RTI Connext DDS

Core Libraries

Release Notes

Version 6.0.0

Printed in U.S.A. First printing.
February 2019.

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

Chapter 1 Introduction

RTI® Connext® DDS 6.0.0 is a general access release. This document describes fixes in the *Connext DDS* Core Libraries in 6.0.0. These enhancements have been made since 5.3.1 was released. This document includes the following:

For an overview of new features in 6.0.0, see RTI Connext DDS Core Libraries Whats New in 6.0.0.

Many readers will also want to look at additional documentation available online. In particular, RTI recommends the following:

- **Use the RTI Customer Portal** (http://support.rti.com) to download RTI software and contact RTI Support. The RTI Customer Portal requires a username and password. You will receive this in the email confirming your purchase. If you do not have this email, please contact **license@rti.com**. Resetting your login password can be done directly at the RTI Customer Portal.
- **The RTI Community Forum** (https://community.rti.com) provides a wealth of knowledge to help you use *Connext DDS*, including:
  - Documentation, at https://community.rti.com/documentation
  - Best Practices,
  - Example code for specific features, as well as more complete use-case examples,

    - Solutions to common questions,
    - A glossary,
    - Downloads of experimental software,
    - And more.
- Whitepapers and other articles are available from http://www.rti.com/resources.
- Performance benchmark results for *Connext* are published online at http://www.rti.com/products/dds/benchmarks.html. Updated results for new releases are typically published within two months after general availability of that release.

Chapter 2 System Requirements

2.1 Supported Operating Systems

Connext DDS requires a multi-threaded operating system. This section describes the supported host and target systems.

In this context, a host is the computer on which you will be developing a *Connext DDS* application. A target is the computer on which the completed application will run. A host installation provides the *RTI Code Generator* tool (*rtiddsgen*), examples and documentation, as well as the header files required to build a *Connext DDS* application for any architecture. You will also need a target installation, which provides the libraries required to build a *Connext DDS* application for that particular target architecture.

Connext DDS is available for the platforms in Table 2.1 Supported Platforms.

Table 2.1 Supported Platforms

Operating System	Version
AIX®	AIX 7.1
Android™ (target only)	Android 5.0, 5.1
INTEGRITY® (target only)	INTEGRITY 10.0.2, 11.0.4, 11.4.4
iOS® (target only)	iOS 8.2
Linux® (ARM® CPU)	NI™ Linux 3 Raspbian Wheezy 7.0 Ubuntu® 16.04 LTS
Linux (Intel® CPU)	CentOS™ 6.0, 6.2 - 6.4, 7.0 Red Hat® Enterprise Linux 6.0 - 6.5, 6.7, 6.8, 7.0, 7.3, 7.5 Ubuntu 12.04 LTS, 14.04 LTS, 16.04 LTS, 18.04 LTS SUSE® Linux Enterprise Server 11 SP2, 11 SP3, 12 Wind River® Linux 7

Table 2.1 Supported Platforms

Operating System	Version
LynxOS® (target only)	LynxOS 5.0
OS X®	OS X 10.11 - 10.13
QNX® (target only)	QNX Neutrino® 6.4.1, 6.5, 7.0
Solaris™	Solaris 2.10 (Only available by request.)
VxWorks® (target only)	VxWorks 6.9, 6.9.3.2, 6.9.4.2, 6.9.4.6, 7.0 VxWorks 653 2.3 (Only available by request.)
Windows®	Windows 7, 8, 8.1, 10[a] Windows Server 2008 R2 Windows Server 2012 R2 Windows Server 2016

The following table lists additional target libraries for which RTI offers custom support. If you are interested in using one of these platforms, please contact your local RTI sales representative or email **sales@rti.com**.

Table 2.2 Custom Supported Platforms

Operating System	Version
INTEGRITY	INTEGRITY 5.0.11 on MPC 8349 CPU
INtime® for Windows	INtime 6.3 with Visual Studio 2017 on x86 CPU[b]
Linux	Debian® 7 on ARMv7 CPU
	Freescale™ 1.4 on QorIQ or P4040/P4080/P4081 CPU
	Red Hat Enterprise Linux 5.2 on x86 CPU
	RedHawk™ Linux 6.0 on x64 CPU
	RedHawk Linux 6.5 on i86 and x64 CPUs
	Wind River Linux 7 and 8 on ARMv7 CPU
	Xilinx® 14.2 on ARMv7 CPU
	Yocto Project® 2.2 on 64-bit ARMv8 CPU

[a]Per Microsoft, this should be compatible with Windows 10 IoT Enterprise with Windows native app.
[b]Tested on 64-bit Windows 10 operating system.

Table 2.2 Custom Supported Platforms

Operating System	Version
QNX	QNX 6.5 on PPC E500 v2 CPU
	QNX 6.6 on ARMv7 and x86 CPUs
	QNX 7.0 on ARMv7 CPU
VxWorks	VxWorks 6.9.3 on ARMv7 and PPC CPUs VxWorks 6.9.3.2 on MIPS CPU VxWorks 653 2.5.0.2 on PPC e500v2 CPU

See the *RTI Connext DDS Core Libraries Platform Notes* for more information on each platform.

2.2 Requirements when Using Microsoft Visual Studio

Note: Debug versions of applications and the various Visual C++ DLLs are not redistributable. Therefore, if you want to run debug versions, you must have the compiler installed.

When Using Visual Studio 2010 — Service Pack 1 Requirement

You must have Visual Studio 2010 Service Pack 1 or the Microsoft Visual C++ 2010 SP1 Redistributable Package installed on the machine where you are *running* an application linked with dynamic libraries.

This includes dynamically linked C/C++ and all .NET and Java applications. To run an application built with debug libraries of the above RTI architecture packages, you must have Visual Studio 2010 Service Pack 1 installed.

The Microsoft Visual C++ 2010 Service Pack 1 Redistributable Package can be obtained from the following Microsoft websites:

For x86 architectures:https://www.microsoft.com/en-us/download/details.aspx?id=8328

For x64 architectures: https://www.microsoft.com/en-us/download/details.aspx?id=13523

When Using Visual Studio 2012 — Update 4 Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2012 Update 4 installed on the machine where you are *running* an application linked with dynamic libraries. This includes dynamically linked C/C++ and all .NET and Java applications.

You can download Visual C++ Redistributable for Visual Studio 2012 Update 4 from this Microsoft website: http://www.microsoft.com/en-ca/download/details.aspx?id=30679

When Using Visual Studio 2013 — Redistributable Package Requirement

You must have Visual C++ Redistributable for Visual Studio 2013 installed on the machine where you are *running* an application linked with dynamic libraries. This includes C/C++ dynamically linked and all

.NET and Java applications.

You can download Visual C++ Redistributable for Visual Studio 2013 from this Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=40784

When Using Visual Studio 2015 — Update 3 Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2015 Update 3 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2015 Update 3 from this Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=53840.

When Using Visual Studio 2017 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2017 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2017 from this Microsoft website: https://www.visualstudio.com/downloads. Look for "Microsoft Visual C++ Redistributable for Visual Studio 2017" in the section called "Other Tools and Frameworks."

2.3 Disk and Memory Usage

Disk usage for a typical host-only installation is approximately 802 MB on Linux systems and 821 MB on Windows systems. Each additional architecture (host or target) requires an additional 498 MB on Linux systems and 609 MB on Windows systems.

We recommend that you have at least 256 MB RAM installed on your host development system. The target requirements are significantly smaller and they depend on the complexity of your application and hardware architecture.

Chapter 3 Compatibility

Below is basic compatibility information for this release.

For backward compatibility information between 6.0.0 and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

3.1 Wire Protocol Compatibility

Connext DDS communicates over the wire using the formal Real-time Publish-Subscribe (RTPS) protocol. RTPS has been developed from the ground up with performance, interoperability and extensibility in mind. The RTPS protocol is an international standard managed by the OMG. The RTPS protocol has built-in extensibility mechanisms that enable new revisions to introduce new message types, extend the existing messages, or extend the Quality of Service settings in the product—without breaking interoperability.

RTPS 1.0 was introduced in 2001. The current version is 2.3. RTI plans to maintain interoperability between middleware versions based on RTPS 2.x.

3.2 Code and Configuration Compatibility

The *Connext DDS* core uses an API that is an extension of the OMG Data Distribution Service (DDS) standard API, version 1.4. RTI strives to maintain API compatibility between versions, but will conform to changes in the OMG DDS standard.

The *Connext DDS* core primarily consists of a library and a set of header files. In most cases, upgrading simply requires you to recompile your source using the new header files and link the new libraries. In some cases, minor modifications to your application code might be required; any such changes are noted in the *Migration Guide* on the RTI Community Portal (https://-community.rti.com/documentation). The *Migration Guide* also indicates whether and how to regenerate code.

3.3 Extensible Types Compatibility

This release of *Connext DDS* includes partial support for the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification[1] from the Object Management Group (OMG), version 1.2. This support allows systems to define data types in a more flexible way, and to evolve data types over time without giving up portability, interoperability, or the expressiveness of the DDS type system.

For information related to compatibility issues associated with the Extensible Types support, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). See also the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types* for a full list of the supported and unsupported extensible types features.

3.4 ODBC Database Compatibility

To use the Durable Writer History and Durable Reader State features, you must install a relational database such as MySQL.

To see if a specific architecture has been tested with the Durable Writer History and Durable Reader State features, see the *RTI Connext DDS Core Libraries Platform Notes*. To see what databases are supported, see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Database Setup*.

[1] See the specification at http://www.omg.org/spec/DDS-XTypes/.

Chapter 4 What's Fixed in 6.0.0

Release 6.0.0 is a general access release based on the maintenance release 5.3.1[1]. This section describes bugs fixed in the Core Libraries in 6.0.0.

For an overview of new features and improvements in the Core Libraries in 6.0.0, please see RTI Connext DDS Core Libraries Whats New in 6.0.0.

4.1 Fixes Related to Discovery

4.1.1 Endpoint discovery initialization errors during participant creation left participant in inconsistent state

If an error occurred during a participant creation's endpoint discovery initialization, the initialization may have incorrectly reported success. In this situation, the participant was left in an inconsistent state. Trying to enable the participant would have failed with errors similar to the following:

```
[D0064|CREATE Participant|D0064|ENABLE]DISCSimpleEndpointDiscoveryPlugin_
enable:!precondition: me == NULL || worker == NULL
[D0064|CREATE Participant|D0064|ENABLE]DDS_DomainParticipantDiscovery_enableI:!enable
simple endpoint discovery plugin
[D0064|CREATE Participant|D0064|ENABLE]DDS_DomainParticipant_enableI:!enable discovery
```

This problem is now resolved.

[RTI Issue ID CORE-8249]

4.1.2 Writer/reader resource limits affected wrong builtin endpoints

Certain members of the DomainParticipantResourceLimitsQosPolicy affected the wrong endpoint-discovery builtin endpoints. For example, **writer_property_list_max_length** determined the max-

[1]For What's Fixed in 5.3.1, see the *RTI Connext DDS Core Libraries Release Notes* provided with 5.3.1.

imum number of properties that the SubscriptionBuiltinTopicDataDataWriter could serialize when it sent discovery information about *DataReaders*. Consequently, the following scenario would fail:

1. Set **reader_property_list_max_length** to one.
2. Set **writer_property_list_max_length** to zero.
3. Create a reader with one property.
4. Create no writers.
5. The reader's discovery information would fail to be sent.

The following members of DomainParticipantResourceLimitsQosPolicy had this kind of problem:

- **writer_property_list_max_length**
- **reader_property_list_max_length**
- **writer_user_data_max_length**
- **reader_user_data_max_length**
- **publisher_group_data_max_length**
- **subscriber_group_data_max_length**

This problem has been resolved by making writer resource limits affect the PublicationBuiltinTopic endpoints and reader resource limits affect the SubscriptionBuiltinTopic endpoints.

[RTI Issue ID CORE-8411]

4.1.3 Failure to send TypeObject when type had base with no members

Connext DDS applications publishing or subscribing to a topic whose type definition had a base type with no members may have failed to send the TypeCode during endpoint discovery.

This problem may have affected only systems running *Connext DDS* 5.0.0+ applications along with pre-5.0.0 applications, or RTI tools that required the availability of the type information to work. Systems running only 5.0.0 and above relied on the TypeObject, which was not affected by this problem.

[RTI Issue ID CORE-8526]

4.1.4 Received incorrect QoS policy values through discovery when communicating with other vendors

During the discovery process, if a remote participant omitted a QoS policy because it was set to the default value, *Connext DDS* may have set that QoS policy to a default value not matching the OMG DDS and RTPS specifications. This issue affected the following QoS policies:

- DDS_ParticipantBuiltinTopicData::lease_duration
- DDS_ParticipantBuiltinTopicData::reachability_lease_duration
- DDS_PublicationBuiltinTopicData::reliability
- DDS_SubscriptionBuiltinTopicData::reliability
- DDS_PublicationBuiltinTopicData::ownership_strength

This issue only affected communication with other vendors, since *Connext DDS* never omits the propagation of these QoS policies.

This issue is now fixed: the above-mentioned QoS policies that are not received from other vendors are now set to the default value defined in the OMG DDS and RTPS specifications.

[RTI Issue ID CORE-8882]

4.2 Fixes Related to Reliability Protocol

4.2.1 VOLATILE DataReader may have received historical samples from DataWriter

A VOLATILE *DataReader* may have received some historical samples from a matching *DataWriter* after being created.

Specifically, the *DataReader* would have received all the samples from the *DataWriter* queue that were not acknowledged by all the matching *DataReaders* at the moment the new *DataReader* was discovered.

This problem has been resolved. Now a VOLATILE *DataReader* does not receive historical samples from a *DataWriter*.

[RTI Issue ID CORE-4923]

4.2.2 Wrong RTPS GAP messages emitted by reliable DataWriters in some cases

A reliable *DataWriter* may have sent invalid GAP messages to a VOLATILE reliable *DataReader* after receiving a preemptive ACK or after receiving a NACK for samples that the *DataReader* had already received. These messages did not cause the *DataReader* to misbehave, but they could have led to issues when interoperating with other DDS vendors.

This issue, which was incorrectly marked as fixed in 5.3.0, is now resolved. A reliable *DataWriter* now sends correctly formatted GAPs in this case.

[RTI Issue ID CORE-7836]

4.2.3 Individual sample fragment losses may have triggered full sample repair

In some cases, losing individual fragments for a sample may have triggered the repair of the whole sample, as opposed to a repair of the missing fragments. This incorrect behavior resulted in excessive bandwidth usage.

This problem has been resolved. Now individual fragment losses are always repaired individually.

[RTI Issue ID CORE-8354]

4.2.4 DataReader may have ignored some piggyback heartbeats, leading to performance degradation

A *DataReader* may have ignored some piggyback heartbeats from a *DataWriter* when the *DataReader* used a content-filtered topic that led to writer-side filtering or when the *DataReader* created TopicQueries. By not processing the piggyback heartbeats, the *DataReader* may not have sent ACKNACK messages to the *DataWriter*. This problem may have caused the send window to fill up and the *DataWriter* to unnecessarily block on a **write()** call. This problem has been resolved.

[RTI Issue ID CORE-8908]

4.2.5 max_bytes_per_nack_response was ignored when using asynchronous publication

A *DataWriter* setting publish_mode to DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS ignored the protocol setting **max_bytes_per_nack_response**.

This meant that the number of samples that were sent in response to a NACK message from a *DataReader* was not limited based on the **max_bytes_per_nack_response** setting. This issue has been resolved.

[RTI Issue ID CORE-8973]

4.2.6 Disabling positive ACKs may have caused segmentation fault in publishing application

Disabling positive ACKs by setting **protocol.disable_positive_acks** to TRUE on the *DataWriter* or *DataReader* may have led to a rare segmentation fault. This problem has been fixed.

[RTI Issue ID CORE-9144]

4.2.7 Unexpected DDS_RETCODE_OUT_OF_RESOURCES error when writing a sample

The **DataWriter::write operation** may have failed with an unexpected DDS_RETCODE_OUT_OF_RESOURCES error when the *DataWriter* had enough resources to accept the sample that was being

written.

This error occurred only when:

- Communication was reliable.
- The *DataWriter* set <max_samples> to a finite value.
- Some of the *DataReaders* matching with the *DataWriter* used a ContentFilteredTopic.

This problem has been fixed.

[RTI Issue ID CORE-9190]

4.3 Fixes Related to Instance Management and Lifecycle

4.3.1 Instances may not have transitioned to NOT_ALIVE_NO_WRITERS on DataReader matching with a DataWriter using MultiChannelQosPolicy

Some instances may not have transitioned to NOT_ALIVE_NO_WRITERS state when a *DataReader* stopped matching with a *DataWriter* configured to use MultiChannel by setting **writer_qos.multi_channel**. This problem might have occurred only for instances that were not published in all channels. This problem has been resolved.

[RTI Issue ID CORE-8458]

4.3.2 DomainParticipantFactory::get_instance always returned success

In release 5.3.0, the operation **DomainParticipantFactory::get_instance** always returned success even if there was an error executing it. This problem has been resolved.

[RTI Issue ID CORE-8518]

4.3.3 DataReaders may not have purged samples from instances in NOT_ALIVE_NO_WRITERS state when autopurge_nowriter_samples_delay was set to finite value

A *DataReader* may not have purged samples from instances that transitioned to NOT_ALIVE_NO_WRITERS state when **reader_qos.reader_data_lifecycle.autopurge_nowriter_samples_delay** was set to a finite value.

This problem only occurred when *DataWriters* did not unregister instances explicitly by calling the unregister operation. For example:

- When a *DataWriter* was shut down gracefully or ungracefully without calling unregister for the instances that were in its cache.

- When a *DataWriter* became incompatible with a *DataReader* because of a QoS change (for example, a partition).

This issue affected 5.2.x and 5.3.x releases. This problem has been fixed.

[RTI Issue ID CORE-8522]

4.3.4 Instances in NOT_ALIVE_DISPOSED state may not have been purged from DataReader queue when autopurge_disposed_instances_delay was set to zero

Instances in NOT_ALIVE_DISPOSED state without samples may not have been removed from a *DataReader* queue configured with **reader_qos.reader_data_lifecycle.autopurge_disposed_instances_delay** set to zero.

This problem was likely to occur in the following scenarios:

- Scenarios in which the *DataReader* received instances from redundant *Routing Services*.
- Scenarios in which the *DataReader* received instances from *Persistence Service*.
- Scenarios in which destination order was set to BY_SOURCE_TIMESTAMP.

Because the instances were not removed from the queue, unbounded memory growth could have occurred when **reader_qos.resource_limits.max_instances** was configured to UNLIMITED, or sample rejection may have occurred with reason SAMPLES_PER_INSTANCE_LIMIT when **reader_qos.resource_limits.max_instances** had a finite value.

This problem has been fixed.

[RTI Issue ID CORE-8538]

4.3.5 A DataReader may have failed to calculate the keyhash for a sample containing zero-length strings

A *DataReader* may have failed to calculate the keyhash for a sample containing zero-length strings. This would occur only if the strings were not a part of the key themselves and were declared before at least one of the members of the type that was part of the key.

A *DataReader* calculates the keyhash for samples coming from *DataWriters* with **DataWriterProtocolQosPolicy.disable_inline_keyhash** set to TRUE. If keyhash calculation fails, a sample may be incorrectly added to the wrong instance and the correct instance will not be found in the *DataReader*.

This issue has been resolved.

[RTI Issue ID CORE-8603]

4.3.6 Incorrect warning reported while trying to purge disposed instances proactively

When **autopurge_dispose_instances_delay** is not DURATION_INFINITE, *Connext DDS* attempts to purge disposed instances proactively, independently of hitting resource limits. The warning "Writer-HistoryMemoryPlugin_dropFullyAckedDisposedInstance:unregistered instances not fully acked" was logged during this process; however, this warning is valid only when resource limits are hit, not when instances are proactively removed. This problem has been resolved.

[RTI Issue ID CORE-8629]

4.3.7 Purging disposed or unregistered instances based on source timestamp does not work when internal clock is set to monotonic

This issue applied only to release 5.3.0.8 when the internal clock for the participant was set to monotonic. In this case, setting the property **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay** to 1 on a *DataWriter* in order to purge disposed or unregistered instances using the source timestamp, did not work when the purging period was finite. (The purging period is configurable using the QoS values **writer_data_lifecycle.autopurge_disposed_instances_delay** and **writer_data_lifecycle.autopurge_unregistered_instances_delay**.)

This problem has been fixed.

[RTI Issue ID CORE-8637]

4.3.8 Possible leak upon application exit after using NDDSConfigVersion::get_instance() (Traditional C++ API only)

Tools such as Valgrind may have reported a memory leak upon application exit (not a recurring leak) when **NDDSConfigVersion::get_instance()** was called. This leak has been resolved.

[RTI Issue ID CORE-8686]

4.3.9 Unexpected errors when receiving a dispose sample for unbounded DDS_KeyedString BuiltinType topic

When a reader received a dispose sample without a keyhash, with a serialized key for an unbounded DDS_KeyedString bultin type topic, the following errors were logged and the dispose sample was not received:

```
DDS_KeyedStringPlugin_get_serialized_sample_size:value cannot be NULL
PRESCstReaderCollator_addRegisteredInstanceEntry:!serialize key
PRESCstReaderCollator_addInstanceEntry:!add registered instance
DDS_KeyedStringPlugin_get_serialized_sample_size:value cannot be NULL
PRESCstReaderCollator_addRegisteredInstanceEntry:!serialize key
PRESCstReaderCollator_addInstanceEntry:!add registered instance
```

This problem is now resolved: the dispose sample is now received and no errors are logged.

[RTI Issue ID CORE-8747]

4.3.10 instance_replacement not applied correctly for durable DataWriters

If you set <max_instances> to a finite value, <instance_replacement> may not have been applied correctly. That is, when <max_instances> was exceeded, the *DataWriter* may have replaced an instance that did not meet the replacement criteria defined in <instance_replacement>.

For example, if you set the instance replacement to DDS_DISPOSED_INSTANCE_REPLACEMENT, when <max_instances> was exceeded the *DataWriter* may have chosen for replacement an instance or multiple instances that were not in the DISPOSED state.

This problem has been resolved.

[RTI Issue ID CORE-8829]

4.3.11 Incorrect warning reported when replacing DISPOSE/ALIVE instance on a DataWriter

When a *DataWriter* was configured with an instance_replacement different than DDS_UNREGISTERED_INSTANCE_REPLACEMENT, *Connext DDS* incorrectly printed the following warning while trying to replace an instance, even if the replacement was successful:

```
WriterHistoryMemoryPlugin_dropEmptyAndFullyAckedUnregisteredInstance:no unregistered instances
```

This problem has been resolved.

[RTI Issue ID CORE-8902]

4.4 Fixes Related to Content Filters and Query Conditions

4.4.1 Possible crash in creation of a content filter for a type with an aliased base type

Applications creating an SQL content filter for a topic-type that inherits from an aliased type crashed in all the languages' APIs except for the C API.

For example, the creation of a *DataReader* with a ContentFilteredTopic based on the following IDL type "Foo" caused this problem (except in the C API):

```
struct Base { long x; };
typedef Base BaseAlias;
struct Foo : BaseAlias { long y; };
```

This problem has been resolved.

[RTI Issue ID CORE-8462]

4.4.2 Reading samples by instance with QueryCondition returned no data when using TOPIC or GROUP PresentationQosPolicy access_scope

An application using the PresentationQosPolicy **access _scope** TOPIC or GROUP while **ordered_access** = TRUE was not able to retrieve data using the **DataReader::take_next_instance_w_condition**, **DataReader::read_next_instance_w_condition**, **DataReader::take_instance_w_condition**, and **DataReader::read_instance_w_condition** APIs when the condition being used was a *QueryCondition*. In this case, these APIs returned DDS_RETCODE_NO_DATA when in fact there was data matching the requested instance and condition. (The data could be retrieved successfully using other **read** or **take** APIs.) This issue has been resolved.

[RTI Issue ID CORE-8508]

4.4.3 Content Filter issue when filtering on a member of the base type, for types with inheritance

When using a Content Filter on a type with inheritance, if the filter expression required evaluating a member of the base type, the result of the filter was incorrect. This issue has been fixed. Now the filter expression is applied properly, and the result is correct.

[RTI Issue ID CORE-9035]

4.5 Fixes Related to TopicQueries

4.5.1 DataWriter may have deadlocked if receiving "continuous" TopicQueries

A race condition may have caused a thread to deadlock when a *DataWriter* dispatched a continuous TopicQuery. This problem stopped the *DataWriter* from operating normally.

This deadlock could happen only if **topic_query_dispatch.samples_per_period**, in the DataWriterQos, was set to unlimited (the default value), and the *DataWriter* received a TopicQuery with continuous selection kind (the default was history snapshot, not continuous).

This problem affected release 5.3.0.8 only and has been fixed in this release.

[RTI Issue ID CORE-8448]

4.5.2 Instances may not have transitioned to NOT_ALIVE_NO_WRITERS on DataReader matching with a DataWriter with TopicQuery enabled

Some instances may not have transitioned to NOT_ALIVE_NO_WRITERS state when a *DataReader* stopped matching with a *DataWriter* with TopicQuery enabled (**writer_qos.topic_query_dispatch.enable** = TRUE). For example, this problem could have occurred if the *DataWriter* was unmatched due to a partition change. This problem has been fixed.

[RTI Issue ID CORE-8452]

4.5.3 Error when changing partition, group data, or topic data on a Publisher containing a DataWriter with TopicQuery enabled

Changing a partition or the group data on a Publisher containing a *DataWriter* with TopicQuery enabled (**writer_qos.topic_query_dispatch.enable** = TRUE) caused the following error:

```
PRESPsService_assertRemoteEndpoint:!assert pres psRemoteWriter
PRESPsService_notifyOfGroupDataOrPartitionChange:!assert remote endpoint
```

The issue also occurred with a slightly different error message when the topic data for the Topic associated with the Publisher's *DataWriter* was updated.

This problem has been fixed.

[RTI Issue ID CORE-8453]

4.5.4 Possible increasing memory and CPU usage in publishing applications using TopicQueries

A publishing application that enables TopicQuery may have been subject to increasing CPU and memory usage.

This growth was in linear proportion to the preceding number of TopicQueries times the preceding number of samples selected, but growing with respect to a sample only for as long as such a sample remained present in the *DataWriter*'s history. That is, removing a sample from the history would reset the contribution to this growth attributed to that sample.

The *DataWriter* keeps state for each TopicQuery it receives. This state can be deleted when all the samples that the TopicQuery selects have been delivered and acknowledged; however, some state information was not removed until the samples were replaced (for example, when updating an instance and exceeding the history depth).

This problem has been resolved.

[RTI Issue ID CORE-8817]

4.5.5 Spurious log message related to TopicQuery has been removed

The following log message may have been incorrectly displayed with "local" verbosity:

```
PRESPsService_dispatchMatchingTopicQueries:ignoring TopicQuery for this DataWriter because it
doesn't enable them
```

This problem happened when a *DataWriter* discovered a new *DataReader*, regardless of the existence of a TopicQuery.

The log message will no longer be displayed in that situation. It will only be displayed when a *DataWriter* that doesn't enable TopicQueries receives a TopicQuery.

[RTI Issue ID CORE-8820]

4.5.6 DataReader::getKey did not work with TopicQueries

The **DataReader::getKey** API may have returned DDS_RETCODE_BAD_PARAMETER for an instance that was known to a *DataReader* if samples for that instance were only in the *DataReader*'s queue in response to a TopicQuery.

This problem has been resolved. The API now searches for the instance in the queues associated with live data as well as TopicQuery data.

[RTI Issue ID CORE-8953]

4.5.7 TopicQuery samples that failed to be written a first time may have never been sent

If the write operation that publishes TopicQuery samples failed (for example, due to a timeout because of blocking), the sample that failed to be written may never have been sent. This error would happen in one of two cases, given a sample with sequence number n that failed to be written:

- The next sample in the writer queue that matched the TopicQuery expression had a sequence number > n + 1.
- The sample that failed to be written was the last TopicQuery sample in the writer's queue, and the last sample in the writer's queue at the time of the write error had a sequence number > n + 1.

This issue has been resolved. Now if the internal write call for a TopicQuery sample fails, the sample will be sent during the next TopicQuery publication period.

[RTI Issue ID CORE-9188]

4.6 Fixes Related to DynamicData

4.6.1 DynamicData had limitations with members larger than 65,535 bytes

DynamicData did not support resizing or out-of-order assignment of members that were longer than 65,353 bytes. Doing one of these two operations would result in the following error:

```
sparsely stored member exceeds 65535 bytes
```

There are no longer any restrictions regarding a member's size when using any of the DynamicData APIs in any order.

[RTI Issue ID CORE-3177]

4.6.2 Copying into a bound DynamicData object did not work

Using the **DynamicData::copy** operation did not work properly when copying into a DynamicData object that was bound to another object. The operation may have caused corruption of the destination object.

This issue has been resolved.

[RTI Issue ID CORE-3385]

4.6.3 Corrupt DynamicData objects containing sequences with length 0

In some cases, a DynamicData object may have been corrupted if a sequence member with length 0 was set in the object. This would only occur with sequences of primitives with a size greater than 4: double, long long, unsigned long long, long double. The corruption would have been noticeable when set and get operations performed on the DynamicData object returned errors or incorrect values.

This issue has been resolved.

[RTI Issue ID CORE-5161]

4.6.4 Binding to members of a sequence incorrectly created members in the DynamicData API

Binding to member n of a sequence in a DynamicData object that did not previously exist incorrectly created members 0 through n-1 of the sequence, even when the member was unbound without setting any values.

This issue has been resolved. Now, if a member of a sequence is bound and unbound without calling any set APIs, the length of the sequence that was bound to remains unchanged.

[RTI Issue ID CORE-5800]

4.6.5 Error when unbinding from a union DynamicData object

In some rare situations, unbinding from a union member may have returned RETCODE_ERROR with the following error message:

```
DDS_DynamicData_unbind_complex_member:internal error 1 trying to to stream
```

This issue has been resolved.

[RTI Issue ID CORE-6657]

4.6.6 Performance degradation when setting large sequences using the DynamicData API

If a DynamicData object's type contained any optional members, setting a large sequence (tens or hundreds of thousands of elements) using any of the **DynamicData::set_*_seq** APIs may have taken a very long time, on the order of minutes. The sequence itself did not have to be an optional member in order for this issue to occur.

This issue has been resolved.

[RTI Issue ID CORE-6979]

4.6.7 DynamicData::is_member_key did not work for types using inheritance

The **DynamicData::is_member_key** API did not work correctly for members that were part of a base type of the DynamicData object's type. The API may have returned incorrect results for whether or not a member is a key.

This issue has been resolved.

[RTI Issue ID CORE-7505]

4.6.8 DynamicData::set_complex_member did not work with aliased typecodes

The **DynamicData::set_complex_member** API failed with RETCODE_ERROR and an error message similar to the following:

```
DDS_DynamicData_set_complex_member:type mismatch for field aField (id=1)
```

if the complex member being set had a TypeCode with kind TK_ALIAS.

This issue has been resolved.

[RTI Issue ID CORE-7561]

4.6.9 DynamicData::set_string API did not accept NULL strings

The **DynamicData::set_string** API did not accept a NULL string. This behavior was not parallel with generated types, which allowed setting and sending NULL strings.

This issue has been fixed. The **DynamicData::set_string** API now accepts a NULL string. A string with length 0 containing only the null-terminator will be set in the DynamicData object.

[RTI Issue ID CORE-8163]

4.6.10 Large memory allocation when binding to large sequences in the DynamicData API

There was a large memory allocation when calling the **DynamicData::bind_complex_member** API on a large sequence of non-primitive members. More concretely, the allocation was (4 * max length of the sequence) bytes.

This issue has been resolved. Binding to a sequence no longer requires this memory allocation.

[RTI Issue ID CORE-8358]

4.6.11 DynamicData::from_cdr_buffer API did not resize DynamicData object

The **DynamicData::from_cdr_buffer** API did not resize the DynamicData buffer. This meant that the API failed if the same DynamicData object was used in repeated calls to the API in which the CDR buffer was larger than in previous calls. The following errors were printed in these situations:

```
DDS_DynamicDataTypePlugin_parametrized_cdr_to_cdr:deserialization error: insufficient space
DDS_DynamicDataTypePlugin_parametrized_cdr_to_cdr:error converting from extended CDR to CDR
DDS_DynamicDataTypePlugin_deserialize:error converting from extended CDR to CDR
DDS_DynamicData_from_cdr_buffer:deserialization error: buffer
DDS_DynamicData_from_cdr_buffer error 1
```

This issue has been resolved. The **DynamicData::from_cdr_buffer** will resize the DynamicData object's internal buffer if required to accommodate the deserialized sample.

[RTI Issue ID CORE-8394]

4.6.12 Error when unbinding from a DynamicData object

In rare situations, unbinding from a member may have returned RETCODE_ERROR with the following error message:

```
DDS_DynamicData_unbind_complex_member:internal error 1 trying to assert complex member
```

This issue has been resolved.

[RTI Issue ID CORE-8386]

4.6.13 Accessing a member of an array or sequence by member ID failed for member IDs > 65535

Using any of the **DynamicData::get_*** APIs on a sequence or array and providing a member ID greater than 65535 failed with DDS_RETCODE_NO_DATA, even if the member existed.

This issue has been resolved.

[RTI Issue ID CORE-8561]

4.6.14 DynamicData DataReader may have failed to correctly deserialize key for types containing strings

If a DynamicData *DataReader* received a keyed sample with a zero-length string, it may have failed to correctly deserialize the key. This problem occurred if the string was not part of the key and came before at least one of the key members in the type definition. When this issue occurred, the following error would be printed:

```
"DDS_DynamicDataUtility_skip_compact_type:stream error trying to access string"
```

This issue has been resolved.

[RTI Issue ID CORE-8604]

4.6.15 Missing DynamicData::clear_optional_member API in the .NET API

In previous releases, the **DynamicData::clear_optional_member** API was missing from the .NET API. This issue has been resolved.

[RTI Issue ID CORE-9077]

4.6.16 DynamicData::clear_optional_member API incorrectly returned error for unset optional members

Calling the **DynamicData::clear_optional_member** API on an unset optional member returned PRECONDITION_NOT_MET and produced the following error message:

```
DDS_DynamicData_clear_optional_member:cannot clear non-optional members
```

This issue has been resolved. The **DynamicData::clear_optional_member** now returns RETCODE_OK when called on an already unset optional member.

[RTI Issue ID CORE-9092]

4.6.17 DynamicData APIs did not check for an associated TypeCode

The DynamicData APIs were not consistent in checking for a TypeCode associated with the DynamicData object before trying to execute the operation. This inconsistency resulted in unclear errors messages and inconsistent return codes.

This issue has been resolved. DynamicData APIs that require the DynamicData object to have an associated TypeCode now check for a TypeCode and return RETCODE_PRECONDITION_NOT_MET when there is no TypeCode associated with the DynamicData object.

[RTI Issue ID CORE-9093]

4.6.18 Setting members past a sequence's maximum bound was not prohibited in the DynamicData API

Setting a member past a sequence's maximum bound was not prohibited in the DynamicData API. An attempt to set a member past the sequence's maximum bound now returns an error.

[RTI Issue ID CORE-9174]

4.6.19 DynamicData::clear_all_members API did not work on bound DynamicData objects

The **DynamicData::clear_all_members** API did not work on bound DynamicData objects. If you called **DynamicData::clear_all_members** on a bound DynamicData object, unbound that object, and then bound the object again to the same member, all of the previous values were still set, as if the **DynamicData::clear_all_members** API had never been called.

Note: In the Modern C++ API, binding is equivalent to loaning a DynamicData object.

This issue has been resolved.

[RTI Issue ID CORE-9175]

4.7 Fixes Related to Transports

4.7.1 Communication between kernel and RTP Mode Participants with shared memory transport not working on 64-bit VxWorks 6 platforms

Applications could not communicate between kernel and RTP mode using the shared memory transport on 64-bit VxWorks 6 systems. (This is not an issue for 64-bit VxWorks 7 systems.) This problem has been resolved.

As a result of this fix, applications built with older *Connext DDS* releases cannot communicate with applications built with this release if the communication uses shared memory on 64-bit VxWorks 6 systems and occurs between kernel and RTP mode. (This configuration is not a common use case.)

[RTI Issue ID CORE-8171]

4.7.2 Network interface tracker may have reported non-existing changes

In some configurations in which several network interfaces share the same IP address, the network interface tracker may have reported non-existing changes on the interfaces. This problem has been resolved.

[RTI Issue ID CORE-8205]

4.7.3 Possible continuous failure to send over shared memory transport

When using the shared memory transport and rapidly creating and deleting *DomainParticipants*, it was possible for a *DataWriter* or *DataReader* to continuously fail to send content. The following log message was generated with the NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL verbosity and NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION category:

```
RTINetioSender_send: send failed: locator:
shmem://0000:0702:0007:0000:0000:0000:0000:0000:10414
```

(The specific locator numbers may differ.)

This problem has been resolved.

[RTI Issue ID CORE-8295]

4.7.4 Race condition in shared memory transport led to cleanup failure

A race condition in the shared memory transport occurred when two processes attempted to create a *DomainParticipant* with the same domain ID and participant ID at the same time. This race condition led to cleanup problems when one of the *DomainParticipants* was being deleted. It would generate these error messages:

```
NDDS_Transport_Shmem_destroy_recvresource_rrEA:!take mutex
NDDS_Transport_Shmem_destroy_recvresource_rrEA:!give mutex
RTIOsapiSharedMemoryMutex_delete:OS semctl() failure, error 0X16: Invalid argument
```

This race condition has been fixed. These errors no longer appear.

[RTI Issue ID CORE-8534]

4.7.5 UDPv4/UDPv6 transport creation failed when setting send_socket_buffer_size or recv_socket_buffer_size to NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT or NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT

In release 5.3.0, there was an issue provoking UDPv4 and UDPv6 transport creation to fail if any of the **send_socket_buffer_size** or **recv_socket_buffer_size** properties was set to either NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT or NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT.

This problem is now resolved.

[RTI Issue ID CORE-8585]

4.7.6 Unexpected shared memory locator when setting rtps_host_id to a value different than DDS_RTPS_AUTO_ID

In *Connext DDS* 5.3.0, the generated shared memory locator was incorrect when the **rtps_host_id** value was different than DDS_RTPS_AUTO_ID. As a result, 5.3.0 may have failed to communicate over shared memory with previous versions of the product.

Starting with *Connext DDS* 6.0.0, the shared memory locator is generated as follows:

- If the **dds.transport.shmem.builtin.host_id** property is defined, the shared memory locator is derived from this value.
- If the Wire Protocol **rtps_auto_id_kind** is DDS_RTPS_AUTO_ID_FROM_IP and **rtps_host_id** is different than DDS_RTPS_AUTO_ID, the shared memory locator is derived from the **rtps_host_id** value.
- In all other cases, the shared memory locator is derived from the MAC address, IP address, or UUID value depending on the **rtps_auto_id_kind** value.

[RTI Issue ID CORE-8738]

4.7.7 Crash when network interface changed before transport was fully created

A crash may have occurred in IP Mobility scenarios if a network interface changed before the transport creation was completed. This problem has been resolved. Now when the network interface tracker starts, it can properly handle changes in the interfaces.

[RTI Issue ID CORE-8869]

4.7.8 Wrong transport class name when logging 'No interfaces' warning

The logging of a warning due to the absence of valid interfaces in TCP Transport, in TCPv4_LAN or TLSv4_LAN mode, triggered the following two messages:

```
NDDS_Transport_UDP_get_class_name_cEA:!family parameter not valid
NDDS_Transport_IP_selectValidInterfaces:WARNING: No interfaces for transport (null) match
allowed patterns
```

This behavior was wrong: it did not properly show the transport class, and, although the message said WARNING, it actually logged an exception. This problem is now fixed: the message now properly shows the transport class, and it logs a warning instead of an exception.

[RTI Issue ID COREPLG-444]

4.8 Fixes Related to Logging and Distributed Logger

4.8.1 Segmentation fault when simultaneously changing log files and writing to a log file

The functions **NDDS_Config_Logger_set_output_file_set()**, **NDDS_Config_Logger_set_output_file()**, **NDDS_Config_Logger_set_output_file_name()**, and **DDS_DomainParticipantFactory_set_qos()** were not thread-safe. If any of these functions was called to change the logging output file(s) while a separate thread was writing a log message, the writing thread may have crashed with a segmentation fault. This problem has been resolved. These functions are now thread-safe.

[RTI Issue ID CORE-8710]

4.8.2 Unexpected "PRESCstReaderCollator_addInstanceEntry:exceeded max total instances" message at WARNING level

You may have seen the following message when using the NDDS_CONFIG_LOG_VERBOSITY_WARNING verbosity level:

```
PRESCstReaderCollator_addInstanceEntry:exceeded max total instances.
```

This message should not be a warning, since exceeding **max_total_instances** is expected when **reader_qos.resource_limits.max_instances** is finite, **reader_qos.resource_limits.max_total_instances** is AUTO (default), and the *DataReader* receives more than r**eader_qos.resource_limits.max_instances**.

The verbosity level at which this message is printed has been changed to NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL.

[RTI Issue ID CORE-8822]

4.8.3 Segmentation fault when attempting to write to one file of a file set if that file failed to be opened

After an application called the **NDDS_Config_Logger_set_output_file_set()** function, if one of the files failed to be opened (let's call it file N), the first attempt to write to that file failed gracefully (which was correct behavior), but the second attempt to write to that file resulted in a segmentation fault. This problem has been resolved. Now, the second attempt to write will be to file N+1 rather than file N.

[RTI Issue ID CORE-9181]

4.8.4 NDDS_Config_Logger_get_output_device() always returned NULL after DomainParticipant creation

NDDS_Config_Logger_get_output_device() always returned NULL after the first *DomainParticipant* was created. This problem only affected 5.3.x releases and has been fixed. **NDDS_Config_Logger_get_output_device()** will now return the correct device.

[RTI Issue ID CORE-9202]

4.8.5 Unable to configure Distributed Logger profile Using C++ DomainParticipantFactory

C++ *Distributed Logger* was not using the C++ DomainParticipantFactory internally, but using the C factory. This led to issues like these:

- The operations **RTI_DLOptions::setQosLibrary** and **RTI_DLOptions::setQosProfile** could not be used to configure a QoS profile for *RTI Distributed Logger* while using the C++ API. This problem only occurred when the application using *Distributed Logger* loaded its XML configuration using the QoS values **DomainParticipantFactoryQos.profile.url_profile** or **DomainParticipantFactoryQos.profile.string_profile**.
- If *Distributed Logger* used its own DomainParticipant, the threads associated with this DomainParticipant were not created using the ThreadFactory installed on the C++ DomainParticipantFactory with the **API DDS_DomainParticipantFactory::set_thread_factory**.
- Because *Distributed Logger* was using the C DomainParticipantFactory when creating its own DomainParticipant, you may have been forced to finalize this DomainParticipantFactory (using DDS_DomainParticipantFactory_finalize_instance), in addition to the C++ DomainParticipantFactory to avoid a memory leak.

This problem has been resolved.

[RTI Issue ID DISTLOG-155]

4.8.6 Possible segmentation fault when deleting options that were used to create Distributed Logger instance

An application using *Distributed Logger* may have issued a segmentation fault if the options used to create it were deleted before the *Distributed Logger* instance was finalized. This problem has been resolved.

[RTI Issue ID DISTLOG-157]

4.8.7 Error code for setApplicationKind not captured in Distributed Logger C example

The error code was not captured in the *Distributed Logger* C example, when the options were configured with an application kind. This problem has been resolved.

[RTI Issue ID DISTLOG-159]

4.8.8 Logger settings from QoS File not set correctly when using Distributed Logger

If your application created a Distributed Logger instance before loading the QoS profiles, the verbosity settings specified in the QoS file were overwritten with the default settings. This problem has been resolved.

[RTI Issue ID DISTLOG-175]

4.8.9 "Max queue size reached, message will get lost" warning removed from DistributedLogger

Distributed Logger used to print a warning when a message could not be added to the internal messages queue because the queue was full. This warning has been removed. The warning is no longer necessary because there is a new field in **com::rti::dl::LogMessage** called **messageId** that can be used to detect if messages are lost.

[RTI Issue ID DISTLOG-194]

4.9 Fixes Related to XML Configuration

4.9.1 Some XML example files were not compliant with target XSD schema

The following XML example files were not compliant with their target XSD schemas:

- MONITORING_DEMO.xml
- RTIDDSPING_QOS_PROFILES.example.xml
- RTIDDSSPY_QOS_PROFILES.example.xml
- RTI_RECORDING_SERVICE.xml
- USER_RECORDING_SERVICE.xml
- USER_ROUTING_SERVICE.xml

This problem has been resolved; these example files are now compliant.

[RTI Issue ID CORE-8259]

4.9.2 NDDS_QOS_PROFILES.xml not loaded from default location

Connext DDS did not load a file with the name NDDS_QOS_PROFILES.xml from the default location of **resource/xml**. This problem has been resolved. Now, if such a file exists in that location, *Connext DDS* will automatically load the file.

[RTI Issue ID CORE-8279]

4.9.3 is_default_participant_factory_profile="true" was ignored when <participant_factory_qos> only had the base_name attribute specified

If a <participant_factory_qos> didn't have any tags set in XML and specified just a base_name, the metadata information that indicated what fields were set in XML for the parent weren't copied as a part of the single inheritance code path. This led to problems loading the correct value when the **is_default_participant_factory_profile**="true" attribute was used for a <qos_profile> that contained this <participant_factory_qos>. The reason was that other functions relied on the metadata to detect if something other than the default value was being set.

This problem has been resolved in this release, for both single and multiple QoS Profile inheritance via XML. The metadata is now copied during inheritance.

[RTI Issue ID CORE-8364]

4.9.4 Creating a typecode from an XML with a directive tag may have caused a crash

Creating a typecode from an XML file may have caused a segmentation fault if the tag "directive" was used. This problem occurred when calling the function **DDS_TypeCodeFactory_create_tc_from_xml_file**. This problem has been resolved.

[RTI Issue ID CORE-8493]

4.9.5 XML parser error if <domain_library> or <domain_participant_library> split into multiple tags with same name attribute

The *Connext DDS* XML parser reported an error if a <domain_library> or <domain_participant_library> appeared more than once with the same value in its name attribute. This behavior has changed. Now, a domain or participant library can be split into multiple <domain_library> or <domain_participant_library> tags, respectively.

The resulting behavior is equivalent to defining a single library containing all the elements specified in each library, with the same value in the name attribute.

[RTI Issue ID CORE-8294]

4.9.6 Connext DDS XML parser failed to parse a const char with '\0' value

The Connext DDS XML parser failed with the following error when parsing a const char '\0':

```
DDS_ExpressionEvaluator_get_next_token:expression parse error at column 2: invalid scape
character
DDS_ExpressionEvaluator_evaluate:expression parse error at column 2: empty expression
DDS_XMLConst_evaluate_expression:Parse error at line 3: error evaluating const expression
DDS_XMLConst_initialize:error evaluating const expression
DDS_XMLConst_new:!init XML const object
```

```
RTIXMLParser_onStartTag:Parse error at line 3: Error processing tag 'const'
```

Now the parser can parse this character.

[RTI Issue ID CORE-8802]

4.9.7 Value checks were not enforced for writer_qos.writer_resource_limits.instance_replacement field in DataWriterQos

The *RTI Connext DDS* XML parser did not produce an error if the value specified for **writer_qos.writer_resource_limits.instance_replacement** wasn't one of the standard values specified in the documentation. (In the API Reference HTML documentation, search for **DDS_DataWriterResourceLimitsInstanceReplacementKind** for a list of the accepted values.) The parser now produces an error if the value is not one of these values.

[RTI Issue ID CORE-8810]

4.9.8 Last, not first, value was retrieved when getting a QoS by specifying topic_filter

A problem occurred when a Profile had multiple QoSes of the same type, with the **topic_filter** attribute either not specified (NULL) or set to "*". In this case, API calls (such as **DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name()**) retrieved the wrong QoS values when passing those values to the appropriate function calls (such as **DDS_XMLQosProfile_get_datareader_dds_qos_filtered()**) in fallback scenarios only.

The fallback scenarios are the following:

- When a **topic_filter** argument is set to NULL, a lookup API matches against the first QoS type in the Profile with the **topic_filter** attribute set to NULL. When such a QoS does not exist, the fallback scenario is for the API to return the first matching QoS type (in the case of duplicates) with **topic_filter** set to "*".
- Likewise, when a **topic_filter** argument is set to a valid string value, the lookup API matches against the first QoS type in the Profile with the **topic_filter** attribute set to a matching pattern value. If such a QoS does not exist, the fallback scenario is for the API to return the first matching QoS type (in the case of duplicates) with **topic_filter** set to NULL.

In both of these fallback scenarios, the problem was that the API was not returning the first matching QoS type, but the last one.

This problem is now resolved. In both cases, the API now correctly returns the first matching QoS type.

[RTI Issue ID CORE-9024]

4.9.9 Unable to set WriterDataLifecycle::autopurge_disposed_instances_delay for builtin DataWriters via XML

Due to an error in the shipped DTD, it was not possible to set **WriterDataLifecycle::autopurge_disposed_instances_delay** for the builtin *DataWriters* via XML. Attempting to do so would result in errors during participant creation similar to the following:

```
[CREATE Participant] RTIXMLParser_validateOnStartTag:Parse error at line 113: Unexpected tag
'autopurge_disposed_instances_delay'
[CREATE Participant] RTIXMLParser_parseFromFile_ex:Parse error in file 'USER_QOS_PROFILES.xml'
```

This issue has been resolved.

[RTI Issue ID CORE-9082]

4.10 Fixes Related to XML-Based Application Creation

4.10.1 Sequences defined in XML with sequenceMaxLength of -1 were not interpreted as unbounded

If a type defined in XML had its sequences or strings set to **-1** in the attributes **sequenceMaxLength** or **stringMaxLength**, the resulting sequence or string length was interpreted as being of a default size rather than unbounded. This problem has been resolved. The resulting length is now interpreted as unbounded ((2^32)-1 bytes).

[RTI Issue ID CORE-8494]

4.10.2 XML-Based Application Creation in Java didn't work with some TypeCodes

Types larger than 65,535 bytes and types that use extensible type features only available in TypeObject weren't properly serialized when using XML-Based Application Creation in Java. This issue caused *DomainParticipant* creation to produce a BAD_TYPECODE exception for large types, and could have prevented communication between Java applications and applications using a different language binding for types using extensibility features. This issue has been resolved.

[RTI Issue ID CORE-CORE-8957]

4.11 Fixes Related to OMG Specification Compliance

4.11.1 Connext DDS not compliant with RTPS 2.2

Previous releases of *Connext DDS* were not compliant with the OMG RTPS 2.2 specification. In particular, the GUID announced by default was not compliant with the RTPS 2.2 specification (as described in 4.11.2 Default GUID not compliant with RTPS specification on the facing page).

This release is now fully compliant with RTPS 2.2. In addition, the RTPS wire protocol version that *Connext DDS* announces in messages it puts on the wire has been updated to 2.3, since this release is also fully compliant with the RTPS 2.3 wire protocol (described in the RTPS 2.2 and DDS Security 1.1 specifications).

Note that this release also supports some of new RTPS 2.4 wire protocol features. One example is the ability to announce QoS for builtin endpoints (for more information about this, please refer to 4.11.3 Manual by Participant Liveliness stopped working when communicating with old RTI Connext Professional versions or non-RTI DDS implementations below).

[RTI Issue ID CORE-6902]

4.11.2 Default GUID not compliant with RTPS specification

The default value of **DDS_DomainParticipantQos::wire_protocol::rtps_auto_id_kind** was DDS_RTPS_AUTO_ID_FROM_IP. This value caused the **rtps_host_id** to be the IP address by default, which was not compliant with this statement in the RTPS specification:

> *To comply with this specification, implementations of the RTPS protocol shall set the first two bytes of the guidPrefix to match their assigned vendorId.*

This problem has been resolved by changing the default value of **rtps_auto_id_kind** to DDS_RTPS_AUTO_ID_FROM_UUID. The UUID always has the first two bytes equal to the RTI RTPS vendorId. Note that DDS_RTPS_AUTO_ID_FROM_UUID is currently the only value that will result in a specification-compliant GUID.

[RTI Issue ID CORE-8033]

4.11.3 Manual by Participant Liveliness stopped working when communicating with old RTI Connext Professional versions or non-RTI DDS implementations

In release 5.2 and above, Manual by Participant Liveliness may have stopped working when communicating with remote participants in *Connext DDS* Professional 5.1 or below or when communicating with non-RTI remote participants. Note that this issue was only triggered when setting DiscoveryConfig Qos's **participant_message_reader_reliability_kind** to BEST_EFFORT_RELIABILITY (default). In particular, the problem was provoked by a wrong configuration of the Reliability QoS used when communicating with remote liveliness builtin endpoints.

This problem is now resolved by implementing the RTPS 2.3 specification's BuiltinEndpointQos_t propagation mechanism, which ensures the proper QoS is selected when communicating with remote liveliness builtin endpoints. (RTPS 2.3 defines version 2.4 of the wire protocol.)

[RTI Issue ID CORE-8762]

4.11.4 NACK_FRAG message not compliant with RTPS specification

The NACK_FRAG message was not compliant with the format specified in the RTPS specification. This may have led to interoperability issues with other DDS vendors.

This problem has been fixed.

[RTI Issue ID CORE-9193]

4.12 Fixes Related to Vulnerabilities

This release fixes some potential vulnerabilities, including RTI Issue IDs CORE-8581, CORE-8645, CORE-8868, and CORE-9038.

4.13 Fixes Related to Modern C++ API

4.13.1 Compilation failure when NDDS_USER_DLL_EXPORT defined on non-Windows platform

The preprocessor definition NDDS_USER_DLL_EXPORT is only intended for Windows systems and should be ignored on other platforms. But that was not the case in *Connext DDS* 5.3.0, where this situation could have caused a compilation failure in the Modern C++ API. This problem has been resolved.

[RTI Issue ID CORE-8176]

4.13.2 Downcasting a condition into a ReadCondition not supported

In the Modern C++ API, a Reference Type acting as a base class (for example, **dds::core::Entity**) can be downcasted to a child type (for example, **dds::domain::DomainParticipant**) using **dds::-core::polymorphic_cast** (analogous to **std::dynamic_pointer_cast**).

That was the intended behavior with **dds::core::Condition** and **dds::sub::ReadCondition**, as well as **dds::sub::QueryCondition**; however, the following code did not compile:

```
dds::core::Condition c = ...;
auto rc = dds::core::polymorphic_cast<dds::sub::ReadCondition>(c);
auto qc = dds::core::polymorphic_cast<dds::sub::QueryCondition>(rc);
```

This problem has been resolved, and the previous casts now work.

[RTI Issue ID CORE-8347]

4.13.3 ReadCondition handler may not have been dispatched

One-argument condition handlers used to create *ReadConditions* (or *QueryConditions*) were not called by **WaitSet::dispatch()**.

Other types of *Conditions* were not affected, and no-argument handlers worked in all cases.

This problem has been resolved.

[RTI Issue ID CORE-8352]

4.13.4 Compilation error accessing a const LoanedSamples instance

Attempting to access a const LoanedSamples may have caused a compilation error. For example, the following code didn't compile, although it should be legal:

```
void print_samples(const LoanedSamples<Foo>& samples)
{
    for (auto&& s : samples) {
        std::cout << s.data() << std::endl;
    }
}
```

This problem has been resolved. Now it is possible to iterate through a const LoanedSamples using its const_iterator.

[RTI Issue ID CORE-8423]

4.13.5 Missing unregister_thread function

Release 5.2.0 added the function **unregister_thread()**, which allows releasing *Connext DDS* thread-local memory. The Modern C++ API did not provide this function.

This release adds the function **rti::core::unregister_thread()** and the utility type **rti::core::UnregisterThreadOnExit**, which calls **unregister_thread** in its destructor.

[RTI Issue ID CORE-8425]

4.13.6 dds::sub::Sample creation or assignment from a LoanedSample failed when data was invalid

The creation or assignment of a **dds::sub::Sample** from a LoanedSample (the value type of LoanedSamples) threw an exception if the LoanedSample's data was invalid. This was not the expected behavior, since Sample can hold invalid data.

This problem has been resolved, and now this is the behavior:

- The constructor **Sample<T>(const LoanedSample<T>& ls)** will default-construct the sample data when ls.info().valid() is false.
- The assignment operator **operator=(const LoanedSample<T>& ls)** will not modify the sample data (only the sample info) when **ls.info().valid()** is false.

[RTI Issue ID CORE-8446]

4.13.7 Possible symbol collision with Boost compiling the Modern C++ API

An application using both Boost and the Modern C++ API may have failed to compile due to a symbol collision. The problem happened only if the application included (directly or indirectly) the file boost/-detail/iterator.hpp. This problem has been resolved.

[RTI Issue ID CORE-8536]

4.13.8 dds::topic::find could not return AnyTopic

The function **dds::topic::find** can be used to return a typed topic (Topic<Foo>), but was intended to allow returning an AnyTopic. In previous releases, **dds::topic::find<AnyTopic>(...)** didn't compile.

This problem has been resolved.

[RTI Issue ID CORE-8618]

4.13.9 Inconsistency between XML format and QosProvider::type() arguments

The QosProvider allows loading DynamicTypes from an XML document. The function to retrieve them, **type()**, required two arguments: a library name and a type name. But the XML format to define types doesn't require a library name.

This inconsistency has been resolved by adding an overload whose only argument is the type name. See an example in the Modern C++ API Reference HTML documentation, under "Modules > Programming How-To's > DynamicType and DynamicData Use Cases > Create a DynamicType from an XML description."

[RTI Issue ID CORE-8805]

4.13.10 Missing accessors for Liveliness:: assertions_per_lease_duration

In the Modern C++ API, the type **dds::core::policy::Liveliness** did not provide an accessor to the extension field assertions_per_lease_duration.

This problem has been resolved, and a getter and a setter are now provided:

```
dds::core::policy::Liveliness liveliness;
liveliness.extensions().assertions_per_lease_duration(3);
assert(liveliness.extensions().assertions_per_lease_duration() == 3);
```

[RTI Issue ID CORE-8833]

4.13.11 Entity Listeners may have missed some notifications

A race condition during the creation of an *Entity* (*DomainParticipant*, *DataReader*, etc.) with a *Listener* may have caused the *Listener* to miss a notification for a status change that occurred right before the

Entity's constructor finished. This problem has been resolved.

[RTI Issue ID CORE-8905]

4.14 Other Fixes

4.14.1 strict-aliasing warnings when compiling code generated from an IDL with floats or enumerations

When compiling code generated from an IDL with floats or enumerations using the GCC compiler flag **-Wstrict-aliasing**, you would see strict-aliasing warnings that contained the following:

```
warning: dereferencing type-punned pointer will break strict-aliasing rules
note: in expansion of macro 'RTICdrStream_serializeFloat'
```

This problem has been resolved. These compiler warnings no longer appear.

[RTI Issue ID CORE-6778]

4.14.2 Unexpected sample loss notification on non-VOLATILE DataReader

A new *DataReader* that you have just created and enabled, that is configured with non-VOLATILE durability, may have incorrectly reported sample losses upon matching with a non-VOLATILE *DataWriter*.

This issue occurred when the *DataWriter* was configured with **finite autopurge_unregistered_instances_delay** or **autopurge_unregistered_instances_delay** and some of the instances were removed due to these delays before the *DataReader* was matched with the *DataWriter*.

This problem has been fixed.

[RTI Issue ID CORE-7841]

4.14.3 RTPS messages with wrong alignment incorrectly accepted

RTPS messages and submessages with wrong alignment may have been incorrectly accepted by *Connext DDS*.

This problem is now resolved. By default these messages with wrong alignment are now dropped, and the following message is logged:

```
MIGInterpreter_parse:submessage not aligned to 4
```

Note this behavior can be changed by setting the following property on the Participant PropertyQos to true: **dds.participant.use_45d_compatible_alignment_enforcement**.

[RTI Issue ID CORE-8060]

4.14.4 Segmentation fault while looking up vendor-specific topics from a traditional C++ or .NET application

When monitoring libraries were enabled in a traditional C++ or .NET application, the use of **lookup_topicdescription()** or **find_topic()** to get a monitoring topic led to a segmentation fault. This problem has been resolved. These functions now successfully retrieve the monitoring topic.

Similar problems existed for the *Distributed Logger* topics and the *Security Plugins* Builtin Logging topic. These problems have also been resolved.

[RTI Issue ID CORE-8256]

4.14.5 Memory leak when failing to create builtin endpoints

Failure by *Connext DDS* to create certain builtin endpoints resulted in a memory leak. The affected endpoints were the *DataReaders* and *DataWriters* for the publication, subscription, and participant message builtin topics. These leaks have been fixed.

[RTI Issue ID CORE-8367]

4.14.6 Segmentation Fault when Simultaneously Removing Remote Participant and Sending a Message to a Third Participant

A rare segmentation fault may have occurred in the following scenario:

- Three DomainParticipants: P1, P2, P3.
- Delete P1.
- P2 receives an announcement (via ParticipantBuiltinTopicData) from P1 that it has been deleted. P2 starts removing P1 as a remote participant.
- Simultaneously, P2 sends a message to P3. For example, it could be an ACKNACK from P2's DataReader to P3's DataWriter.
- While P2 is sending the message, P2 may have crashed with a bad pointer access.

This problem has been resolved.

[RTI Issue ID CORE-8464]

4.14.7 Could not add property names that were prefixes of existing property names

If you wanted to introduce certain properties in the PropertyQosPolicy that had application-specific meaning, there were certain property names that could not be used because they were the prefix of an existing *Connext DDS* property name.

One example of an unusable property name was "d" because "d" is a prefix of "dds", which is a prefix of many existing property names. If "d" belonged to a *DataWriter*, then the value of "d" would have been misinterpreted as the value of "dds.sample_assignability.accept_unknown_enum_value".

This problem has been resolved. You may now use "d" or any other property name you want.

[RTI Issue ID CORE-8525]

4.14.8 Unbounded memory growth on DataWriter when enable_required_ subscriptions set to true and DataReaders setting role name were created/destroyed continuously

In previous releases there may have been an unbounded memory growth on a *DataWriter* when:

- **writer_qos.availability.enable_required_subscriptions** was set to true.
- You created/deleted matching *DataReaders* where **reader_qos.subscription_name.role_name** was set to a value other than NULL.

This problem has been fixed.

[RTI Issue ID CORE-8650]

4.14.9 Possible crash when applications used two extensible types that differed by one aliased primitive member

The algorithm that verifies the compatibility of two extensible types may have crashed when one of the two types added an aliased primitive member at the end.

For example, an application publishing or subscribing T1 and trying to match with an application using T2 would crash:

```
struct T1 {
    long x;
};

typedef long LongAlias;
struct T2 {
    long x;
    LongAlias y;
};
```

This problem affected types with extensible extensibility (the default). It did not affect types with final or mutable extensibility.

This problem has been resolved.

[RTI Issue ID CORE-8516]

4.14.10 Errors reported when compiling a file including disc_rtps_impl.h with gcc -C

On UNIX-like systems, errors were reported when compiling a file that included disc_rtps_impl.h with the **-C** option. The errors reported by the gcc compiler are now fixed, and the code now compiles without issue.

[RTI Issue ID CORE-8575]

4.14.11 Unlikely segmentation fault when deleting a DataReader and using GROUP ordered access

An unlikely segmentation fault may have occurred when deleting a *DataReader* that was part of a *Subscriber*, when all of the following was true:

- **subscriber_qos.presentation.access_scope** was set to DDS_GROUP_PRESENTATION_QOS or DDS_HIGHEST_OFFERED_PRESENTATION_QOS
- **subscriber_qos.presentation.ordered_access** was set to TRUE
- the *DataReader* matched a *DataWriter* that was part of a Publisher, where publisher_qos.-presentation.access_scope was set to DDS_GROUP_PRESENTATION_QOS

This problem has been resolved.

[RTI Issue ID CORE-8615]

4.14.12 RTIOsapiProcess_getId() returns incorrect PID in VxWorks Kernel Mode

In previous releases, calling **RTIOsapiProcess_getId** in Vxworks Kernel Mode would return different PIDs when called from different threads (tasks). This problem is resolved. **RTIOsapiProcess_getId()** will now always return the same TASK_ID, regardless of the thread that called it.

[RTI Issue ID CORE-8634]

4.14.13 Potential segmentation fault while unregistering a logger output device

The function **NDDS_Config_Logger_set_output_device()** is not thread-safe. For example, when calling this function to unregister a device, it is possible for a different thread to be logging a message using the device. This situation can lead to a segmentation fault in the function **ADVLOGLoggerDeviceMgr_logMessageLNOOP()**. This problem has been fixed. **NDDS_Config_Logger_set_output_device()** is now thread-safe.

[RTI Issue ID CORE-8671]

4.14.14 Incorrect value for type_consistency in SubscriptionBuiltinTopicData in .NET API

When using the .NET API, the value for **type_consistency** within SubscriptionBuiltinTopicData was incorrect. It did not reflect the underlying native value. This problem has been resolved.

[RTI Issue ID CORE-8678]

4.14.15 DomainParticipant creation did not fail when using an unknown flow controller

Using an unknown flow controller to configure one of the built-in *Topics* did not cause the participant creation to fail. For example:

```
<discovery_config>
<publication_writer_publish_mode>  <flow_controller_name>dds.flow_controller.token_
bucket.UNKNOWN</flow_controller_name>
 <kind>ASYNCHRONOUS_PUBLISH_MODE_QOS</kind>
</publication_writer_publish_mode>
</discovery_config>
```

With the above configuration, the **DomainParticipantFactory::create_participant** operation logged the following error, but it still succeeded:

```
DDS_PublishModeQosPolicy_to_presentation_qos_policy: flow controller name 'dds.flow_
controller.token_bucket.UNKNOWN' not found
```

This problem has been resolved. Now the participant creation fails.

[RTI Issue ID CORE-8722]

4.14.16 Removed recursion in include files

Some static code analysis tools detected that there was a recursion in the following header files:

1. **rtixml_extension.h** included **rtixml_object.h**
2. **rtixml_object.h** included **rtixml_object_impl.h**
3. **rtixml_object_impl.h** included **rtixml_extension.h**

Although there are preprocessor guards protecting against multiple inclusion, this recursion has been fixed.

[RTI Issue ID CORE-8739]

4.14.17 max_blocking_time not honored for KEEP_LAST DataWriters

A *DataWriter* might have not honored the ReliabilityQosPolicy's **max_blocking_time** when the HistoryQoSPolicy's **kind** was set to KEEP_LAST and the **max_send_window_size** was less than the

HistoryQoSPolicy's **depth**. This problem has been resolved: now **max_blocking_time** is always honored.

[RTI Issue ID CORE-8753]

4.14.18 Use of -qosProfile argument in DDS Ping did not take topic_filter into consideration

In *RTI DDS Ping* (*rtiddsping*), the **-qosProfile** command-line argument can be used to specify a Profile within an XML QoS file. *rtiddsping* is used for checking reachability between two DDS nodes. It also allows you to specify the topic name for those published samples using the **-topicName** command-line argument.

When using the **-qosProfile** command line argument, if the Profile contained QoSes with topic_filters specified, the expectation was that the utility would use the appropriate QoS (*DataReader* or *DataWriter* for the **-subscriber** or **-publisher** command-line argument respectively) based on the **-topicName** argument or the default name 'PingTopic'. But this wasn't the case in previous releases.

This problem has been resolved in this release. Now *rtiddsping* correctly uses a QoS taking the topic_filter into consideration.

[RTI Issue ID CORE-8837]

4.14.19 Use of -qosProfile argument in DDS Spy did not take topic_filter into consideration

In *RTI DDS Spy* (*rtiddsspy*), the **-qosProfile** command-line argument can be used to specify a Profile within an XML QoS file. Since *rtiddsspy* snoops for any topic names being published on a given domain ID, if the Profile contains QoSes with topic_filters specified, the expectation is that the utility will use the appropriate QoS based on the snooped topic's name. But this wasn't the case, up through release 5.3.1.

This problem has been resolved in this release. Now *rtiddsspy* correctly uses a QoS taking the topic_filter into consideration.

[RTI Issue ID CORE-8838]

4.14.20 Rare race condition may have caused a keep-last DataWriter to time out in write()

The **write()** operation in a reliable *DataWriter* configured with keep-last history QoS shouldn't time out.

A race condition may have caused it to time out if the blocking time was very small.

This problem has been resolved.

[RTI Issue ID CORE-8845]

4.14.21 DDS_PublisherQos_copy() in C API made a shallow copy

DDS_PublisherQos_copy() in the *Connext DDS* C API made a shallow copy for all the DDS_ThreadSettings_t it contained within the asynchronous_publisher members:

- asynchronous_batch_thread
- thread
- topic_query_publication_thread

Because of this shallow copy, if you made a copy of the DDS_PublisherQos from an original value and finalized the copy by calling **finalize()**, accessing the original structure caused a crash, since the DDS_LongSeq corresponding to cpu_list would have been cleaned up due to the nature of shallow copy. The crash occurred because DDS_ThreadSettings_t contained a member of type DDS_LongSeq, which, being a sequence, has just pointers to the actual values on the heap. Since other language wrappers were calling the C API, this problem was present in all languages.

Now, **DDS_PublisherQos_copy()** makes a deep copy by replicating the heap values and not just the pointers, in all languages.

[RTI Issue ID CORE-8855]

4.14.22 Unexpected sample losses with reason DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT when using Group-Ordered access

DataReaders of applications using Group-Ordered Access may have seen unexpected samples losses with lost reason DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT.

This problem has been fixed.

[RTI Issue ID CORE-8871]

4.14.23 Unexpected "!get remote writer queue" warning

In previous releases, when installing a listener on the built-in *Topics*, you may have observed the following warning:

```
PRESPsReader_deleteRemoteWriterQueue:!get remote writer queue
```

The verbosity of the message has been changed to NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL.

[RTI Issue ID CORE-8903]

4.14.24 DataReader may have incorrectly reported on_sample_lost() with a negative total_count

When receiving a piggyback heartbeat around the time of matching with a remote *DataWriter*, a *DataReader* may have incorrectly reported **on_sample_lost()**. Moreover, the total_count of the DDS_SampleLostStatus was a negative number. This problem occurred between a *DataReader* with volatile durability and a *DataWriter* with batching enabled. This problem has been resolved.

[RTI Issue ID CORE-8921]

4.14.25 Unexpected warning printed when a Participant ignored itself

In previous releases, when a Participant ignored itself using the API **DDS_DomainParticipant_ignore_participant**, *Connext DDS* printed the following unexpected warning:

```
PRESParticipant_removeRemoteParticipant:!goto key remoteParticipant
```

This issue has been fixed. This warning is no longer printed.

[RTI Issue ID CORE-8922]

4.14.26 Potential crash when setting new_participant_domain_id in Java monitoring libraries

In release 5.3.0, a crash may have occurred when using Java monitoring libraries with a **new_participant_domain_id** different than the default. This issue is now resolved: the Java monitoring libraries no longer crash when setting **new_participant_domain_id**.

[RTI Issue ID CORE-8940]

4.14.27 TypeCode.print_complete_IDL() failed if a type inherited through an alias

TypeCode.print_complete_IDL() failed by producing an IllegalStateException if the base type was specified to be an alias (a type created through a typedef). This issue has been resolved.

[RTI Issue ID CORE-8986]

4.14.28 Possible error "Too many open files" specifying the discovery peers by a file

Specifying the discovery peers in a file with the well-known name NDDS_DISCOVERY_PEERS may have triggered the error "Too many open files". This error occurred because the file descriptor was not properly closed. This problem has been resolved. Now the file descriptor is closed after it is used.

[RTI Issue ID CORE-9122]

4.14.29 Error receiving batches on a DataWriter from a DataReader with a different endianness

A *DataReader* may not have received samples published by a *DataWriter* where batching is enabled. If the endianness of the platform in which the *DataReader* was running was different than the endianness of the platform in which the *DataWriter* was running, the *DataReader* printed deserialization errors. This problem has been resolved (samples are now received).

[RTI Issue ID CORE-9128]

4.14.30 Performance degradation when DataWriters sent HeartbeatFrag RTPS messages

You may have observed performance degradation in large data scenarios in which a *DataWriter* sends DataFrag RTPS messages.

The problem occurred only when the *DataWriter* publishing the DataFrag messages also sent HeartbeatFrag messages. This problem happened only with third-party vendor *DataWriters* or when using a *Connext DDS Micro* 3.0.0 (or higher) *DataWriter* with *Connext DDS Professional* 5.3.0 or lower. (The problem does not occur when using Micro 3.0.0 with *Connext DDS* 6.0.0.)

This problem has been fixed.

[RTI Issue ID CORE-9211]

4.14.31 Unexpected COMMENDSrReaderService_onSubmessage:!add NACK_FRAG error message

You may have seen this error message on DataReaders receiving RTPS data fragments (DataFrag):

```
COMMENDSrReaderService_onSubmessage:!add NACK_FRAG
```

This error occurred only when the *DataWriter* publishing the DataFrag messages also sent HeartbeatFrag messages. The error occurred only with third-party vendor *DataWriters* or when using a *Connext DDS Micro* 3.0.0 (or higher) *DataWriter* with *Connext DDS Professional* 5.3.0 or lower. (The problem does not occur when using *Micro* 3.0.0 with *Connext DDS* 6.0.0.)

This problem has been fixed.

[RTI Issue ID CORE-9213]

4.14.32 Visual Studio run-times copied to .NET project directory

Every Windows machine needs to use the redistributable libraries installed on that particular system. Previously, some incompatible redistributable libraries were mistakenly copied into the final application binary directory. Those incompatible libraries in the application's binary directory were used instead of the system

redistributable libraries. In this release, those libraries are no longer copied into the application's binary directory. This allows the correct system ones to be used instead.

[RTI Issue ID PLATFORMS-1316]

Chapter 5 Known Issues

5.1 AppAck Messages Cannot be Greater than Underlying Transport Message Size

A *DataReader* with **acknowledgment_kind** (in the ReliabilityQosPolicy) set to DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE or DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE cannot send AppAck messages greater than the underlying transport message size.

If a *DataReader* tries to send an AppAck message greater than the transport message size, *Connext DDS* will print the following error message:

```
COMMENDFacade_sendAppAck:!add APP_ACK to MIG
COMMENDSrReaderService_sendAppAck:!send APP_ACK
PRESPsService_onReaderAppAckSendEvent:!send acknowledgment
```

To recover from the above error, the *DataReader* must acknowledge samples until the size of the AppAck message goes below the transport message size threshold.

Why does an AppAck message increase its size? An AppAck message contains a list of sequence number intervals where each interval represents a set of consecutive sequence numbers that have been already acknowledged. As long as samples are acknowledged in order, the AppAck message will always have a single interval. However, when samples are acknowledged out of order, the number of intervals and the size of the AppAck will increase.

For more information, see Section 6.3.12, Application Acknowledgment, in the *RTI Connext DDS Core Libraries User's Manual*.

[RTI Issue ID CORE-5329]

5.2 Cannot Open USER_QOS_PROFILES.xml in rti_workspace/examples from Visual Studio

When trying to open the **USER_QOS_PROFILES.xml** file from the resource folder of one of the provided examples, you may see the following error:

```
Could not find file : C:\Users\<user>\Documents\rti_workspace\5.3.0\examples\connext_
dds\c\<example>\win32\USER_QOS_PROFILES.xml
```

The problem is that the Visual Studio project is looking for the file in a wrong location (win32 folder).

You can open the file manually from here:

C:\Users\<user>\Documents\rti_workspace\5.3.0\examples\connext_dds\c\<example>\USER_QOS_PROFILES.xml

This issue does not affect the functionality of the example.

[RTI Issue ID CODEGENII-743]

5.3 DataReader Cannot Persist AppAck Messages Greater Than 32767 Bytes

A *DataReader* using durable reader state, whose **acknowledgment_kind** (in the ReliabilityQosPolicy) is set to DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE or DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE, cannot persist an AppAck message greater than 32767 bytes.

To recover from the previous error, the *DataReader* must acknowledge samples until the size of the AppAck message goes below the transport message size threshold.

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

[RTI Issue ID CORE-5360]

5.4 DataReaders with Different Reliability Kinds Under Subscriber with GROUP_PRESENTATION_QOS may Cause Communication Failure

Creating a *Subscriber* with **PresentationQosPolicy.access_scope** GROUP_PRESENTATION_QOS and then creating *DataReaders* with different **ReliabilityQosPolicy.kind** values creates the potential for situations in which those *DataReaders* will not receive any data.

One such situation is when the *DataReaders* are discovered as late-joiners. In this case, samples are never delivered to the *DataReaders*. A workaround for this issue is to set the **AvailabilityQosPolicy.max_data_availabilty_waiting_time** to a finite value for each *DataReader*.

[RTI Issue ID CORE-7284]

5.5 DataWriter's Listener Callback on_application_acknowledgment() not Triggered by Late-Joining DataReaders

The *DataWriter's* listener callback **on_application_acknowledgment()** may not be triggered by late-joining *DataReaders* for a sample after the sample has been application-level acknowledged by all live *DataReaders* (no late-joiners).

If your application requires acknowledgment of message receipt by late-joiners, use the Request/Reply communication pattern with an Acknowledgment type (see the chapter "Introduction to the Request-Reply Communication Pattern," in the *RTI Connext DDS Core Libraries User's Manual*).

[RTI Issue ID CORE-5181]

5.6 Discovery with Connext DDS Micro Fails when Shared Memory Transport Enabled

Given a *Connext DDS* 6.0.0 application with the shared memory transport enabled, a *Connext DDS* Micro 2.4.x application will fail to discover it. This is due to a bug in *Connext DDS* Micro that prevents a received participant discovery message from being correctly processed. This bug will be fixed in a future release of *Connext DDS* Micro. As a workaround, you can disable the shared memory transport in the *Connext DDS* application and use UDPv4 instead.

[RTI Issue ID EDDY-1615]

5.7 Examples and Generated Code for Visual Studio 2017 may not Compile (Error MSB8036)

The examples provided with *Connext DDS* and the code generated for Visual Studio 2017 will not compile out of the box if the Windows SDK version installed is not 10.0.15063.0. If that happens, you will see the compilation error MSB8036. To compile these projects, select an installed version of Windows SDK from the Project menu -> Retarget solution.

Another option is to set the enviroment variable RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to the SDK version number. For example, set RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to 10.0.16299.0. (Note: the environment variable will not work if you have already retargeted the project via the Project menu.)

[RTI Issue ID CODEGENII-800]

5.8 HighThroughput and AutoTuning built-in QoS profiles may cause Communication Failure when Writing Small Samples

If you inherit from either the **BuiltinQosLibExp::Generic.StrictReliable.HighThroughput** or the **BuiltinQosLibExp::Generic.AutoTuning** built-in QoS profiles, your *DataWriters* and *DataReaders* will fail

to communicate if you are writing small samples.

In *Connext DDS* 5.1.0, if you wrote samples that were smaller than 384 bytes, you would run into this problem. In version 5.2.0 onward, you might experience this problem when writing samples that are smaller than 120 bytes.

This communication failure is due to an interaction between the batching QoS settings in the **Generic.HighThroughput** profile and the *DataReader*'s **max_samples** resource limit, set in the **BuiltinQosLibExp::Generic.StrictReliable** profile. The size of the batches that the *DataWriter* writes are limited to 30,720 bytes (see max_data_bytes). This means that if you are writing samples that are smaller than 30,720/**max_samples** bytes, each batch will have more than **max_samples** samples in it. The *DataReader* cannot handle a batch with more than **max_samples** samples and the batch will be dropped.

There are a number of ways to fix this problem, the most straightforward of which is to overwrite the *DataReader's* **max_samples** resource limit. In your own QoS profile, use a higher value that accommodates the number of samples that will be sent in each batch. (Simply divide 30,720 by the size of your samples).

[RTI Issue ID CORE-6411]

5.9 Memory Leak if Foo:initialize() Called Twice

Calling **Foo:initialize()** more than once will cause a memory leak.

[RTI Issue ID CORE-7678]

5.10 Shared Memory Communication Requires Setting dds.transport.shmem.builtin.hostid in Transport Mobility Scenarios

On some platforms, to use the shared memory transport in a transport mobility scenario, you will also need to set the **dds.transport.shmem.builtin.hostid** property in the *DomainParticipant's* Properties QoS policy. Use this property to assign a unique hostid to the transport. In this release, that unique hostid (a 32-bit integer) must be generated by the user and must be the same for all applications running on the same host.

For instance, if you want to two *Connext DDS* applications to communicate using shared memory and one application is started while no NICs are enabled, and after that you enable a NIC and start another *Connext DDS* application, your applications will not communicate by default on certain platforms. To make both applications communicate, you need to set the property **dds.transport.shmem.builtin.hostid** to the same value in both applications.

Affected platforms: AIX, Solaris.

[RTI Issue ID CORE-8040]

5.11 TopicQueries not Supported with DataWriters Configured to Use Batching or Durable Writer History

Getting TopicQuery data from a *DataWriter* configured to use Batching or Durable Writer History is not supported.

[RTI Issue IDs CORE-7405, CORE-7406]

5.12 Uninstalling on AIX Systems

To uninstall *Connext DDS* on an AIX system: if the original installation is on an NFS drive, the uninstaller will hang and fail to completely uninstall the product. As a workaround, you can remove the installation with this command:

```
rm -rf $INSTALL_PATH/rti_connext_dds-6.0.0
```

[RTI Issue ID INSTALL-323]

5.13 Writer-Side Filtering May Cause Missed Deadline

If you are using a ContentFilteredTopic and you set the Deadline QosPolicy, the deadline may be missed due to filtering by a *DataWriter*.

[RTI Issue ID CORE-1634, Bug # 10765]

5.14 Wrong Error Code After Timeout on write() from Asynchronous Publisher

When using an asynchronous publisher, if **write()** times out, it will mistakenly return DDS_RETCODE_ERROR instead of the correct code, DDS_RETCODE_TIMEOUT.

[RTI Issue ID CORE-2016, Bug # 11362]

5.15 Instance does not Transition to ALIVE when "live" DataWriter Detected

The "Data Distribution Service for Real-time Systems" specification allows transitioning an instance from the NO_WRITERS state to the ALIVE state when a "live" *DataWriter* writing the instance is detected. Currently, this state transition is not supported in *Connext DDS*. The only way to transition an instance from NO_WRITERS to ALIVE state is by receiving a sample for the instance from one of the *DataWriters* publishing it.

Example:

1. A *DataWriter* writes a particular instance. The *DataReader* receives the sample. The *DataWriter* loses liveliness with the *DataReader*, making the instance transition from ALIVE to NO_WRITERS. The writer later becomes alive again, but it doesn't resume writing samples of the instance. In this case, the instance will stay in a NO_WRITERS state.
2. The *DataWriter* publishes a new sample for the instance. Only then does the instance state change on the *DataReader* from NO_WRITERS to ALIVE.

[RTI Issue ID CORE-3018]

5.16 Communication may not be Reestablished in Some IP Mobility Scenarios

If you have two *Connext DDS* applications in different nodes and they change their IP address at the same time, they may not reestablish communication. This situation may happen in the following scenario:

- The applications see each other only from one single network.
- The IP address change happens at the same time in the network interface cards (NICs) that are in the network that is in common for both applications.
- The IP address change on one of the nodes happens before the arrival of the DDS discovery message propagating the address change from the other side.

[RTI Issue ID CORE-8260]

5.17 DomainParticipantFactoryQos in XML may not be Loaded

The DomainParticipantFactoryQos set in XML may not be loaded in the Modern C++ API. This happens when a QosProvider is accessed before any DDS entities have been created. The most straightforward workaround for this issue is to set the DomainParticipantFactoryQos in code as follows:

```
dds::core::QosProvider qos_provider = dds::core::QosProvider("USER_QOS_PROFILES.xml",
"MyQosLibrary::MyQosProfile");

dds::domain::qos::DomainParticipantFactoryQos factory_qos;
factory_qos->entity_factory.autoenable_created_entities(false);

// Set the DomainParticipantFactoryQos
dds::domain::DomainParticipant::participant_factory_qos(factory_qos);

// Continue with application code
dds::domain::DomainParticipant participant(domain_id, qos_provider.participant_qos());
```

[RTI Issue ID CORE-6846]

5.18 Known Issues with Dynamic Data

- The conversion of data by member-access primitives (**get_X()** operations) is limited when converting to types that are not supported on all platforms. This limitation applies when converting to a 64-bit long long type (**get_longlong()** and **get_ulonglong()** operations) and a 128-bit long double type (**get_longdouble()**). These methods will always work for data members that are actually of the correct type, but will only support conversion from values that are stored as smaller types on a subset of platforms. Conversion to 64-bit long longs from a 32-bit or smaller integer type is supported on all Windows, Solaris, and Linux architectures, and any additional 64-bit architectures. Conversion to 128-bit long doubles from a float or double is only supported on Solaris SPARC architectures.

  [RTI Issue ID CORE-2986]

- Types that contain bit fields are not supported by DynamicData. Therefore, when rtiddsspy discovers any type that contains a bit field, *rtiddsspy* will print this message:

  ```
  DDS_DynamicDataTypeSupport_initialize:type not supported (bitfield member)
  ```

  [RTI Issue ID CORE-3949]

5.19 Known Issues in RTI Monitoring Library

5.19.1 Problems with NDDS_Transport_Support_set_builtin_transport_property() if Participant Sends Monitoring Data

If a *Connext DDS* application uses the **NDDS_Transport_Support_set_builtin_transport_property()** API (instead of the PropertyQosPolicy) to set built-in transport properties, it will not work with *Monitoring Library* if the user participant is used for sending all the monitoring data (the default settings). As a workaround, you can configure Monitoring Library to use another participant to publish monitoring data (using the property name **rti.monitor.config.new_participant_domain_id** in the PropertyQosPolicy).

[RTI Issue ID MONITOR-222]

5.19.2 Participant's CPU and Memory Statistics are Per Application

The CPU and memory usage statistics published in the *DomainParticipant* entity statistics topic are per application instead of per *DomainParticipant*.

[RTI Issue ID CORE-7972]

5.19.3 XML-Based Entity Creation Nominally Incompatible with Static Monitoring Library

If setting the *DomainParticipant* QoS programmatically in the application is not possible (i.e., when using XML-based Application Creation), the monitoring **create** function pointer may still be provided via an XML profile by using the environment variable expansion functionality. The monitoring property within the *DomainParticipant* QoS profile in XML must be set as follows:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>rti.monitor.library</name>
                <value>timonitoring</value>
            </element>
            <element>
                <name>rti.monitor.create_function_ptr</name>
                <value>$(MONITORFUNC)</value>
            </element>
        </value>
    </property>
</participant_qos>
```

Then in the application, before retrieving the DomainParticipantFactory, the environment variable must be set programmatically as follows:

```
...
sprintf(varString, "MONITORFUNC=%p", RTIDefaultMonitor_create);
int retVal = putenv(varString);
...
//DomainParticipantFactory must be created after env. variable setting
```

[RTI Issue ID CORE-5540]

5.19.4 ResourceLimit channel_seq_max_length must not be Changed

The default value of DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length can't be modified if a DomainParticipant is being monitored. If this QoS value is modified from its default value of 32, the monitoring library will fail.

[RTI Issue ID MONITOR-220]

Chapter 6 Experimental Features

This software may contain experimental features. These are used to evaluate potential new features and obtain customer feedback. They are not guaranteed to be consistent or supported and they should not be used in production.

In the API Reference HTML documentation, experimental APIs are marked with **<<experimental>>**.

The APIs for experimental features use the suffix **_exp** to distinguish them from other APIs. For example:

```
const DDS::TypeCode * DDS_DomainParticipant::get_typecode_exp(
    const char * type_name);
```

Experimental features are also clearly noted as such in the *User's Manual* or *Getting Started Guide* for the component in which they are included.

Disclaimers:

- Experimental feature APIs may be only available in a subset of the supported languages and for a subset of the supported platforms.
- The names of experimental feature APIs will change if they become officially supported. At the very least, the suffix, **_exp**, will be removed.
- Experimental features may or may not appear in future product releases.
- Experimental features should not be used in production.

Please submit your comments and suggestions about experimental features to **support@rti.com** or via the RTI Customer Portal (https://support.rti.com/).