between 6.1.0 and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

1 High-Performance WAN Connectivity over UDP that is Secure and Scalable, Using RTI Real-Time WAN Transport and RTI Cloud Discovery Service

RTI Real-Time WAN Transport (RWT) is a new, smart transport that enables secure, scalable, and high-performance communication over wide area networks (WANs), including public networks. It extends *Connext DDS* capabilities to WAN environments. *Real-Time WAN Transport* uses UDP as the underlying IP transport-layer protocol to better anticipate and adapt to the challenges of diverse network conditions, device mobility, and the dynamic nature of WAN system architectures.

Real-Time WAN Transport, in combination with *RTI Cloud Discovery Service*, provides a complete, seamless solution out of the box for WAN connectivity. This WAN connectivity solution, including *Real-Time WAN Transport* and *Cloud Discovery Service*, is available as an optional add-on.



An example scenario of an Edge-to-Data Center deployment for a fleet of vehicles using a Routing Service in the vehicles and in the data center. This is one of the many deployment configurations that RTI Real-Time WAN Transport supports.

Real-Time WAN Transport replaces the transport capabilities of the *Secure WAN Transport* optionally available with previous *Connext DDS* releases, and provides the following capabilities:

- **NAT (Network Address Translator) traversal:** Ability to communicate between *DomainParticipants* running in a Local Area Network (LAN) that is behind a NAT-enabled router, and *DomainParticipants* on the outside of the NAT across a WAN. This functionality is provided in combination with *Cloud Discovery Service*.
- **IP mobility:** Support for network transitions and changes in IP addresses in any of the *DomainParticipants* participating in the communication
- **Security:** Secure communications between *DomainParticipants* using *Security Plugins*

Real-Time WAN Transport does not require third-party components, such as STUN servers, or protocols like SIP to handle session establishment. Using a single API and security model, you can leverage the extensive capabilities of the *Connext DDS* framework and ecosystem, including tools and infrastructure services, even for real-time connectivity from edge to cloud and back in highly distributed systems that communicate across wide area networks.

All new applications that communicate over wide area networks using UDP should use *Real-Time WAN Transport*. This release includes *RTI Secure WAN Transport* (described in Part 6: RTI Secure WAN Transport on page 1) only for compatibility with existing applications, which should upgrade to *Real-Time WAN Transport*. RTI may not support *Secure WAN Transport* in future versions of *Connext DDS* and no longer provides it to new customers.

For more information on setting up and using the *Real-Time WAN Transport*, see the "RTI Real-Time WAN Transport" part in the *RTI Connext DDS Core Libraries User's Manual*.

2 New C# Language Binding Allows Building Multi-Platform Connext DDS Applications for .NET 5

This release includes a new C# language binding for .NET Standard 2.0, which will replace the previous binding.

Unlike the previous binding, which ran exclusively on Windows® and .NET Framework, the new binding runs on .NET Standard 2.0-compatible systems, including .NET 5, .NET Core 2+, and .NET Framework 4.6.1+; and on Linux, macOS, and Windows.

The new binding includes a new DDS API and new IDL-to-C# code generation. The API has been redesigned to follow modern C# best-practices. Some of the most significant improvements are:

- Seamless integration with Visual Studio® Code, Visual Studio, Visual Studio for Mac, and the dotnet CLI.
- Documentation is readable via IntelliSense.
- Use of .NET naming conventions and other common practices, such as properties and events.
- Use of generics to define types such as *Topic*, *DataWriter*, *DataReader*.
- Use of standard .NET types and interfaces: IEnumerable, IList, IEquatable, etc.
- Simplified entity lifecycle: entities implement IDisposable, which enables the “using” keyword.
- Value types are designed as immutable types with fluent mutators to enhance robustness.

The following is a simple Hello World subscriber:

```
using var participant = DomainParticipantFactory.Instance
   .CreateParticipant(domainId: 0);
var topic = participant.CreateTopic<Shape>("Example Shape");
var subscriber = participant.CreateSubscriber();
var reader = subscriber.CreateDataReader(topic);
reader.DataAvailable += _ =>
{
  using var samples = reader.Take();
  foreach (var sample in samples.ValidData())
  {
    Console.WriteLine($"Received {sample}");
```

```
    }
};
// ...
```

The first release of this binding is distributed as a separate RTI package and via the NuGet package manager. It is not yet included in the *RTI Connext DDS Professional* package.

The previous .NET binding is still available, but deprecated, and will be removed in a future release.

For more information, see the RTI Connext DDS Getting Started Guide, and select C#. The API reference is available on the RTI Community Portal.

3 New Getting Started Guides in Traditional C++, Modern C++, and C#

This release presents a brand new *RTI Connext DDS Getting Started Guide*, available in three languages: modern C++, traditional C++, and C#. It is available on the RTI Community portal, as well as now being part of the *Connext DDS* installation (in **<installdir>/doc/manuals/connext_dds_professional/getting_started_guide**). If you or your co-workers are new to *Connext* or DDS, this is the right place to learn the fundamentals with the included hands-on exercises.

Some highlights of the guide:

- A modular approach to learning *Connext DDS* concepts that you can learn at your own pace: introduction to publish/subscribe, data types, keys and instances, QoS basics, content filtering, and discovery.
- Simple, user-friendly language, with pictures.
- Introduction to tools such as *RTI Admin Console*.
- Links to more in-depth topics in the *RTI Connext DDS Core Libraries User's Manual*, for next steps.

4 Compressed Application Data Using Builtin Support for zlib, LZ4, and bzip2 Algorithms

This release adds support for user data compression for any communication between a *DataWriter* and *DataReader*. There are three different compression algorithms currently supported: zlib, LZ4, and bzip2. This new feature will help to reduce bandwidth usage and increase throughput on networks with low capacity.

In support of this feature, a new field, **compression_settings**, has been added to the DATA_REPRESENTATION QoS Policy. This field contains the following settings:

- compression_ids: Chosen compression algorithm, such as COMPRESSION_ID_ZLIB or COMPRESSION_ID_BZIP2.

- writer_compression_level: Level of compression to use when compressing data, ranging between BEST_COMPRESSION and BEST_SPEED.
- writer_compression_threshold: Threshold, in bytes, above which a serialized sample will be eligible to be compressed.

See the "DATA_REPRESENTATION QosPolicy" section in the *RTI Connext DDS Core Libraries User's Manual* for more information.

5 Network Capture Utility, Analyzing Network Traffic for DomainParticipants - Works even with Shared Memory and Encrypted Data

This release introduces a new feature, network capture, that enables *Connext DDS* to capture the network traffic that one or more *DomainParticipants* send or receive. This feature can be used to analyze and debug communication problems between your DDS applications.

The result of capturing traffic for a *DomainParticipant* is a pcap-based file that can be opened by a packet analyzer like Wireshark. Network capture has several advantages over more general tools:

- It can capture shared memory traffic.
- It is available from all platforms that support a file system.
- It is security-friendly. The capture can include the decryption of RTPS packets.
- You can exclude user data from the capture to preserve confidentiality and reduce the file size.

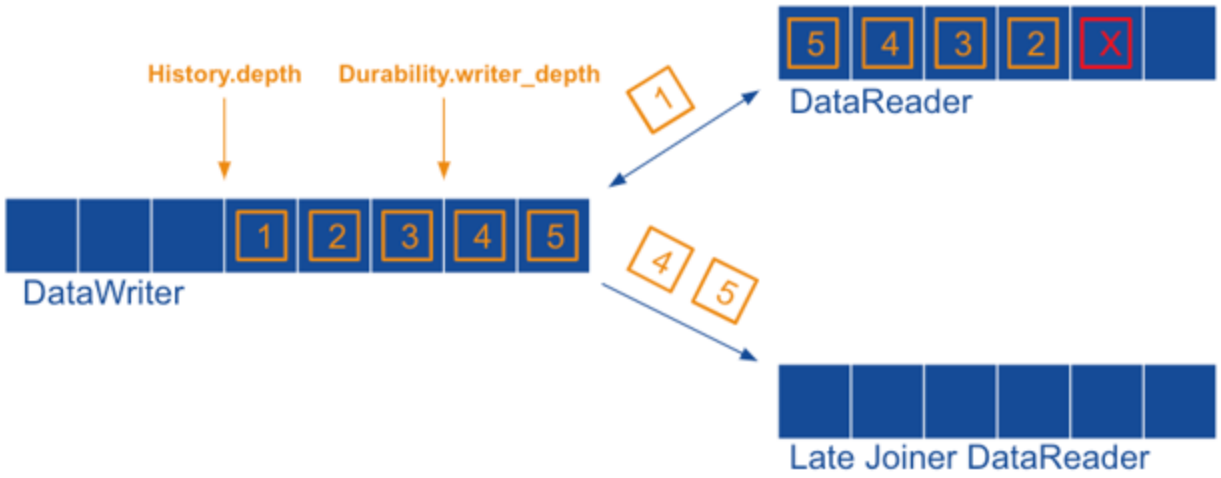
You can enable network capture and start capturing traffic for one or more *DomainParticipants* through new APIs that have been added to the C, Traditional C++, Modern C++, Java, and .NET languages. For information about the use of these APIs, please refer to the API Reference HTML documentation.

For more information about network capture, see the "Network Capture" section in the "Troubleshooting" chapter of the *RTI Connext DDS Core Libraries User's Manual*.

6 Separate Durability and History Depths, Using New writer_depth Durability QoS

In this release, it is possible to configure the reliability window (the number of samples kept in the queue for reliability purposes) separately from the durability window (the number of samples kept in the *DataWriter* queue for that are delivered to late-joining *DataReaders*). This allows an application to achieve the level of reliability that is required and still only deliver a subset of data to late-joining *DataReaders* when using a non-VOLATILE **kind** in the DURABILITY QoS Policy.

The reliability window is configured with the existing **depth** field in the HISTORY QoS Policy. The durability window is configured with a new **writer_depth** field in the DURABILITY QoS Policy.



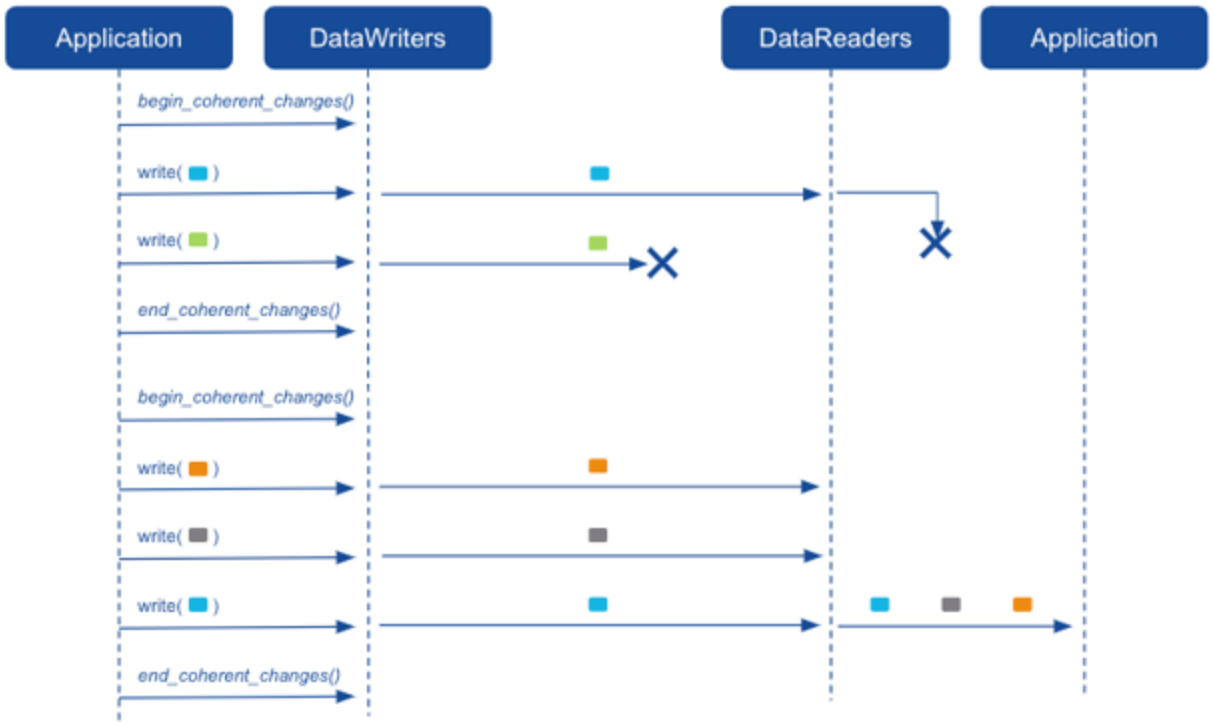
While a History.depth of 5 samples is maintained to repair losses for DataReaders that haven't acknowledged the samples yet, Durability.writer_depth is set up to send only the last two samples to late-joining DataReaders

Previously, it was not possible to configure use cases such as strict reliability while only delivering the latest state per instance to late-joining *DataReaders*. This can now be achieved by using reliable communication and setting HistoryQosPolicy.kind = KEEP_ALL and DurabilityQosPolicy.writer_depth = 1.

7 Coherent Access with Group Presentation QoS: Ensure a Set of Samples Sent from Multiple DataWriters within a Publisher is Received as a Cohesive Unit

This release adds support for coherent access with group presentation (DDS_GROUP_PRESENTATION_QOS). In previous releases, this functionality was supported only with topic or instance presentation (TOPIC_PRESENTATION_QOS or INSTANCE_PRESENTATION_QOS).

A publishing application can request that a set of DDS data samples across all the *DataWriters* within a *Publisher* be propagated in such a way that they are interpreted at the receivers' side as a cohesive set of modifications. In this case, the matching *DataReaders* will only be able to access the data after all the modifications in the set are available at the subscribing end.



By default, the first coherent set was not provided to the application because it was deemed incomplete by the DataReaders. (Different colored rectangles represent samples from different Topics.) The second coherent set was received completely by the DataReaders, and was therefore provided to the application.

As part of this feature, this release also adds a way to identify a sample as part of a coherent set by introducing a new optional field called **coherent_set_info** in the SampleInfo data structure with type DDS_CoherentSetInfo_t:

```
struct DDS_CoherentSetInfo_t {
    DDS_GUID_t group_guid;
    DDS_SequenceNumber_t coherent_set_sequence_number;
    DDS_SequenceNumber_t group_coherent_set_sequence_number;
    DDS_Boolean incomplete_coherent_set;
};
```

A group coherent set is uniquely identified by the pair (**group_guid**, **group_coherent_set_sequence_number**) where **group_guid** identifies the *Publisher*.

A topic coherent set is uniquely identified by the pair (**group_guid**, **coherent_set_sequence_number**) where **group_guid** is the *DataWriter's* **protocol.virtual_guid**.

The field **incomplete_coherent_set** is used to indicate if a sample is part of an incomplete coherent set. An incomplete coherent set is a coherent set for which some of the samples have not been received. This includes samples that are filtered by content or time on the *DataWriter* side.

By default, the samples that are received from an incomplete coherent set are dropped by the *DataReader(s)* and they are not provided to the application. By setting the new QoS parameter **subscriber_**

qos.presentation.drop_incomplete_coherent_set to FALSE, you can change this behavior and, in this case, samples from incomplete coherent sets will be provided to the application. These samples have **sample_info.coherent_set_info.incomplete_coherent_set** set to TRUE.

For more information, see "The SampleInfo Structure" and "PRESENTATION QosPolicy" in the *RTI Connext DDS Core Libraries User's Manual*.

See also: *Known Issues* in the RTI Core Libraries Release Notes.

8 Activity Context in Messages: Identify the Source of a Logged Message More Easily with Added Resources and Activities Information

The "activity context" is the information that log messages provide about the context in which an error or a warning occurs (for example, when a *DataWriter* fails to write a data sample, the log message includes information such as the topic name, the domain ID or the writer name). This release expands the information that is presented as well as the situations in which this information is available.

Previously, activity context was included in select NDDS_Config_LogPrintFormat options, such as NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT.

The activity context functionality has been expanded and enhanced:

- It has been added to the NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG option. See 11.6.3 Ability to see new activity context information available as part of NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG print format on page 24.
- It is easier to use, with an updated format.

The activity context is a group of *resources* and *activities* associated with an action such as the creation of an entity:

- A *resource* is an abstraction of an entity. It can contain attributes such as Topic or Domain ID.
- An *activity* is a general task that the resource is doing, such as "Getting QoS."

The activity context is used in two places:

- Logging: activity context is one of the **NDDS_Config_LogPrintFormat** DDS logging infrastructure formats. If a format that prints activity context is selected (see the "Message Formats" table in the section "Format of Logged Messages," in the *RTI Connext DDS Core Libraries User's Manual*), then every time *Connext DDS* logs a message, it will contain the contextual information.

- Heap monitoring: every time memory is allocated and heap monitoring is enabled, the string representation of the activity context will be associated with the allocation. This information will be available when taking the snapshot.

For example, in the creation of a *DataWriter*, the activity context will provide information about:

- Resource: the *Publisher* creating the *DataWriter*. The attributes of the publisher will be GUID, kind, name, and Domain ID.
- Activity: entity creation. It will have one parameter, entity kind, in this case a *DataWriter*.

The string representation of the above activity context would be:

```
[0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000088{E=Pu,N=TestPublisher,D=1}|CREATE Writer WITH
TOPIC TestTopic]
```

Where:

- GUID is `0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000088`
- Entity kind is `E=Pu` (for Publisher)
- Entity name is `N=TestPublisher`
- Domain ID is `D=1`
- Activity is `CREATE Writer WITH TOPIC TestTopic`

You can now also configure the attributes used in the activity context. These are the attributes that **NDDS_Config_ActivityContextAttribute** uses in the string representation of the activity context. You can configure these attributes through a mask. This mask indicates what resource attributes are used when *Connext DDS* logs a message or when the Heap Monitoring utility saves statistics for a memory allocation.

```
void NDDS_Config_ActivityContext_set_attribute_mask(
       NDDS_Config_ActivityContextAttributeKindMask attribute_mask);

enum NDDS_Config_ActivityContextAttributeKind {
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX,
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC,
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE,
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND,
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID,
 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME
}
#define NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT
#define NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE
```

For more information, see the "Activity Context" section in the *RTI Connext DDS Core Libraries User's Manual*.

9 New Lost, Rejected, and Dropped Statistics to Better Identify why a Subscribing Application is not Seeing Samples

This feature provides information through statistics about every sample that is not seen by a subscribing application. A sample that is never seen by a subscribing application is now considered one (and only one) of the following statuses at any given time:

- **Lost:** a sample that is lost will never be received. This is the same behavior as in previous releases, but now a sample can no longer be both lost and rejected at the same time.
- **Rejected:** the sample is rejected due to resource limit configuration. This is the same behavior as in previous releases, but now a sample can no longer be *both* lost and rejected. (A rejected sample can, however, *later* be reported as lost, dropped, or accepted.)
- **Dropped:** this is a new status in this release for samples dropped for non-resource limit configuration reasons such as out-of-order samples, source timestamp tolerances, KEEP_LAST history replacement, content filtering, and so on.

Lost and rejected samples are exposed through statuses (SampleLostStatus and SampleRejectedStatus) and *DataReader* listener callbacks (**on_sample_lost()** and **on_sample_rejected()**). Dropped samples are exposed through counters in the DataReaderCacheStatus and DataReaderProtocolStatus.

- Dropped sample counter

  The following counters have been added:
  - DDS_DataReaderCacheStatus:old_source_timestamp_dropped_sample_count
  - DDS_DataReaderCacheStatus:tolerance_source_timestamp_dropped_sample_count
  - DDS_DataReaderCacheStatus:ownership_dropped_sample_count
  - DDS_DataReaderCacheStatus:content_filter_dropped_sample_count
  - DDS_DataReaderCacheStatus:time_based_filter_dropped_sample_count
  - DDS_DataReaderCacheStatus:virtual_duplicate_dropped_sample_count
  - DDS_DataReaderCacheStatus:replaced_dropped_sample_count
  - DDS_DataReaderCacheStatus:expired_dropped_sample_count
  - DDS_DataReaderCacheStatus:writer_removed_batch_sample_dropped_sample_count
  - DDS_DataReaderProtocolStatus:out_of_range_rejected_sample_count

  You can find more information in the sections "DATA_READER_CACHE_STATUS" and "DATA_READER_PROTOCOL_STATUS" in the *RTI Connext DDS Core Libraries User's Manual*.

- Lost sample reasons

  The following reasons have been added:
  - DDS_LOST_BY_DESERIALIZATION_FAILURE
  - DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT
  - DDS_LOST_BY_SAMPLES_LIMIT
  - DDS_LOST_BY_DECODE_FAILURE

  You can find more information in the section "SAMPLE_LOST Status" section in the *RTI Connext DDS Core Libraries User's Manual*.

- Rejected samples reason

  The following reason has been added
  - DDS_REJECTED_BY_DECODE_FAILURE

  The following reasons have been removed
  - DDS_REJECTED_BY_UNKNOWN_INSTANCE
  - DDS_REJECTED_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT
  - DDS_REJECTED_BY_VIRTUAL_WRITERS_LIMIT
  - DDS_REJECTED_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT
  - DDS_REJECTED_BY_REMOTE_WRITERS_LIMIT

  You can find more information in the section "SAMPLE_REJECTED Status" section in the *RTI Connext DDS Core Libraries User's Manual*.

10 New Protocol Status Statistics for DataWriters and DataReaders to more Easily Monitor Fragmented Messages

The DDS_DataReaderProtocolStatus and DDS_DataWriterProtocolStatus structures have been extended to include statistics relating to fragmented data.

The following fields have been added to DDS_DataWriterProtocolStatus:

- pushed_fragment_count
- pushed_fragment_bytes
- pulled_fragment_count
- pulled_fragment_bytes

- received_nack_fragment_count
- received_nack_fragment_bytes

The following fields have been added to DDS_DataReaderProtocolStatus:

- received_fragment_count
- dropped_fragment_count
- reassembled_sample_count
- sent_nack_fragment_count
- sent_nack_fragment_bytes

These fields have also been added to the topics published by the monitoring libraries: the new *DataWriter* fields have been added to the DataWriterEntityStatistics type published by the monitoring libraries, and the new *DataReader* fields have been added to the DataReaderEntityStatistics type published by the monitoring libraries, to include the statistics related to data fragmentation. The way in which these monitoring types have been extended means that they are backwards compatible.

The following fields within DDS_DataReaderProtocolStatus did not previously work when data fragmentation was used, but now they are correct when data fragmentation is used:

- received_sample_count
- received_sample_count_change
- received_sample_bytes
- received_sample_bytes_change

In release 5.3.0, partial support was added for the following fields in DDS_DataWriterProtocolStatus; they worked when data fragmentation was used, but only when obtained for the local *DataWriter* (i.e., they did not work when obtained for a matched subscription or matched locator):

- pushed_sample_count
- pushed_sample_count_change
- pushed_sample_bytes
- pushed_sample_bytes_change
- pulled_sample_count
- pulled_sample_count_change

- pulled_sample_bytes
- pulled_sample_bytes_change

These fields are now correct when the DataWriterProtocolStatus is obtained for the local *DataWriter*, matched locator, or matched subscription.

For more information about these fields, see the "DATA_WRITER_PROTOCOL_STATUS" and "DATA_READER_PROTOCOL_STATUS" sections in the *RTI Connext DDS Core Libraries User's Manual*.

11 Increased Visibility into Connext DDS Applications

11.1 QoS

11.1.1 View current QoS of entities being used through new APIs

New APIs have been added to the C, Traditional C++, Modern C++, Java, and .NET APIs that allow top-level QoS objects to be converted into strings and printed, so that you can see the current QoS being used. Top-level QoS objects are defined as DataReaderQos, DataWriterQos, PublisherQos, SubscriberQos, TopicQos, DomainParticipantQos and DomainParticipantFactoryQos.

In C, there are three new APIs per top-level QoS object (DataWriterQos is used as an example below):

```
DDS_DataWriterQos_print(const struct DDS_DataWriterQos *self)
DDS_DataWriterQos_to_string(const struct DDS_DataWriterQos *self, char *string, DDS_
UnsignedLong *string_size)
DDS_DataWriterQos_to_string_w_params(const struct DDS_DataWriterQos *self, char *string, DDS_
UnsignedLong *string_size, const struct DDS_DataWriterQos *base, const struct DDS_
QosPrintFormat *format)
```

In Traditional C++, the same functionality is achieved through overloads:

```
DDS_DataWriterQos::print()
DDS_DataWriterQos::to_string(char *string, DDS_UnsignedLong& string_size)
DDS_DataWriterQos::to_string(char *string, DDS_UnsignedLong& string_size, const DDS_
DataWriterQos& base)
DDS_DataWriterQos::to_string(char *string, DDS_UnsignedLong& string_size, const DDS_
QosPrintFormat &format)
DDS_DataWriterQos::to_string(char *string, DDS_UnsignedLong& string_size, const DDS_
DataWriterQos &format, const DDS_QosPrintFormat &format)
```

In Modern C++, the **to_string** APIs are free-standing functions:

```
std::string to_string(const DataWriterQos& qos, const QosPrintFormat& format = QosPrintFormat
())
std::string to_string(const DataWriterQos& qos, const DataWriterQos& base, const
QosPrintFormat& format = QosPrintFormat())
std::string to_string(const DataWriterQos& qos, const qos_print_all_t& qos_print_all, const
QosPrintFormat& format = QosPrintFormat())
std::ostream& operator<<(std::ostream& out, const DataWriterQos& qos)
```

In Java, **Object.toString** is overridden, and additional overloads are available:

```
String DataWriterQos.toString()
String DataWriterQos.toString(DataWriterQos baseQos, QosPrintFormat format)
String DataWriterQos.toString(QosPrintFormat format)
String DataWriterQos.toString(DataWriterQos baseQos
```

In .NET, **Object.ToString** is overridden, and additional overloads are available:

```
String ^DataWriterQos::ToString()
String ^DataWriterQos::ToString(DataWriterQos ^base, QosPrintFormat ^format)
String ^DataWriterQos::ToString(QosPrintFormat ^format)
String ^DataWriterQos::ToString(DataWriterQos ^base)
```

For more information about the use of these APIs, please refer to the API Reference HTML documentation.

11.1.2 View QoS used in DDS Entity creation in logs using log level 'LOCAL' and category 'API'

Creating and/or setting the QoS of a DDS Entity (DDS_DomainParticipant, DDS_Topic, DDS_Publisher, DDS_DataWriter, DDS_Subscriber, DDS_DataReader) or DDS_DomainParticipantFactory, will now result in the QoS of that entity being logged. This QoS is logged in XML format with a verbosity level of LOCAL and a category of API. Only the differences between the configured QoS and the documented default for that QoS are logged.

11.2 Sample Losses

11.2.1 Detect and accept samples marked as 'removed' from a batch by a DataWriter

When the *DataReader* receives a batch, it could contain samples marked as removed by the *DataWriter*. Examples of removed samples in a batch are samples that were replaced due to KEEP_LAST_HISTORY_QOS on the *DataWriter* or samples that outlived the *DataWriter's* LifespanQosPolicy duration. By default, any sample marked as removed in a batch is dropped.

Now, each time the *DataReader* receives a sample marked as removed in a batch, a new counter, **writer_removed_batch_sample_dropped_sample_count**, in the DataReaderCacheStatus will be incremented. This way, you can now detect these removed samples.

You can also choose to accept samples marked as removed by setting the property **dds.data_reader.accept_writer_removed_batch_samples** to TRUE (by default it is set to FALSE); you can set this property via the PropertyQosPolicy (DDS Extension).

If a sample marked as removed in a batch is accepted and received by the *DataReader*, the **SampleInfo::flag** will contain the new value DDS_WRITER_REMOVED_BATCH_SAMPLE.

For more information, see the "BATCH QosPolicy" section of the *RTI Connext DDS Core Libraries User's Manual*.

11.2.2 Detect samples dropped due to deserialization errors using DDS_LOST_BY_DESERIALIZATION_FAILURE status

Previously when a sample could not be deserialized, it was dropped and a message was dropped. You could only detect this scenario by checking the log with a LoggerDevice.

Now, the sample will be lost with the new reason DDS_LOST_BY_DESERIALIZATION_FAILURE.

11.2.3 Detect when received sample is lost or rejected due to decoding errors using new statuses

There are two new statuses:

- LOST_BY_DECODE_FAILURE: When using BEST_EFFORT_RELIABILITY_QOS, a received sample was lost because it could not be decoded.
- REJECTED_BY_DECODE_FAILURE: When using RELIABLE_RELIABILITY_QOS, a received sample was rejected because it could not be decoded.

11.3 Errors and Unexpected Behavior

11.3.1 ReliableWriterCacheChangedStatus extended to include information about unacknowledged replaced samples

The ReliableWriterCacheChangedStatus structures have been extended to provide information about the unacknowledged samples that have been replaced in the *DataWriter's* cache after applying the KEEP_LAST history policy.

The following field has been added to ReliableWriterCacheChangedStatus:

replaced_unacknowledged_sample_count

The monitoring topics have also been updated to publish this information.

11.3.2 View value of field causing inconsistent configuration for READER_DATA_LIFECYCLE QoS Policy as part of error messages

The error messages that were reported when an inconsistent configuration was set for the READER_DATA_LIFECYCLE QoS Policy only showed the name of the field causing the inconsistency. Now these error messages also show the value of the field that is causing the inconsistency and the values accepted for that field.

11.3.3 See information about root cause in space assert error messages

The following error message is now more descriptive in order to provide a guide about how to solve the problem:

```
Mx0A:GeneratorContext.c:2537:MIGGeneratorContext_addData:RTI0x20a3001:!space assert
```

The new error message will depend on the root of the problem.

- If the problem is related to the message size, the following message appears:

```
MIGGeneratorContext_addData:!space assert.
         New message size (131096), current message size (36), maximum message size
(131072). Consider increment 'message_size_max'
```

- If the problem is related to the gather send buffer, the following message appears:

```
MIGGeneratorContext_addData:!space assert.
         New buffer size (24), current scratch buffer size (16), maximum scratch buffer
size (8192).
         Extra gather buffer count (1), current gather buffer index (1), maximum gather
buffer count (2).
         Consider increment 'gather_send_buffer_count_max'
```

11.3.4 See the property name in error messages related to issues in adding a property twice with the same name

Before when adding a property twice with the same name, the following error message was logged:

```
DDS_PropertySeq_add_element:!new element. ELEMENT ALREADY EXISTS. EITHER REMOVE THIS CALL OR
CALL assert_element INSTEAD. DDS_PropertyQosPolicyHelper_add_property:!add element
```

This message has been improved; now it includes the property name:

```
DDS_PropertySeq_add_element:!new element. ELEMENT 'dds.transport.UDPv4.builtin.parent.message_
size_max' ALREADY EXISTS. EITHER REMOVE THIS CALL OR CALL assert_element INSTEAD.
DDS_PropertyQosPolicyHelper_add_property:!Add property:
dds.transport.UDPv4.builtin.parent.message_size_max
```

11.3.5 Trace root cause of failures better using new function history logging feature for all supported platforms, as alternative to backtrace functionality

Backtrace support, which was added in release 6.0.1 (see "Logging a backtrace for failures" in What's New in 6.0.1) is available only on Linux, macOS, and Windows®.

For the rest of the platforms, *Connext DDS* now offers the function history. Function history is a soft version of the backtrace feature for the last function called.

Due to the performance impact, function history will only be available in debug mode and only for the platforms that do not support backtrace. Function history will be disabled by default.

You can enable function history by calling the following API method before the creation of the logger:

```
NDDS_Utility_enable_function_history()
```

11.3.6 Configure validation of property names at plugin level

Previously, the validation of property names could only be configured at the entity level using the property **dds.participant.property_validation_action**. However, it was not possible to configure validation at the plugin level. If you used an unknown or incorrect plugin property name, the creation of the plugin failed with the following error message:

```
DDS_PropertyQosPolicy_validate_plugin_property_suffixes:Unexpected property:
dds.transport.TCPv4.tcp1.invalidPropertyTest. Closest valid property:
dds.transport.TCPv4.tcp1.aliases

NDDS_Transport_TCPv4_Property_parseDDSProperties:Inconsistent QoS property:
dds.transport.TCPv4.

NDDS_Transport_TCPv4_create:!get transport TCPv4 plugin property from DDS Property
```

This lack of configuration has been resolved. Now you can decide the plugin property name validation behavior using a new property, **<plugin_name>.property_validation_action**:

- VALIDATION_ACTION_EXCEPTION: validate properties. Upon failure, log errors and fail.
- VALIDATION_ACTION_SKIP: skip validation.
- VALIDATION_ACTION_WARNING: validate properties. Upon failure, log warnings and do not fail.

If the property is not set, the plugin property validation behavior will be the same as the participant's plugin property validation behavior, which by default is VALIDATION_ACTION_EXCEPTION.

Here is an example of setting a plugin property's name validation:

```
<domain_participant_qos>
    <property>
        <value>
            <element>
                <name>dds.transport.load_plugins</name>
                <value>dds.transport.TCPv4.tcp1</value>
            </element>
            <element>
                <name>dds.transport.TCPv4.tcp1.property_validation_action</name>
                <value>VALIDATION_ACTION_WARNING</value>
            </element>
        </value>
    </property>
</domain_participant_qos>
```

11.4 Application State

11.4.1 Identify Connext DDS threads more easily using updated and consistent names

In previous releases, thread names were inconsistent or not set. Now, thread names have been updated with the goal of identifying each thread easily.

The general rules for thread names are as follows:

- The maximum length for a thread name is 16, including the '\0'.
- The first character '**r**' means that the thread has been created by *RTI Connext DDS*.
- The second and third characters identify the **module**: for example, **Co** for 'Core' or **Tr** for 'Transport.'
- The **task type** is represented with three characters: for example **Evt** for 'Event' or **Rcv** for 'Receive.'

Fields are named as follows:

- **Participant identifier** is five characters, as follows:
  - The first 3 characters and last 2 characters of the **participant_name**, if set.
  - The **DomainId** (3 characters) plus **participant_id** (2 characters), if **participant_name** is not set.
  - The last five digits of the **rtps_instance_id** in the participant GUID if **participant_name** is not set and **participant_id** is set to -1 (default value).
- **Transport name** is four characters: for example, **TCP4** for 'Transmission Control Protocol version 4 (TCPv4)' or **DTLS** for 'Datagram Transport Layer Security (DTLS).'

Table 1.1 Example Thread Names

Thread Information	Name	Fields	Example: Domain: 111 Participant Id : 22 ThreadIndex: 33 Topic: HelloWorld DataBase: Test Application Name: TestPersistence
Receive thread	rCo%5s##%02dRcv	Participant identifier, thread index	rCo11122##33Rcv
Asynchronous waitset thread	rCo%5s##%02dAWs	Participant identifier, thread index	rCo11122##33AWs

Thread Information	Name	Fields	Example: Domain: 111 Participant Id : 22 ThreadIndex: 33 Topic: HelloWorld DataBase: Test Application Name: TestPersistence
Database thread	rCo%5s####Dtb	Participant identifier	rCo11122####Dtb
Event thread	rCo%5s####Evt	Participant identifier	rCo11122####Evt
TCP event thread	rTr%5s%04sEvt	Participant identification, transportName (TCP4)	rTr11122TCP4Evt
DTLS event thread	rTr%5s%04sEvt	Participant identification, transportName (DTLS)	rTr11122DTLSEvt
WAN server thread	rTr%5s%04sSvr	Participant identification, transportName (WAN)	rTr11122#WANCtr
Interface tracking thread	rTr%5s%04sITr	Participant identification, transportName (UDP4, UDP6, TCP4)	rTr11122UDP4ITr
Persistence Service publication thread	rPs%07s%02dPub	topic name, thread index	rPsHello##33Pub
Recording Service timer thread	rRe#########Tim		rRe#########Tim
Monitor event thread	rMo%5s####Evt	Participant identifier	rREHelloWorlPub
Routing Service filter tracker event thread	rRsFilterTr#Evt		rRsFilterTr#Evt
Database Integrated Service connection thread	rDs%.9sCon	Database name	rDsTestsCon

For complete information, see the section "Identifying Threads Used by Connext DDS" in the *RTI Connext DDS Core Libraries User's Manual*.

11.4.2 See updated name of interface tracker thread of the IP Mobility feature

The name of the interface tracker thread of the IP Mobility feature has been updated to **rTr<Participant identifier><Transport name>ITr**:

- **r**: Specify that the thread has be created by *RTI Connext DDS*.
- **Tr**: Identify the transport module.

- **Participant identifier:** five characters to identify the participant. It is described in 11.4.1 Identify Connext DDS threads more easily using updated and consistent names on page 18 with more details.
- **Transport name**: four characters to identify the transport. It can be UDP4 or UDP6 or TCP4.
- **ITr**: taskType of the thread, in this case, "Interface tracker".

11.4.3 Receive discovery information implicitly from RTPS header

If the following fields are not sent as part of the BuiltinTopicData in the discovery process, *Connext DDS* now derives them from the RTPS header and from other fields:

- ParticipantBuiltinTopicData: VendorId, Protocol Version, Participant Guid.
- PublicationBuiltinTopicData: VendorId, Protocol Version, Virtual Guid (derived from endpoint Guid).
- SubscriptionBuiltinTopicData: VendorId, Protocol Version, Virtual Guid (derived from endpoint Guid).

Note that *Connext DDS* always propagates these fields; this enhancement has no effect when discovering *Connext DDS* entities. This enhancement is useful when discovering entities from remote vendors, which might not always send these fields.

11.4.4 View product version and type name used in pool allocation for heap monitoring snapshots

Heap monitoring now prints the correct type name used in the pool allocation for the heap monitoring snapshot. (Previously, the name of the type allocated by the pool was not accurate, and it was redundant when printing the snapshot of heap monitoring.) The product version is now also in the header of the heap monitoring snapshot file.

For example:

```
Product Version: NDDSCORE_BUILD_6.0.0.0_20191211T113449Z_RTI_ENG
Process virtual memory: 1518837760
Process physical memory: 509612032
Current heap usage: 887096247
High watermark: 888834186
Alloc count: 10791197
Free count: 9485215
block_id, timestamp, block_size, pool_alloc, pool_buffer_size, pool_buffer_count, topic_name,
activity, alloc_method_name, type_name
.
.
.
1576509471, 64, POOL, 64, 1, PRESServiceRequest, PRESCstReaderCollator_new, RTIOsapiHeap_
allocateBufferAligned, struct REDASkiplistNode
```

```
18817, 1576509471, 2000, POOL, 1000, 2, PRESServiceRequest, PRESCstReaderCollator_new,
RTIOsapiHeap_allocateBufferAligned, struct PRESCstReaderCollatorRemoteWriterQueue

18821, 1576509471, 9984, POOL, 312, 32, PRESServiceRequest, PRESCstReaderCollator_new,
RTIOsapiHeap_allocateBufferAligned, struct PRESCstReaderCollatorRegisteredKeyedEntry

18831, 1576509471, 2560, POOL, 80, 32, PRESServiceRequest, PRESCstReaderCollator_new,
RTIOsapiHeap_allocateBufferAligned, struct PRESCstReaderCollatorInstanceFilterMemberNode

18839, 1576509471, 5376, POOL, 168, 32, PRESServiceRequest, PRESCstReaderCollator_new,
RTIOsapiHeap_allocateBufferAligned, struct PRESCstReaderCollatorKeyedEntry
```

11.4.5 See messages from Security Plugins marked as related to security

Messages from *RTI Security Plugins* now have the **is_security_message** flag in NDDS_Config_LogMessage set to TRUE.

11.5 Backtrace for Fatal Error Debugging

Backtrace support was added in release 6.0.1 (see "Logging a backtrace for failures" in the 6.0.1 *Core Libraries Release Notes*). In this release, the following enhancements have been made to the backtrace functionality.

11.5.1 View logs related to crashes and fatal errors using FATAL log level, which is printed by default in DEBUG format

A new **NDDS_Config_LogLevel** has been added: NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR. This log level indicates an unrecoverable situation in the functioning of *Connext DDS*. Error messages with this log level usually indicate a violation of an internal invariant or a segmentation fault.

Now by default, the **print_format** NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG is set for the log level NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR. (For the rest of the log levels, NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT is used.)

This means that by default the backtrace is logged in precondition and segmentation faults; however, you can disable the backtrace for NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR. In the following code, the log level NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR uses the **print_format** NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT, which does not contain the backtrace information:

```
NDDS_Config_Logger *logger = NDDS_Config_Logger_get_instance();
NDDS_Config_Logger_set_print_format_by_log_level(
           logger,
           NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT,
           NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR));
```

See the "Logging a Backtrace for Failures" section in the *RTI Connext DDS Core Libraries User's Manual*.

11.5.2 Redundant backtrace information no longer logged for same error message

Previously, the backtrace was logged for each error message, but this was redundant for an error that had several related errors following it.

This issue has been improved. The backtrace feature is now smart enough to log the backtrace only once for a given error and not for the following errors in the same code path of the caller's functions.

For example, in the failure of the creation of the DDSDomainParticipant, *Connext DDS* logs the backtrace for just one error instead of logging it for all of the error messages in the same code path:

```
U00007f86a87df700 Mx08:Udpv4SocketFactory.c:685:RTI0x2080010:invalid port 5562900
Backtrace:
#3 NDDS_Transport_UDPv4_Socket_bind_with_ip ??:? [0xCB235C]
#4 NDDS_Transport_UDPv4_SocketFactory_create_receive_socket ??:? [0xCB2619]
#5 NDDS_Transport_UDP_create_recvresource_rrEA Udp.c:? [0xCAB170]
#6 RTINetioReceiver_addEntryport ??:? [0xCA33F3]
#7 COMMENDActiveFacade_addEntryport ActiveFacade.c:? [0xC12B56]
#8 DDS_DomainParticipantPresentation_reserve_entryportI DomainParticipantPresentation.c:?
[0x7E4F11]
#9 DDS_DomainParticipantPresentation_reserve_participant_index_entryports ??:? [0x7E8015]
#10 DDS_DomainParticipant_reserve_participant_index_entryports DomainParticipant.c:? [0x7B0B7E]
#11 DDS_DomainParticipant_enableI DomainParticipant.c:? [0x7CC15E]
#12 DDS_Entity_enable ??:? [0x72EC92]
#13 DDS_DomainParticipantFactory_create_participant ??:? [0x7DACF1]
#14 main ??:? [0x40675F]
#15 ?? ??:0 [0xA76F4830]
#16 _start ??:? [0x405EC9]
U00007f86a87df700 Mx0F:DomainParticipant.c:13313:RTI0x20f0c02:Automatic participant index
failed to initialize. PLEASE VERIFY CONSISTENT TRANSPORT / DISCOVERY CONFIGURATION.
U00007f86a87df700 Mx0F:DomainParticipantFactory.c:1314:RTI0x20f000e:ERROR: Failed to auto-
enable entity
U00007f86a87df700 Mx01:DomainParticipantTester.c:9325:RTI0x2000007:!
[DomainParticipantTester.c:9325] pointer is null: participant
```

See the "Logging a Backtrace for Failures" section in the *RTI Connext DDS Core Libraries User's Manual*.

11.5.3 Enable backtrace information for log levels using print formats

Backtrace information is now part of the format used to output *Connext DDS* logging. The backtrace will be logged in the following print formats:

- NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG
- NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL

See the "Format of Logged Messages" section in the *RTI Connext DDS Core Libraries User's Manual*.

11.5.4 Backtrace functionality disabled by default in ppc32e6500Linuxgcc4.9.1 platforms

In general, for all architectures, the Backtrace functionality is enabled by default.

Some platforms, however, have not had their C standard library (libc) built with “libc-backtrace". So if the backtrace is logged on these platforms, your application fails, printing the following error at runtime:

```
./app: relocation error:
./app: symbol backtrace, version GLIBC_2.1 not defined in file libc.so.6 with link time
reference
```

Therefore, for ppc32e6500Linuxgcc4.9.1, RTI has disabled the backtrace functionality by default. For this platform, if you set an **NDDS_Config_LogPrintFormat** that contains the backtrace, the bit for the backtrace will be ignored. However, if you know that the C standard library (libc) has been built with "libc-backtrace" and you would like to force the use of the backtrace functionality, you can do so using a special **NDDS_Config_LogPrintFormat**.

In order to force the backtrace for this platform, use the bit “0x80”. For example:

```
NDDS_Config_Logger_set_print_format_by_log_level(
            NDDS_Config_Logger_get_instance(),
            NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG | 0x80,
            NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR);
```

See the *RTI Connext DDS Core Libraries Platform Notes* and "Logging a Backtrace for Failures" in the *RTI Connext DDS Core Libraries User's Manual* for more information.

11.6 Improvements in Log Messages

11.6.1 View Exclusive Area (EA) names and stacktrace in logs related to deadlock risk errors

Previously, when there was a deadlock using Exclusive Areas (EA), an error message was logged, similar to the following:

```
REDAWorker_enterExclusiveArea:worker rCoRTImo####Evt deadlock risk: cannot enter 0x2811300 of
level 35 from level 40
```

In this release, *Connext DDS* adds the EA names and the stacktrace where the issue is happening to the message. For example:

```
[0x0101A11B,0xA02FC811,0x290A76BA:0x80000003{K=DW,T=testEventsTopic,Y=DDS::String,D=10}|LINK
0x0101A11B,0xA02FC811,0x290A76BA:0x80000004{Y=DDS::String}|:0x80018C42
{K=DW,T=rti/dds/monitoring/dataWriterEntityMatchedSubscriptionWithLocatorStatistics,Y=rti::dds
::monitoring::DataWriterEntityMatchedSubscriptionWithLocatorStatistics,D=10}|WRITE] REDAWorker_
enterExclusiveArea:worker rCo51610####Evt deadlock risk: cannot enter 'PUBLISHER_EA' of level
35 from 'DP_REMOTE_EA' of level 40.
Backtrace:
#1      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(REDAWorker_enterExclusiveArea+0x100)
[0x10e6a66]
#2      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(REDACursor_modifyReadWriteArea+0x31)
[0x10e9285]
```

```
#3      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(PRESPsWriter_writeInternal+0x8ff)
[0xec5083]
#4      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(DDS_DataWriter_write_untyped_
generalI+0x43c) [0xbe1dff]
#5      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(rti_dds_monitoring_
DataWriterEntityMatchedSubscriptionWithLocatorStatisticsDataWriter_write+0x19) [0xabbbf9]
#6      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(RTIDefaultMonitorParticipantObject_
publishWriterMatchedWithLocatorStatsI+0x122) [0xad17e1]
#7      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(RTIDefaultMonitorParticipantObject_
sampleAndPublishWriterMatchedWithLocatorStats+0xce) [0xad1a21]
#8      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(RTIDefaultMonitorPublisher_
onEventNotify+0x20eb) [0xac6fa3]
#9      ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(DDS_DomainParticipantMonitoringListener_
notify_library+0x1a9) [0xb5c9c5]
#10     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(DDS_DomainParticipantMonitoring_
onNewWriterLocatorPair+0xd0) [0xb5e754]
#11     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(PRESPsService_
writerStatusListenerOnNewWriterLocatorPair+0x2ec) [0xf2faca]
#12     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(COMMENDBeWriterService_
assertRemoteReader+0x34ed) [0xffd542]
#13     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0xf57002]
#14     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0xf58da1]
#15     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(PRESPsService_
linkLocalWriterToRemoteReaders+0x7d7) [0xf5bae8]
#16     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester(PRESPsService_
onLinkToRemoteEndpointEvent+0x132) [0xf17838]
#17     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0x1088ebf]
#18     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0x110d18e]
#19     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0x110d18e]
#20     ./monitor.1.0/lib/x64Linux3gcc5.4.0/monitorTester() [0x1106681]
#21     /lib/x86_64-linux-gnu/libpthread.so.0(+0x76ba) [0x7f89c90b66ba]
#22     /lib/x86_64-linux-gnu/libc.so.6(clone+0x6d) [0x7f89c8be44dd]
```

11.6.2 See warning message in logs when receive socket buffer size is larger than the maximum

The following message was logged with STATUS_LOCAL verbosity:

```
NDDS_Transport_UDPv4_SocketFactory_create_receive_socket:The specified recv_socket_buffer_size,
67108864, was not set. The actual receive socket buffer size is 425984
```

It has been changed to be logged with WARNING verbosity.

11.6.3 Ability to see new activity context information available as part of NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG print format

Connext DDS now includes the activity context as part of the **NDDS_Config_LogPrintFormat** NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG.

The activity context provides extra contextual information to the log message when using the DEBUG **NDDS_Config_LogPrintFormat**. It describes the activity (such as, “Get Qos” or “Sending participant discovery announcements”) that a resource was doing when the logging occurred. For example, in the

creation of a *DataWriter*, the activity context will provide information about the resource—the *Publisher* creating the entity. The activity will be "entity creation."

For example, a message using NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG will now look like this (the **bolded** information is new):

```
U00007f71fe36d700_dds_c3Tester
[0X1019D1D,0XBD6B47B0,0XA6C11F6B:0X80004202{E=Writer,T=Example Stock,C=Stock,D=1}|WRITE]
Mx16:Memory.c:7311:RTI0x2161000:instance not found
Backtrace:
#3      WriterHistoryMemoryPlugin_addSample ??:? [0xC236CE]
#4      PRESWriterHistoryDriver_addWrite ??:? [0xE11B7F]
#5      PRESPsWriter_writeInternal ??:? [0xB5EBEA]
#6      DDS_DataWriter_write_untyped_generalI ??:? [0x8AED4D]
#7      StockDataWriter_write ??:? [0x71B1D1]
#8      DDSCDataWriterTester_testAutoRegisterInstance ??:? [0x41FC10]
#9      RTITestSetting_runTestsExt ??:? [0xABA2C3]
#10     DDSCDataWriterTester_run ??:? [0x436EA0]
#11     RTITestSetting_runTestsExt ??:? [0xABA2C3]
#12     RTITestSetting_runTests ??:? [0xABB05B]
#13     main ??:? [0x40687F]
#14     ?? ??:0 [0xFD49C830]
#15     _start ??:? [0x405FE9]
```

11.6.4 Easier to identify Connext DDS threads with improved consistency in thread names in logs

The thread identifier information is part of the **NDDS_Config_LogPrintFormat** logging infrastructure formats. Sometimes, this field was inconsistently populated.

This problem has been resolved. The thread identifier is:

- The thread name for the Connext DDS threads.
- "U" + Thread Id + Thread Name (only on Posix platforms), for the User threads.

11.6.5 Improved messages on fixing issues with reserving memory for writer/reader pools

Previously, an error during *DataWriter/DataReader* creation looked similar to the following ones:

- For the *DataWriter*:

```
[D001|Pub(80000008)|T=ExampleTest|CREATE Writer] PRESTypePluginDefaultEndpointData_
createWriterPool:!create writer buffer pool
```

- For the *DataReader*:

```
[D001|Sub(80000009)|T=ExampleTest|CREATE Reader] PRESCstReaderCollator_new:!create
serializedKeyPool
```

Now the messages include information about how to fix this issue:

- For the *DataWriter*:

```
[0X101301B,0X2BD1038F,0X503C2B68:0X80000008\{E=Pu,D=1}|CREATE Writer WITH TOPIC
ExampleTest] PRESTypePluginDefaultEndpointData_createWriterPool:Failed to create writer
buffer pool, cannot allocate 32 initial samples with size 520000. Consider setting
dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size if your type has a
large or unbounded max serialized size or reduce initial_samples.
```

- For the *DataReader*:

```
[0X1013122,0XDDCD138E,0X7FF55651:0X80000009\{E=Su,D=1}|CREATE Reader WITH TOPIC
ExampleTest] PRESCstReaderCollator_new:!create serializedKeyPool. Consider setting the
property 'dds.data_reader.history.memory_manager.fast_pool.pool_buffer_max_size'
```

11.6.6 See improved logging messages for issues in destroying all participants upon deletion of a DomainParticipantFactory instance

When you deleted the *DomainParticipantFactory* instance and not all the participants were destroyed, the following error message was logged.

```
DDS_DomainParticipantFactory_deleteI:!delete factory instance: outstanding participant(s)
```

This message now has more debugging information:

```
DDS_DomainParticipantFactory_deleteI:ERROR: Failed to delete the DomainParticipantFactory
instance. Not all the participants created were destroyed (2 left).
DDS_DomainParticipantFactory_deleteI:   0) DomainParticipant  with GUID
(0x01012A3D,0x934AAC0F,0xF52D39B2:0x000001C1) was not destroyed.
DDS_DomainParticipantFactory_deleteI:   1) DomainParticipant  with GUID
(0x01014185,0x60E370D7,0xB83C5848:0x000001C1) was not destroyed.
```

11.6.7 View suggestions in logs for how to proceed when Connext DDS detects an unexpected property

The error message logged during the validation of properties in the PROPERTY QoS Policy has been improved.

Before, *Connext DDS* logged the following message when a property was not expected:

For an entity:

```
DDS_PropertyQosPolicy_validatePropertyNames:Unexpected property: dds.type_consistnecy.ignore_
sequence_bounds. Closest valid property: dds.type_consistency.ignore_sequence_bounds
DDS_DataReaderQos_is_consistentI:inconsistent QoS property
DDS_Subscriber_create_datareader_disabledI:ERROR: Inconsistent QoS
```

For a plugin, such as TCPv4:

```
DDS_PropertyQosPolicy_validate_plugin_property_suffixes:Unexpected property:
dds.transport.TCPv4.tcp1.invalidPropertyTest. Closest valid property:
dds.transport.TCPv4.tcp1.aliases
NDDS_Transport_TCPv4_Property_parseDDSProperties:Inconsistent QoS property:
dds.transport.TCPv4.
NDDS_Transport_TCPv4_create:!get transport TCPv4 plugin property from DDS Property
```

Now there is extra information in case you wish to proceed with that property:

Entity:

```
DDS_PropertyQosPolicy_validateEntityPropertyNames:Unexpected property: dds.type_
consistnecy.ignore_sequence_bounds. Closest valid property: dds.type_consistency.ignore_
sequence_bounds. If you wish to proceed with this property name anyway, change
'dds.participant.property_validation_action' to 'VALIDATION_ACTION_SKIP' or 'VALIDATION_ACTION_
WARNING'.
DDS_DataReaderQos_is_consistentI:inconsistent QoS property
DDS_Subscriber_create_datareader_disabledI:ERROR: Inconsistent QoS
```

Plugin, such as TCPv4:

```
DDS_PropertyQosPolicy_validate_plugin_property_suffixes:Unexpected property:
dds.transport.TCPv4.tcp1.invalidPropertyTest. Closest valid property:
dds.transport.TCPv4.tcp1.aliases. If you wish to proceed with this property name anyway, change
'dds.transport.TCPv4.tcp1.property_validation_action' to 'VALIDATION_ACTION_SKIP' or
'VALIDATION_ACTION_WARNING'.
NDDS_Transport_TCPv4_Property_parseDDSProperties:Inconsistent QoS property:
dds.transport.TCPv4.
NDDS_Transport_TCPv4_create:!get transport TCPv4 plugin property from DDS Property
```

12 Improved Control over Application Behavior

12.1 Instance Lifecycle Management

12.1.1 Configure how existing instances will be replaced to make space for new instances when max_instances is reached, using a new DataReader-side instance replacement policy

A *DataWriter* has long used the **instance_replacement** field in the DATA_WRITER_RESOURCE_LIMITS QoS Policy whenever the **max_instances** limit in the RESOURCE_LIMITS QoS Policy is reached. Now the *DataReader* also has an **instance_replacement** field, set in the DATA_READER_RESOURCE_LIMITS QoS Policy, that is used to determine the behavior whenever **max_instances** in the RESOURCE_LIMITS QosPolicy is reached.

Now when the **max_instances** limit in the RESOURCE_LIMITS QosPolicy is reached, a *DataReader* will try to make space for a new instance by replacing an existing instance according to the instance replacement kind set in the **instance_replacement** field.

The **instance_replacement** field is useful for managing potentially unbounded sets of instances that come and go. It is important to be able to set an upper limit on the resources that will be used by an application to avoid running into decreased performance and potentially running out of system resources. This new QoS on the *DataReader* side allows you to set an upper bound on the resources that will be used for instances.

Before this QoS was in place, when the **max_instances** resource limit was reached, no more instances could be accepted by the DataReader before others were unregistered. This put an unnecessary burden on the applications to unregister instances and manage the instance lifecycle in the application. Now, you can set this QoS, allowing *DataReaders* to make room for new instances by replacing older ones.

For example, a hospital may have 100 beds. Many patients (instances) come and go, so at any given time you only need resources for 100 instances, but over time you will see an unbounded number of instances. An instance replacement policy can help manage this flow.

For each instance state (ALIVE, NOT_ALIVE_DISPOSED, and NOT_ALIVE_NO_WRITERS), you can set the following removal kinds for the DataReader:

- The **alive_instance_removal** kind sets a removal policy for ALIVE instances (default: DDS_NO_INSTANCE_REMOVAL).
- The **disposed_instance_removal** kind sets a removal policy for NOT_ALIVE_DISPOSED instances (default: DDS_EMPTY_INSTANCE_REMOVAL).
- The **no_writers_instance_removal** kind sets a removal policy for NOT_ALIVE_NO_WRITERS instances (default: DDS_EMPTY_INSTANCE_REMOVAL).

For each of the above removal kinds, you can choose among the following replacement criteria:

- DDS_NO_INSTANCE_REMOVAL: Instances in the associated state cannot be replaced. This means that samples for new instances that exceed the max_instances resource limit will be lost with the reason LOST_BY_INSTANCES_LIMIT (see SAMPLE_LOST Status).

- DDS_EMPTY_INSTANCE_REMOVAL: Instances in the associated state can be replaced only if they are empty (all samples have been taken or removed from the DataReader queue due to the LIFESPAN QoS Policy or sample purging due to the READER_DATA_LIFECYCLE QoS Policy, and there are no outstanding loans on any of the instance's samples).

- DDS_FULLY_PROCESSED_INSTANCE_REMOVAL: Instances in the associated state can be replaced only if every sample has been processed by the application.

- DDS_ANY_INSTANCE_REMOVAL: Instances in the associated state can be replaced regardless of whether the subscribing application has processed all of the samples. Samples that have not been processed will be dropped and accounted for by the DataReaderCacheStatus total_samples_dropped_by_instance_replacement statistic.

See the "DATA_READER_RESOURCE_LIMITS QosPolicy" section of the *RTI Connext DDS Core Libraries User's Manual* for more information.

As part of this feature, a new **total_samples_dropped_by_instance_replacement** field has been added to in the DDS_DataReaderCacheStatus to count the number of NOT_READ samples replaced as a result of *DataReader*-side instance replacement.

12.1.2 Define minimum duration for which Data Reader will maintain information about NOT_ALIVE_NO_WRITERS instance with no samples, using a new QoS setting

The **autopurge_nowriter_instances_delay** defines the minimum duration for which the *DataReader* will maintain information about a NOT_ALIVE_NO_WRITERS instance with no samples in the *DataReader* queue. With the addition of this field, the behavior of the *DataReader* with regards to the lifecycle of the instances it manages offers the same configuration options for both NOT_ALIVE_DISPOSED and NOT_ALIVE_NO_WRITERS instances. Currently the only supported values are 0 or INFINITE; the default value is 0.

12.1.3 Query liveliness of matched remote entities using new APIs

Two new APIs have been added, **DDS_DataWriter_is_matched_subscription_active** and **DDS_DataReader_is_matched_publication_alive**. These can be used to query the liveliness of matched remote entities.

See also issue CORE-9366 in "Fixes Related to OMG Specification Compliance" in the RTI Connext DDS Core Libraries Release Notes. As part of CORE-9366, the **DDS_DataWriter_get_matched_subscriptions** and **DDS_DataReader_get_matched_publications** APIs now return the instance handles for any matching remote entities, including those that are not alive.

12.1.4 Filter out NOT-ALIVE instances from historical part of responses to a CONTINUOUS TopicQuery

Connext DDS 6.0.0 added a new feature to TopicQuery that allows selecting only ALIVE instances for HISTORY_SNAPSHOT TopicQueries (see "Ability to select only alive instances with TopicQuery" in the 6.0.0 *Core Libraries Release Notes*).

This release extends this functionality to filter out NOT-ALIVE instances from the historical portion of the responses to a CONTINUOUS TopicQuery, when the string @instance_state = ALIVE is prepended to the TopicQuery filter expression. That is, when the continuous TopicQuery is first dispatched by the *DataWriter*, no previously-written samples or meta-samples (disposed or unregistered samples) are delivered in response to the query for instances in a NOT-ALIVE state; however, all subsequently written disposed or unregistered meta-samples for any instance will be delivered—as well as samples matching the rest of the filter expression—as long as the continuous TopicQuery remains active (i.e., not deleted).

12.1.5 Obtain statistics for currently maintained instances in DataReaderCacheStatus and DataWriterCacheStatus

The DataReaderCacheStatus and DataWriterCacheStatus structures have been extended to provide information about the instances that are currently being maintained by that *DataReader* or *DataWriter*.

The following fields have been added to DataReaderCacheStatus:

- alive_instance_count
- alive_instance_count_peak
- no_writers_instance_count
- no_writers_instance_count_peak
- disposed_instance_count
- disposed_instance_count_peak
- detached_instance_count
- detached_instance_count_peak

The following fields have been added to DataWriterCacheStatus:

- alive_instance_count
- alive_instance_count_peak
- disposed_instance_count
- disposed_instance_count_peak
- unregistered_instance_count
- unregistered_instance_count_peak

The monitoring topics have also been updated to publish this information.

12.2 Logging Usability

12.2.1 Enable log warnings to indicate when certain operations are taking longer than expected

You will now be able to configure logging a warning when a specific operation takes more time than expected. The different operations are:

- Send operation: Print warning message when the send operation time exceeds the time threshold configured by the property **dds.participant.logging.time_based_logging.send.timeout**.
- Event operations: Print warning message when the event start/execution time exceeds the time threshold configured by the property **dds.participant.logging.time_based_logging.event.timeout**.
- Process received data operation: Print warning message when the processing of a received message on a specific port exceeds a time threshold set in **dds.participant.logging.time_based_logging.process_received_message.timeout**.

- Authentication process: Print warning message when the authentication operation time exceeds the time threshold configured by the property **dds.participant.logging.time_based_logging.authentication.timeout**.

See "Setting Warnings for Operation Delays" in the *RTI Connext DDS Core Libraries User's Manual* for more information.

12.2.2 Verbosity level for log messages printed for samples written with an out of order sequence number now set to NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL

Setting the property **dds.data_writer.history.allow_out_of_order_write** to TRUE allows writing a sample using **DataWriter::write_w_params**, where **identity.sequence_number** is smaller than the sequence number of the last sample written by the *DataWriter*.

However, when the sequence number ordering was violated, *Connext DDS* printed the following warning:

```
PRESWriterHistoryDriver_resolveAndCheckOriginalWriterInfo:sequence number out of order.
Expected greater or equal to (x,y)
```

This release changes the verbosity of the message to be NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL. Warning was not the right level because, by setting **dds.data_writer.history.allow_out_of_order_write** to TRUE, the user accepted out-of-order writing as valid.

12.2.3 Control level of verbosity for every log level by specifying the print format

Now, you will be able to set a different print format, which controls the level of verbosity for every log level. To configure this, there are two new APIs:

- Set the **print_format** at which *Connext DDS* will log diagnostic information in the given logLevel.

```
DDS_Boolean NDDS_Config_Logger_set_print_format_by_log_level(
 NDDS_Config_Logger *self,
 NDDS_Config_LogPrintFormat print_format,
 NDDS_Config_LogLevel log_level);
```

- Get the **print_format** at which *Connext DDS* will log diagnostic information in the given logLevel.

```
NDDS_Config_LogPrintFormat
NDDS_Config_Logger_get_print_format_by_log_level(
 const NDDS_Config_Logger *self,
 NDDS_Config_LogLevel log_level);
```

You could use a less verbose **print_format**, such as NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL, for warnings, as follows:

```
NDDS_Config_Logger *logger = NDDS_Config_Logger_get_instance();
NDDS_Config_Logger_set_print_format_by_log_level(
```

```
    logger,
    NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL,
    NDDS_CONFIG_LOG_LEVEL_WARNING));
```

You could use a more verbose **print_format**, such as NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG (which contains the backtrace) when you are troubleshooting errors, as follows:

```
NDDS_Config_Logger *logger = NDDS_Config_Logger_get_instance();
NDDS_Config_Logger_set_print_format_by_log_level(
    logger,
    NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG,
    NDDS_CONFIG_LOG_LEVEL_ERROR));
```

This way, you will reduce the amount of logging on warnings, and errors will contain more information. This configuration is key to understanding and solving issues when needed.

By default, NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT is assigned to all log levels except FATAL_ERROR. By default, FATAL_ERROR is assigned to NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG, which prints the backtrace information.

See the "NDDSConfigLogger Operations" table in the *RTI Connext DDS Core Libraries User's Manual*.

12.2.4 Easily access distributed log levels in C and C++ using new APIs

Previously, to access distributed log levels, you needed to use a .idl file, generate code, and include the header in your project.

Now, you can access the distributed log levels, in C and C++, using new APIs:

```
DDS_Long RTI_DL_DistLogger_get_fatal_log_level();
DDS_Long RTI_DL_DistLogger_get_error_log_level();
DDS_Long RTI_DL_DistLogger_get_warning_log_level();
DDS_Long RTI_DL_DistLogger_get_notice_log_level();
DDS_Long RTI_DL_DistLogger_get_info_log_level();
DDS_Long RTI_DL_DistLogger_get_debug_log_level();
```

```
DDS_Long DistLogger::getFatalLogLevel();
DDS_Long DistLogger::getErrorLogLevel();
DDS_Long DistLogger::getWarningLogLevel();
DDS_Long DistLogger::getNoticeLogLevel();
DDS_Long DistLogger::getInfoLogLevel();
DDS_Long DistLogger::getDebugLogLevel();
```

12.2.5 Logging verbosity now remains unchanged if creation of Distributed Logger instance fails

If other threads are writing log messages while the main thread is trying to call **RTI_DL_DistLogger_getInstance**, it is possible for **RTI_DL_DistLogger_getInstance** to fail with these errors:

```
DL Error: RTI_DL_DistLogger_createInstance: Unable to hook up RTI Logger
DL Error: RTI_DistLogger_getInstance: Unable to create DistLogger singleton!
```

If these errors occurred, then the logging verbosity would incorrectly be set at NDDS_CONFIG_LOG_VERBOSITY_SILENT, which prevented the NDDS_Config_Logger from generating any further log messages. This problem has been fixed. The logging verbosity now remains unchanged if Distributed Logger creation fails.

12.3 Resource Usage Tuning

12.3.1 Configure how to allocate memory for serialized typeObjects using new QoS field in DDS_DomainParticipantResourceLimitsQosPolicy

A new QoS field has been added to the DDS_DomainParticipantResourceLimitsQosPolicy for configuring how to allocate the serialized typeObject.

serialized_type_object_dynamic_allocation_threshold is a threshold, in bytes, for dynamic memory allocation for the serialized typeObject. Above this threshold, the memory for a TypeObject is allocated dynamically. Below it, the memory is obtained from a pool of fixed-size buffers.

If **type_object_max_serialized_length** is not LENGTH_UNLIMITED and is smaller than **serialized_type_object_dynamic_allocation_threshold**:

- **serialized_type_object_dynamic_allocation_threshold** will be adjusted to **type_object_max_serialized_length**.
- The following warning will be logged:

```
DDS_DomainParticipantResourceLimitsQosPolicy_is_consistent:inconsistent QoS policies:
serialized_type_object_dynamic_allocation_threshold and type_object_max_serialized_
length. serialized_type_object_dynamic_allocation_threshold will be adjusted with type_
object_max_serialized_length.
```

By default, **serialized_type_object_dynamic_allocation_threshold** is the same value as **type_object_max_serialized_length**, 8192. This means that the typeObject memory is obtained from a pool of fixed-size buffers.

The new field can be configured in the DDS_DomainParticipant as follows:

```
<domain_participant_qos>
    <resource_limits>
        <rtps_reliable_writer>
<serialized_type_object_dynamic_allocation_threshold>3072</serialized_type_object_dynamic_
allocation_threshold>
        </rtps_reliable_writer>
    </resource_limits>
</domain_participant_qos>
```

12.3.2 Control throughput of certain topics by disabling repair of piggyback heartbeats using new QoS field in RtpsReliableWriterProtocol

A new QoS field has been added to the **RtpsReliableWriterProtocol** in the DATA_WRITER_PROTOCOL QoS Policy (for application *DataWriters*) and DISCOVERY_CONFIG QoS Policy (for builtin *DataWriters*) for preventing piggyback heartbeats from being sent with repair samples.

When samples are repaired, the *DataWriter* resends **RtpsReliableWriterProtocol_t_max_bytes_per_nack_response** bytes and a piggyback heartbeat with each message. You can configure the *DataWriter* to not send the piggyback heartbeat and instead rely on the **RtpsReliableWriterProtocol_t_late_joiner_heartbeat_period** to control the throughput used to repair samples.

This QoS setting is only mutable for application *DataWriters* using the DATA_WRITER_PROTOCOL QoS Policy. The QoS setting is not mutable for builtin *DataWriters*.

The default value for this field is BOOLEAN_FALSE.

The new QoS setting can be configured in DATA_WRITER_PROTOCOL and DISCOVERY_CONFIG as follows:

```
<datawriter_qos>
    <protocol>
        <rtps_reliable_writer>
            <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
        </rtps_reliable_writer>
    </protocol>
    </datawriter_qos>
    <domain_participant_qos>
        <discovery_config>
            <secure_volatile_writer>
                <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
            </secure_volatile_writer>
            <publication_writer>
                <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
            </publication_writer>
            <subscription_writer>
                <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
            </subscription_writer>
            <participant_message_writer>
                <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
            </participant_message_writer>
            <service_request_writer>
                <disable_repair_piggyback_heartbeat>false</disable_repair_piggyback_heartbeat>
            </service_request_writer>
        </discovery_config>
    </domain_participant_qos>
```

12.4 Improvements in Transport Functionality

12.4.1 Detect changes in the IP address for a name resolved by the DNS Service

Connext DDS allows adding peers based on a hostname instead of an IP address. Those hostnames were resolved into an IP address only when they were added. Therefore, changes in the IP address that the hostname was resolved to were not noticed by *Connext DDS*.

This release introduces a way to detect these changes in the IP address that a hostname is resolved to and update the related peers accordingly. This mechanism creates a new thread that regularly polls the DNS service and uses a callback to notify *Connext DDS* of the changes in the resolved IP address of a tracked hostname.

To enable, disable, and configure this feature, set the appropriate value in the **dns_tracker_polling_period** field in the DISCOVERY_CONFIG QoS Policy.

See the section "Using DNS Tracker to Keep the Peer List Updated," in the *RTI Connext DDS Core Libraries User's Manual* for more information.

12.4.2 IP mobility change events will now cause reduced network traffic in certain scenarios

In previous releases, IP mobility events that qualified as a change (for example, the change of the IP address of one of the local interfaces) always triggered the sending of both Participant and Endpoint discovery updates to all the matched remote entities.

This was true even in scenarios where, due to the nature of the change, it was enough just to send Participant Discovery updates—for example, when *DataWriters* and *DataReaders* were using the default Participant locators (as opposed to setting specific locators through TransportSelection or TransportUnicast QoS policies).

This release changes the behavior of *Connext DDS* in this scenario: a Participant will now only send Participant Discovery updates, saving the network bandwidth associated with sending Endpoint Discovery updates. *Connext DDS* will still propagate Endpoint Discovery updates in scenarios where *DataWriters* and *DataReaders* are using specific locators instead of the ones inherited from the Participant.

> IMPORTANT: This change breaks backwards compatibility with previous versions of *Connext DDS* once an IP mobility change occurs. Previous versions of *Connext DDS* still need to receive the redundant Endpoint Discovery traffic to process the change. Or you can set the property **dds.participant.discovery_config.force_endpoint_announcement_on_ip_mobility_event** to true.

12.4.3 TCP transport in TLS mode now logs error message if CA certificate file is missing

When using the TCP transport in TLS mode and the CA Certificate file was missing, there was no error message specifying the problem.

Now the following error is logged:

```
RTITLS_configuration_verify:Identifying certificate not specified
NDDS_Transport_TCPv4_new:!create connection endpoint factory
DDS_DomainParticipantConfigurator_setup_custom_transports:!create custom transport plugin
DDS_DomainParticipantConfigurator_enable:!install transport plugin aliases = custom transports
DDS_DomainParticipant_enableI:!enable transport configurator
DDS_DomainParticipantFactory_create_participant:ERROR: Failed to auto-enable entity
```

To specify the CA certificate file, add it to the property **dds.transport.TCPv4.tls.tls.identity.certificate_chain_file**. For example, in the XML file:

```
<element>
    <name> dds.transport.TCPv4.tls.tls.identity.certificate_chain_file </name>
    <value>security/certificates/peer1.pem</value>
</element>
```

12.4.4 Improved performance with RTI TCP Transport when force_asynchronous_send is set to 1

In previous releases of the TCP Transport, the buffers in which RTPS messages were copied before being sent, when the property **dds.transport.TCPv4.tcp1.force_asynchronous_send** was set to 1, came from a pool that was shared across all the TCP connections. The size of this pool was configured using the transport properties **dds.transport.TCPv4.tcp1.write_buffer_allocation.(initial_count|max_count|incremental_count)**. When the number of buffers in the pool was exhausted, new messages were dropped.

This approach for asynchronous writing had two main drawbacks:

- The pool of buffers was shared across TCP connections. High-throughput connections may have starved low-throughput connections.
- When the pool was exhausted, new messages were dropped. Because of this, old data was prioritized over new data. In extreme cases, messages may not have been received on a TCP connection.

With this improvement, the behavior when **force_asynchronous_send** is set to 1 is changed as follows:

- There is a new property, **dds.transport.TCPv4.tcp1.shared_write_buffer_allocation**, which configures whether the pool of buffers is shared or exclusive per TCP connection. Setting this property to 0 (default value) will help with the starvation problem once the pool of resources is exhausted. You can still revert to old behavior by setting this property to 1.
- When there are no buffers left in the pool, a new message will replace the oldest message that is not currently in the process of being sent. This guarantees that new messages are prioritized, while at the same time not running into a situation in which messages are not received.

See the "Properties for NDDS_Transport_TCPv4_Property_t" table in the *Core Libraries User's Manual* for details.

12.4.5 View diagnostic information in TCP Transport log messages when using NDDS_Config_LogPrintFormat

Some of the error messages in the TCP Transport log were missing diagnostic information. For example:

```
Connection established to server at: 10.101.100.113:7401
```

Now, TCP Transport log messages apply the NDDS_Config_LogPrintFormat, which provides diagnostic information such as method name, activity context, and timestamp. See the section "Format of Logged Messages" in the *RTI Connext DDS Core Libraries User's Manual* for more information.

12.4.6 More robust TCP Transport creation behavior when disable_interface_tracking property set to true

TCP Transport creation may have failed when the **disable_interface_tracking** property was set to true. This may have only happened if the transport was configured to operate in either "TCP over LAN" or "TLS over LAN" mode.

This problem is fixed. TCP Transport will no longer fail creation when interface tracking is disabled.

12.4.7 Configure transport thread name using property <transport_property_prefix>.parent.thread_name_prefix

The transport thread name can now be configured using the property **<transport_property_prefix>.parent.thread_name_prefix**.

This property creates the thread name of the transport. The maximum size of the property is 8 characters.

For example:

```
<element>
    <name>dds.transport.UDPv4.builtin.parent.thread_name_prefix</name>
    <value>myPrefix</value>
</element>
```

If you do not set this property, *Connext DDS* automatically generates it as: **rTr<Participant Identifier>**:

- **r** specifies that the thread has been created by RTI Connext DDS.
- **Tr** identifies the transport module.
- **Participant Identifier** is five characters to identify the participant. (See 11.4.1 Identify Connext DDS threads more easily using updated and consistent names on page 18 for more details.)

The name of the thread, created as part of the transport, will be: <Thread Name Prefix><Transport name><Task type>. See "Identifying Threads Used by Connext DDS" in the *RTI Connext DDS Core Libraries User's Manual*.

12.4.8 UDP send_blocking property now supports string constant

Previously, the properties **dds.transport.UDPv4.builtin.send_blocking** and **dds.transport.UDPv6.builtin.send_blocking** only accepted "1" or "0".

This limitation has been resolved, and now those properties accept string constants, too:

- **dds.transport.UDPv4.builtin.send_blocking** and **dds.transport.UDPv4_WAN.builtin.send_blocking** support:

```
"0",
"1",
"NDDS_TRANSPORT_UDP_BLOCKING_NEVER",
"NDDS_TRANSPORT_UDPV4_BLOCKING_NEVER",
"TRANSPORT_BLOCKING_NEVER",
"NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS",
"NDDS_TRANSPORT_UDPV4_BLOCKING_ALWAYS",
"TRANSPORT_BLOCKING_ALWAYS"
```

- **dds.transport.UDPv6.builtin.send_blocking** supports:

```
"0",
"1",
"NDDS_TRANSPORT_UDP_BLOCKING_NEVER",
"NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER",
"TRANSPORT_BLOCKING_NEVER",
"NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS",
"NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS",
"TRANSPORT_BLOCKING_ALWAYS"
```

12.5 New Character Support in Filters

12.5.1 Use special characters in filter expressions using MATCH operator

Escaping special characters in the MATCH operator expressions is now supported. These special characters are: `,\'?*[]-^!`. Previously, it was not possible to match any one of these characters directly or to use them in the filter expression at all. For example, if the filter expression was "myString MATCH 'Won't Match'", the filter expression failed to compile:

```
[D0100|CREATE CFTopic|T=cft] DDS_SqlFilter_compileWithOptimizationLevel:SQL compiler failed
with error-code: -1 (Syntax error)
[D0100|CREATE CFTopic|T=cft] PRESParticipant_createContentFilteredTopic:content filter compile
error 1
[D0100|CREATE CFTopic|T=cft] DDS_ContentFilteredTopic_createI:!create DDS_ContentFilteredTopic
[D0100|CREATE CFTopic|T=cft] DDS_DomainParticipant_create_contentfilteredtopic_with_
filter:!create content filtered topic
create_cft error
```

Other special characters did not cause the expression to fail to compile, but they could not be matched. For example, '?' has special meaning: without an escape character, the filter expression "myString MATCH

'?'" returns all one-character strings; "myString MATCH 'h?'" returns all two-character strings starting with 'h.' Now, "myString MATCH 'h\?'" will return only the string 'h?'.

The matching rules have been updated so that every occurrence of a backslash ('\') followed by a character in the pattern is replaced by that character and not treated with any special meaning.

12.5.2 Use non-ASCII UTF-8 characters in filtering of IDL strings

The filtering features in previous releases did not support filtering samples based on the value of IDL string fields that contained non-ASCII UTF-8 characters. This means that non-ASCII UTF-8 characters were not allowed in filter expressions, filter parameters, or the value of IDL string members referenced by the filter expression. The usage of non-ASCII UTF-8 characters may have led to either errors parsing the filter expression or wrong results during the filter evaluation.

For example, the creation of a ContentFilterTopic with the following expression failed with the error messages below:

```
msg MATCH '\u0403*'
```

```
[D0056|CREATE CFTopic|T=CFT Example MyType] DDS_SqlFilter_compileWithOptimizationLevel:SQL
compiler failed to parse parameter string ''Ѓ''
[D0056|CREATE CFTopic|T=CFT Example MyType] DDS_SqlFilter_compileWithOptimizationLevel:SQL
compiler failed with error-code: -13 (Invalid parameter string)
[D0056|CREATE CFTopic|T=CFT Example MyType] PRESParticipant_createContentFilteredTopic:content
filter compile error 1
[D0056|CREATE CFTopic|T=CFT Example MyType] DDS_ContentFilteredTopic_createI:!create DDS_
ContentFilteredTopic
[D0056|CREATE CFTopic|T=CFT Example MyType] DDS_DomainParticipant_create_contentfilteredtopic_
with_filter:!create content filtered topic
```

This release adds full support for UTF-8 characters to the following filtering features:

- ContentFilteredTopics
- Query conditions
- TopicQueries
- MultiChannel *DataWriters*

Note that the name of the fields in the expression is still restricted to ASCII characters as described in the latest Interface Definition Language Version 4.0 (https://www.omg.org/spec/IDL/4.0/PDF).

Normalization

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

Precomposed Ḽ is represented by \u1e3c

Composed L̂ = L + ^ is represented by \u004c + \u032d

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

This new feature includes a *DomainParticipant* Property QoS property called **dds.domain_participant.filtering_unicode_normalization** that allows you to configure the normalization kind for UTF-8 strings that are part of the filter expression, the string filter parameters, or the value of IDL string members referenced by the filter expression.

The possible values of the normalization property are:

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

Because normalization may affect performance, the property allows disabling the normalization process per *DomainParticipant* using the value OFF.

Normalization only affects the filter evaluation. *Connext DDS* does not normalize the content of the IDL string fields when they are serialized and sent on the wire. It is the responsibility of your application to do that.

Character encoding

Connext DDS offers ISO 8859-1 as an alternative encoding for IDL strings. The default is UTF-8. In order to configure ISO 8859-1 for filtering, set the value of a new *DomainParticipant* Property QoS property **dds.domain_participant.filtering_character_encoding** to ISO-8859-1.

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

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

13 Language Bindings, APIs, XML Configuration

13.1 Print data state (sample, view, instance states) in Modern C++ using new operator<< definitions

New operator<< definitions have been added for for DataState, SampleState, ViewState, and InstanceState.

These new operator definitions provide a convenient way for applications to print changes in the data state. For example:

```
for (const auto& sample : reader.take()) {
    if (sample.info().valid()) {
        std::cout << sample.data() << std::endl;
    } else {
        std::cout << "Instance state changed to "
                  << sample.info().state().instance_state() << std::endl;
    }
}
```

13.2 Simplify Listener lifecycle management in Modern C++ API

Starting in this release, *Entities* expect their *Listeners* to be passed as a **std::shared_ptr**. Previously, *Listeners* were expected as raw pointers.

Example of the new API:

```
class MyReaderListener : public dds::sub::DataReaderListener<Foo> {
   // …
} ;

// …

auto my_listener = std::make_shared<MyReaderListener>();
dds::sub::DataReader<Foo> reader(subscriber, topic, qos, my_listener);
```

This change simplifies the lifecycle of the listener and its entity. Previously, an *Entity* with a *Listener* was "retained" (it wouldn't be automatically destroyed even if its reference count reached zero) to allow for the application to unset the listener and delete it.

Now an *Entity* with a *Listener* holds a reference to the **shared_ptr**, keeping the *Listener* alive while the *Entity* is alive. If the *Entity* reference count reaches zero, it is destroyed even if it has a *Listener*.

The following APIs have changed:

- Entity constructors now take a **shared_ptr** to the *Listener*. Constructors taking a raw pointer are deprecated and may be removed in a future version.
- New functions in each *Entity* called **set_listener** and **get_listener** have been added. They receive and return a **shared_ptr** to the listener. The previous functions that received and returned a raw pointer are deprecated and may be removed in a future version.
- The **rti::core::ListenerBinder** utility is deprecated because it is no longer needed and may be removed in a future version.

The deprecated constructor and *Listener* setters behave as they used to (they prevent the automatic destruction of the entity), so existing code that upgrades to this version will not see any difference. However, it is recommended that applications transition to the new APIs.

13.3 Introduced Remote Procedure Calls (RPC) - Experimental Feature

Remote Procedure Calls, or RPC, is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space.

RPC interfaces are defined in IDL, for example:

```
exception TooFastError {
};

@final
struct Coordinates {
    int32 x;
    int32 y;
};

@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed) raises(TooFastError);
    float get_speed();
    attribute string<128> name;
};
```

From this definition, Code Generator generates a client that can be used as follows:

```
Coordinates final_position = robot_client.walk_to(Coordinates(150, 200), 85.0f);
```

And a service skeleton:

```
class RobotControlExample : public RobotControl {
public:
    Coordinates walk_to(const Coordinates& destination, float speed) override
    {
    …
    }
    …
};
```

The client and service each run on a *DomainParticpant* and under the hood, they use the request-reply communication pattern: the *client* uses a *Requester* to send requests and receive replies; the *service* uses a *Replier* to receive the requests and send the replies.

> **Note:** RPC is an experimental feature available only on C++11, for certain platforms. See the Core Libraries Platform Notes for the supported architectures.

For more details, see the new "Remote Procedure Calls (RPC)—Experimental Feature" chapter in the *RTI Connext DDS Core Libraries User's Manual*.

13.4 GetTypeCode from a definition provided in an XML configuration file using the type name

Connext DDS has added a function to get a TypeCode from a definition provided in an XML configuration file using the type name.

Its usage in different API's is demonstrated in the following example:

XML definition

```
<types>
    <struct name="MyType">
        ...
    </struct>
</types>
```

[C]

```
const DDS_TypeCode * type = DDS_DomainParticipantFactory_get_typecode_from_config
(domainParticipantFactoryPtr, "MyType");
```

[Traditional C++]

```
const DDS_TypeCode * type = domainParticipantFactoryPtr->get_typecode_from_config("MyType");
```

[.NET]

```
DDS.TypeCode type = domainParticipantFactory.get_typecode_from_config("MyType");
```

[Java]

```
TypeCode type = domainParticipantFactory.get_typecode_from_config("MyType");
```

Note that the modern C++ API already provided this functionality through the QosProvider:

[Modern C++]

```
const DynamicType& type = qosProvider.extensions().type("MyType");
```

13.5 XML fields of type duration have unset tags default to 0 with a warning log message

The duration type tag has two subfields, <sec> and <nanosec>. Some QoS Policies that use these fields, such as the DEADLINE QoS Policy, set the default duration to INFINITE. Therefore, if you had set just one of these fields (such as <sec>, but not <nanosec>, or vice-versa), the resulting duration value was still INFINITE.

This problem is resolved in this release. Now if you set only one of these fields (<sec> or <nanosec>) in the XML file, the other value defaults to 0. (If you set neither one of them, the default duration for that policy would be used.) A warning message will also be logged by the parser specifying the parent tag, the missing subfield, and the line number.

13.6 Simple new component to process new data in a thread pool

This release introduces SampleProcessor, a new component that simplifies the code to process new data in a *DataReader*. A SampleProcessor uses an AsyncWaitSet and its thread pool to process each individual sample in a *DataReader* with a user-provided handler.

This component provides the concurrency benefits of a thread pool without the user code required to manually operate a WaitSet.

The following C++11 example shows how to create a *DataReader* and register a handler for new data using a SampleProcessor:

```
dds::sub::DataReader<Foo> reader(subscriber, topic);
rti::sub::SampleProcessor sample_processor;
sample_processor.attach_reader(reader, [](const rti::sub::LoanedSample<Foo>& sample) {
if (sample.info().valid()) {
std::cout << "Received " << sample.data() << std::endl;
}
});
// The handler is now called asynchronously for each sample received
```

SampleProcessor is available in the the following language bindings: C (**DDS_SampleProcessor**), modern C++ (**rti::sub::SampleProcessor**), and C# (**Rti.Dds.Subscription.SampleProcessor**).

14 Platform and Build Changes

14.1 Use of Core Libraries now supported on these additional platforms

This release adds support for these platforms:

- macOS® 10.15 (x64) (x64Darwin17clang9.0)
- QNX® Neutrino® 7.0.4 (x64) (x64QNX7.0.0qcc_gpp5.4.0, x64QNX7.0.0qcc_cxx5.4.0)
- QNX Neutrino 7.0.4 (Arm v8) (armv8QNX7.0.0qcc_gpp5.4.0, armv8QNX7.0.0qcc_cxx5.4.0)
- QNX Neutrino 7.0.4 (Arm v7) (custom supported platform armv7QNX7.0.0qcc_cxx5.4.0)
- Red Hat® Enterprise Linux® 7.6 (x64) (x64Linux3gcc4.8.2)
- Ubuntu® 18.04 LTS (Arm v7) (armv7Linux4gcc7.5.0)
- Ubuntu 18.04 LTS (Arm v8) (armv8Linux4gcc7.3.0)
- Ubuntu 20.04 LTS (x64) (x64Linux4gcc7.3.0)
- VxWorks® 7.0.0 SR0630 (x64) (x64Vx7SR0630llvm8.0.0.2[_rtp])
- Yocto Project® 2.5 (Arm v8) (custom supported platform armv8Linux4gcc7.3.0)

This release also adds support for the following POSIX-compliant platforms, which are made available with *RTI Connext TSS*:

- CentOS® 7.0 (x64) (x64Linux3gcc4.8.2FACE_GP)
- Red Hat Enterprise Linux 7, 7.3, 7.5, 7.5 (x64) (x64Linux3gcc4.8.2FACE_GP)
- Red Hat Enterprise Linux 8 (x64) (x64Linux4gcc7.3.0FACE_GP)
- Ubuntu 14.04 LTS (x64) (x64Linux3gcc4.8.2FACE_GP)
- Ubuntu 18.04 LTS (x64) (x64Linux4gcc7.3.0FACE_GP)

14.2 Use of Core Libraries no longer supported on these platforms

- 32-bit host bundles for Linux and Windows platforms
- AIX®
- Android™ 5.0, 5.1
- Debian® 7 (custom supported platform)
- Freescale™ Linux 1.4 (custom supported platform)
- INTEGRITY® 5.0.11
- iOS®
- LynxOS®
- macOS 10.12
- Red Hat Enterprise Linux 5.2 (custom supported platform)
- Solaris™
- SUSE Linux Enterprise Server 11
- Ubuntu 12.04 LTS
- VxWorks 653
- VxWorks 6.9.0 (pentiumVx6.9gcc4.3.3 and ppc604Vx6.9gcc4.3.3 only)
- Wind River Linux 7 (x64, and Arm v7 custom supported platform)
- Xlinx® Linux 14.2 (custom supported platform)
- Yocto Project 2.2 (custom supported platform)

14.3 Build applications for Linux architectures without using -lnsl flag

The **-lnsl** flag is no longer required when building applications for Linux architectures.

glibc 2.26 deprecated **libnsl** and **glibc** no longer includes the module in the **glibc** library.

The "Building Instructions for Linux Architectures" table in the *RTI Connext DDS Core Libraries Platform Notes* has been updated accordingly.

14.4 Generate build system once, and build Release and Debug configurations using fully supported multiconfiguration generators in FindRTIConnextDDS script

The FindRTIConnextDDS script now includes full support for multiconfiguration CMake® generators like Visual Studio® or Ninja Multi-Config projects. This new support allows developers to generate the build system once and build Release and Debug configurations.

14.5 Configure the development environment in Z shell using a new script

Connext DDS provides a number of scripts to configure the development environment in different operating systems and shells. Among other things, these scripts configure paths, library paths, and environment variables that are often required to build *Connext DDS* applications. This release adds support for the Z shell (ZSH). For that purpose, it includes a new script named **rtisetenv_<architecture>.zsh**, which is located in **<installation_directory>/resource/scripts** and can be "sourced" to configure the environment within a ZSH session.

15 Changes to Defaults

15.1 Default for WriterDataLifeCycleQosPolicy.autodispose_unregistered_instances changed to FALSE, now no longer applies during DataWriter deletion

In previous releases, the deletion of a Reliable *DataWriter*, where **writer_data_lifecycle.autodispose_unregistered_instances** is set to TRUE, may not have caused the *DataWriter's* registered instances to transition to DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE on the Reliable *DataReader* side. Instead, some instances may have transitioned to the DDS_NOT_ALIVE_NO_WRITERS state. This is due to one or multiple of the following reasons:

- There is a race condition in which a *DataReader* may have detected that the *DataWriter* is gone through discovery mechanisms before receiving and ACKing all dispose messages sent by the *DataWriter* for its instances during the execution of the **Publisher::delete_datawriter** operation.

- Reliable *DataWriters* configured with KEEP_ALL history will never send more than **min(max_send_window_size, max_samples)** dispose messages. The rest of the instances will never be disposed.

To address these issues, *DataWriter* deletion is now treated differently than an explicit call to **DataWriter::unregister_instance**. The **autodispose_unregistered_instances** setting in the WRITER_DATA_LIFECYCLE QoS Policy (**writer_data_lifecycle.autodispose_unregistered_instances**) no longer applies during *DataWriter* deletion and only applies when a *DataWriter* calls **unregister_instance** explicitly.

The default value for **writer_data_lifecycle.autodispose_unregistered_instances** has also been changed from TRUE to FALSE. Disposing an instance and unregistering from an instance are two distinct actions that a *DataWriter* can take. Disposing an instance indicates that the instance no longer exists, while unregistering from an instance means that the *DataWriter* will not be updating the instance anymore. Unregistering from an instance says nothing about the instance itself, and other *DataWriters* may still continue to update the instance. In the majority of use cases, it is better and more transparent to explicitly perform each action when appropriate in your application rather than relying on *Connext DDS* to perform either action automatically.

15.2 Default value for max_objects_per_thread increased from 1024 to 2048

The default value for **max_objects_per_thread** in the SYSTEM_RESOURCE_LIMITS QoS Policy has been increased from 1024 to 2048. This increase now allows you to create about 20 or 21 participants.

15.3 Default behavior for a Connext DDS application that detects an incorrect property name is now to log an error instead of a warning

Previously when you specified an incorrect property name via the PROPERTY QoS Policy, *Connext DDS* logged a warning similar to the following:

```
DDS_PropertyQosPolicy_validatePropertyNames:Unexpected property: dds.type_consistnecy.ignore_
sequence_bounds. Closest valid property: dds.type_consistency.ignore_sequence_bounds
```

The message contained the invalid property and the closest property name.

Now by default, *Connext DDS* logs an error when it does not recognize the property name, and stops the creation of the entity or the plugin. (See also 11.3.6 Configure validation of property names at plugin level on page 17.) You can configure the validation of the property by using **dds.participant.property_validation_action**. For more information, see the "PROPERTY QosPolicy (DDS Extension)" section of the *RTI Connext DDS Core Libraries User's Manual*.

15.4 Changes to Default Stack size for INTEGRITY Platforms

The default stack size for middleware-created threads has changed for INTEGRITY platforms:

Thread	New Value
Asynchronous Publisher and Asynchronous flushing thread	64*1024
Database thread	64*1024
Event thread	4*64*1024
ReceiverPool threads	4*64*1024

See the *RTI Connext DDS Core Libraries Platform Notes* for more details.

16 Performance Improvements

16.1 Improved support for concurrency in data reception events in DataReaders in a DomainParticipant

Increasing data reception concurrency across the *DataReaders* in a *DomainParticipant* requires:

1. Associating each *DataReader* with its own *Subscriber.*
2. Using a different receiver port per *DataReader* by configuring the QoS policy <unicast> or <multicast> on the DataReaderQos.

In previous releases, even after following the previous steps, there was a contention point that required each received thread to take the same *DomainParticipant* mutex after every packet is received. This mutex was also used by different events and operations within the middleware. An operation holding the mutex for a long time would have blocked data reception. This release improves concurrency by removing the need for a receive thread to take the *DomainParticipant* mutex that may have led to long delays in data reception.

16.2 Improved performance in response to a TopicQuery issued by a DataReader

This release introduced changes that may lead to a reduction in the time that it takes to respond to a TopicQuery issued by a *DataReader*.

The changes will be more noticeable in large systems as a function of the number of discovered endpoints.

17 Deprecations

17.1 Use of refilter field in HISTORY QoS Policy no longer supported

The **refilter** field in the HISTORY QoS Policy has been removed along with the associated public enum DDS_RefilterQosPolicyKind. If you are using this QoS setting, you will need to remove it from any source code and XML configuration files, and recompile your application.

The filtering behavior of a *DataWriter* can now only be controlled through the **max_remote_reader_filters** field in the DATA_WRITER_RESOURCE_LIMITS QoS Policy. The behavior of **max_remote_reader_filters** is as follows, and has not changed. The following descriptions assume that all other conditions have been met that allow for writer-side filtering:

- UNLIMITED (default): The *DataWriter* will filter for up to (2^31)-2 *DataReaders*. However, in this case, the *DataWriter* does not store the filtering result per sample per *DataReader*. Thus, if a

sample is resent (such as due to a loss of reliable communication), the sample will be filtered again.

- A finite value N: The *DataWriter* will filter for up to N *DataReaders*. The *DataWriter* will store the filtering result per sample per *DataReader*. Thus, if a sample is resent (such as due to a loss of reliable communication), the sample will not need to be filtered again.
- 0: The *DataWriter* will not perform writer-side filtering.

17.2 CORBA Compatibility Kit has been removed in this release

RTI CORBA Compatibility Kit is not supported in this release.

17.3 *RTI Prototyper* has been deprecated in this release

RTI Prototyper is deprecated starting with release 6.1.0, which is the last release that supports it. After release 6.1.0, *Prototyper* will not be supported. *RTI Connector* replaces it and supports more scripting languages.

17.4 Previous release's C# / .NET binding is deprecated

A new C# language binding for .NET 5 and .NET Standard 2.0 is available (see 2 New C# Language Binding Allows Building Multi-Platform Connext DDS Applications for .NET 5 on page 3), and will replace the previous binding. The previous .NET binding is still available, but deprecated, and will be removed in a future release.

17.5 Support for pre-C++11 compilers is deprecated

The *Code Generator* option **-language C++03** has been deprecated. This release will include a warning message during code generation that C++03 support will be removed in future releases. See the *Code Generator Release Notes* for more information.

17.6 -legacyPlugin option has been removed

The *Code Generator* option **-legacyPlugin** has been removed and is not supported in this release. See the *Code Generator Release Notes* for more information.

17.7 DynamicData::set_buffer and DynamicData::get_estiamted_max_buffer_size APIs have been removed

The **DynamicData::set_buffer** and **DynamicData::get_estiamted_max_buffer_size** APIs have been removed, having previously been deprecated. If you were using these APIs to get a Common Data Representation (CDR) of the DynamicData object, now use **DynamicData::to_cdr_buffer** for that instead.

17.8 RTI Secure WAN Transport may be deprecated in a future release

RTI may not support *Secure WAN Transport* in future versions of *Connext DDS*. Existing applications that use *Secure WAN Transport* should be updated to take advantage of *RTI Real-Time WAN Transport* as soon as feasible. All new applications should use *Real-Time WAN Transport*. (See 1 High-Performance WAN Connectivity over UDP that is Secure and Scalable, Using RTI Real-Time WAN Transport and RTI Cloud Discovery Service on page 1.)

17.9 RTI Spreadsheet Add-in for Microsoft Excel is deprecated in this release

Spreadsheet Add-in for Microsoft Excel is deprecated starting with release 6.1.0, which is the last release that supports it. After release 6.1.0, *Spreadsheet Add-in for Microsoft Excel* will not be supported. Source code for the plug-in will be available in the RTI Community Github repository (https://-github.com/rticommunity) when official support ends.