RTI Connext DDS

Core Libraries

Release Notes

Version 5.2.3

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/

Chapter 1 Introduction

Connext DDS 5.2.3 is a maintenance release based on feature release 5.2.0. This document includes the following:

- System Requirements (Chapter 2 on page 3)
- Transitioning from 5.1 to 5.2 (Chapter 3 on page 8)
- Compatibility (Chapter 4 on page 9)
- What's Fixed in 5.2.3 (Chapter 5 on page 27)
- Known Issues (Chapter 6 on page 46)
- Experimental Features (Chapter 7 on page 53)

For an overview of new features and improvements in 5.2.3, see the separate ***What's New*** document for 5.2.3.

This document discusses fixes included in 5.2.3. For what's new and fixed in 5.2.0, see the *What's New* and *Release Notes* documents provided with 5.2.0, respectively.

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, access documentation 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** (http://community.rti.com) provides a wealth of knowledge to help you use RTI® Connext™ DDS, including:

  - 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.

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 all the platforms in Table 2.1 Supported Platforms. If you want to use a platform that is not on RTI's download portal, please contact RTI Support.

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

Table 2.1 Supported Platforms

Operating System	Version
AIX®	AIX 5.3, 7.1
Android™	Android 2.3 - 4.4, 5.0, 5.1
INTEGRITY® (target only)	INTEGRITY 5.0.11, 10.0.2, and 11.0.4
iOS®	iOS 8.2
Linux® (ARM® CPU)	Raspbian Wheezy 7.0

Table 2.1 Supported Platforms

Operating System	Version
Linux (Intel® CPU)	CentOS 5.4, 5.5, 6.0, 6.2 - 6.4, 7.0 Red Hat® Enterprise Linux 4.0, 5.0-5.2, 5.4, 5.5, 6.0 - 6.5, 6.7, 7.0 SUSE® Linux Enterprise Server 11 SP2, SP3 Ubuntu® Server 12.04 LTS, Ubuntu 14.04 LTS Wind River® Linux 4
Linux (PowerPC® CPU)	Freescale P2020RDB Wind River Linux 3 Yellow Dog™ Linux 4.0
LynxOS® (target only)	LynxOS 4.0, 4.2, 5.0
Mac OS®	OS X 10.8, 10.10, 10.11
QNX® (target only)	QNX Neutrino® 6.4.1, 6.5
Solaris™	Solaris 2.9, 2.10
VxWorks® (target only)	VxWorks 5.5, 6.3 - 6.9, 6.9.3.2, 6.9.4, 7.0 VxWorks 653 2.3
Windows®	Windows 7, 8, 8.1, 10 Windows Server 2003, 2008 R2, 2012 R2 Windows Vista® Windows XP Professional SP2

Table 2.2 Custom Supported Platforms lists additional target libraries available for Connext DDS, for which RTI offers custom support. If you are interested in using one of these platforms, please contact your local RTI representative or email **sales@rti.com**.

Other platforms not listed in this document may be supported through special development and maintenance agreements. Contact your RTI sales representative for details.

Table 2.2 Custom Supported Platforms

Operating System	Version
INTEGRITY	INTEGRITY 5.0.11 on PPC 8349 CPU

Table 2.2 Custom Supported Platforms

Operating System	Version
Linux	Debian Linux 3.12 on ARMv7a Cortex-A9 CPU
	Linux 3.8.13 on QorIQ or P4040/P4080/P4081 CPU
	NI Linux Real-Time 3.2[a] on ARMv7 CPU
	Red Hat Enterprise Linux 5.2 on x86 CPU with gcc 4.2.1
	RedHawk Linux 6.0 on x64 CPU
QNX	QNX 6.6

2.2 Requirements when Using Microsoft Visual Studio

When Using Visual Studio 2008 — Service Pack 1 Requirement

You must have Visual Studio 2008 Service Pack 1 or the Microsoft Visual C++ 2008 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. The Microsoft Visual C++ 2008 SP1 Redistributable Package can be downloaded from the following Microsoft websites:

For x86 architectures:

http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en

For x64 architectures:

http://www.microsoft.com/downloads/details.aspx?FamilyID=ba9257ca-337f-4b40-8c14-157cf-dffee4e&displaylang=en

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:

[a]Requires a hardware FPU in the processor and are compatible with systems that have soft-float libc.

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 — Update 4 Redistributable Package Requirement

You must have Visual C++ Redistributable for Visual Studio 2013 Update 4 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 Update 4 from this Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=40784

When Using Visual Studio 2015 — Update 1 Redistributable Package Requirement

You must have Visual C++ Redistributable for Visual Studio 2015 Update 1 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 2015 Update 1 from this Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=49984

2.3 Disk and Memory Usage

Disk usage for a typical host-only installation is approximately 385 MB on Linux systems and 625 MB on Windows systems. Each additional architecture (host or target) requires an additional 162 MB on Linux systems and 402 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.

2.4 Networking Support

Connext DDS includes full support for pluggable transports. Connext DDS applications can run over various communication media, such as UDP/IP over Ethernet, and local inter-process shared memory—provided the correct transport plug-ins for the media are installed.

By default, the Connext DDS core uses built-in UDP/IPv4 and shared-memory[a] transport plug-ins.

A built-in IPv6 transport is available (disabled by default) for some platforms.

A TCP transport is also available (but is not a built-in transport) for some platforms.

See the *RTI Connext DDS Core Libraries Platform Notes* for details on which platforms support the IPv6 and TCP transports.

[a]The shared-memory transport is not supported on VxWorks 5.5 platforms.

Chapter 3 Transitioning from 5.1 to 5.2

Please read this before installing. If you are transitioning from 5.1 to 5.2, there are a few important differences to be aware of.

- New Installation Procedure

  This release is packaged in a different structure than previous releases. There are still host and target bundles. However, the host bundle is a .run file and the target is a .rtipkg file.

  To install these bundles, you will run the host bundle (such as rti_connext_dds-5.2.0-core-host-x64Linux.run). The installer will walk you through installation. After installing the host, you will install your target(s). To do so, you can use the RTI Package Installer utility that's available in RTI Launcher, or you can run the **rtipkginstall** script from **<install directory>/bin**. For example, to install the target bundle, you would enter this command:

  ```
  bin/rtipkginstall <target-bundle.rtipkg>
  ```

  The *Getting Started Guide* has more details on installing.

- New Directory Structure

  The second difference you will see is that the directory structure is different. This means you should not install in the same place as your current RTI release. After you install both the host and target, you can find the libraries in **rti_connext_dds-5.2.0/lib/<target architecture>** are the header files in **rti_connext_dds-5.2.0/include/ndds**.

- New Location for Scripts

  The scripts for all tools, services, and utilities are now in the **/bin** directory (these were in /scripts in the previous release).

Chapter 4 Compatibility

4.1 Wire Protocol Compatibility

4.1.1 General Information on RTPS (All Releases)

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 2.1. RTI plans to maintain interoperability between middleware versions based on RTPS 2.x.

4.1.2 Release-Specific Information for Connext DDS 5.x

4.1.2.1 Large Data with Endpoint Discovery

An endpoint (*DataWriter* or *DataReader*) created with Connext DDS 5.x will not be discovered by an application that uses a previous release (4.5f or lower) if any of these conditions are met:

The endpoint's TypeObject is sent on the wire and its size is greater than 65535 bytes. For information on TypeObjects, see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

The endpoint's UserDataQosPolicy value is greater than 65535 bytes.

TypeObjects and UserDataQosPolicy values with a serialized size greater than 65535 bytes require extended parameterized encapsulation when they are sent as part of the endpoint discovery information. This parameterized encapsulation is not understood by previous Connext DDS releases.

4.1.3 Release-Specific Information for Connext DDS 4.5 and 5.x

Connext DDS 4.5 and 5.x are compatible with RTI Data Distribution Service 4.2 - 4.5, except as noted below.

4.1.3.1 RTPS Versions

Connext DDS 4.5 and 5.x support RTPS 2.1. Some earlier releases (see Table 4.1 RTPS Versions) supported RTPS 2.0 or 1.2. Because these RTPS versions are incompatible with each other, applications built with Connext DDS/RTI Data Distribution Service 4.2e and higher will not interoperate with applications built with RTI Data Distribution Service 4.2c or lower.

Table 4.1 RTPS Versions

	RTPS Version
Connext DDS 4.5f and higher	2.1
Data Distribution Service 4.2e - 4.5e	2.1
Data Distribution Service 4.2c	2.0
Data Distribution Service 4.2b and lower	1.2

4.1.3.2 double, long long, unsigned long long or long double Wire Compatibility

If your Connext DDS application's data type uses a 'double,' 'long long,' 'unsigned long long,' or 'long double,' it will not interoperate with applications built with RTI Data Distribution Service 4.2e or lower, unless you use the **-use42eAlignment** flag when generating code with *rtiddsgen*.

Starting with Connext DDS 5.2.3, you must also set the following *DataWriter* and *DataReader* QoS properties to "1":

DataWriter QoS property: **dds.data_writer.type_support.use_42e_alignment**

DataReader QoS property: **dds.data_writer.type_support.use_42e_alignment**

4.1.3.3 Sending 'Large Data' between *RTI Data Distribution Service* 4.4d and Older Releases

The 'large data' format in RTI Data Distribution Service 4.2e, 4.3, 4.4b and 4.4c is not compliant with RTPS 2.1. ('Large data' refers to data that cannot be sent as a single packet by the transport.)

This issue is resolved in Connext DDS and in RTI Data Distribution Service 4.4d-4.5e. As a result, by default, large data in Connext DDS and in RTI Data Distribution Service 4.4d-4.5e is not compatible with older versions of RTI Data Distribution Service. You can achieve backward compatibility by setting the following properties to 1.

```
dds.data_writer.protocol.use_43_large_data_format
dds.data_reader.protocol.use_43_large_data_format
```

The properties can be set per *DataWriter/DataReader* or per *DomainParticipant*.

For example:

```
<participant_qos>
    <property>
        <value>
           <element>
               <name>
                   dds.data_writer.protocol.use_43_large_data_format
               </name>
               <value>1</value>
           </element>
           <element>
               <name>
                  dds.data_reader.protocol.use_43_large_data_format
               </name>
               <value>1</value>
           </element>
       </value>
   </participant_qos>
```

4.2 Code Compatibility

4.2.1 General Information (All Releases)

The Connext DDS core uses an API that is an extension of the OMG Data Distribution Service (DDS) standard API, version 1.2. 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 this document.

RTI allows you to define the data types that will be used to send and receive messages. To create code for a data type, Connext DDS includes RTI Code Generator (also known as *rtiddsgen*). For input, *RTI Code Generator* takes a data-type description (in IDL, XML, XSD, or WSDL format); RTI Code Generator generates header files (or a class in Java) that can be used to send and receive data of the defined type. It also generates code that takes care of low-level details such as transforming the data into a machine-independent representation suitable for communication.

While this is not the common case, some upgrades require you to regenerate the code produced by *RTI Code Generator*. The regeneration process is very simple; you only need to run the new version of *RTI*

Code Generator using the original input IDL file. This process will regenerate the header and source files, which can then be compiled along with the rest of your application.

4.2.2 Release-Specific Information for Connext DDS 5.x

This section points out important differences in Connext DDS 5.x compared to 4.5f that may require changes on your part when upgrading from 4.5f (or lower) to 5.x

4.2.2.1 Required Change for Building with C++ Libraries for QNX Platforms—New in Connext DDS 5.0.0

For QNX architectures, in release 5.x: The C++ libraries are now built without **-fno-rtti** and with **-fexceptions**. To build QNX architectures with release 5.x, you must build your C++ applications without **-fno-exceptions** in order to link with the RTI libraries. In summary:

Do not use **-fno-exceptions** when building a C++ application or the build will fail. It is not necessary to use **-fexceptions**, but doing so will not cause a problem.

It is no longer necessary to use **-fno-rtti**, but doing so will not cause a problem.

4.2.2.2 Changes to Custom Content Filters API

Starting with Connext DDS 5.0.0, the ContentFilter's **evaluate()** function now receives a new '**struct DDS_FilterSampleInfo ***' parameter that allows it to filter on meta-data.

The **evaluate()** function of previous custom filter implementations must be updated to add this new parameter.

4.2.2.3 Changes to FooDataWriter::get_key_value()

Starting with Connext DDS 5.2.0, the return value of the function **FooDataWriter::get_key_value()** has changed from DDS_RETCODE_ERROR to DDS_RETCODE_BAD_PARAMETER if the instance handle passed to the function is not registered.

This change in behavior was done to align with the DDS specification (RTI Issue ID CORE-6096).

4.2.2.4 Changes in Generated Type Support Code in Connext DDS 5.0.0

The *rtiddsgen*-generated type-support code for user-defined data type changed in 5.0.0 to facilitate some new features. **If you have code that was generated with *rtiddsgen* 4.5 or lower, you must regenerate that code** using the version of rtiddsgen provided with this release.

4.2.2.5 Changes in Generated Type Support Code in Connext DDS 5.1.0

The *rtiddsgen*-generated type-support code for user-defined data type changed in 5.1.0 to facilitate some new features. **If you have code that was generated with rtiddsgen 5.0.0 or lower, you must regenerate that code** using the version of rtiddsgen provided with this release.

4.2.2.6 New Default Value for DomainParticipant Resource Limit, participant_property_string_max_length

Starting with Connext DDS 5.1.0, the default value of **participant_property_string_max_length** in the DomainParticipantResourceLimitsQosPolicy has been changed from 1024 characters to 2048 to accommodate new system properties (see Section 8.7, System Properties, in the *RTI Connext DDS Core Libraries User's Manual*).

4.2.2.7 New Default Value for DomainParticipant's participant_name.name

Starting with Connext DDS 5.1.0, the default value for **participant_qos.participant_name.name** has been changed from the string "[ENTITY]" to NULL to provide consistency with the default name of other entities such as DataWriters and DataReaders.

4.2.2.8 Constant DDS_AUTO_NAME_ENTITY no Longer Available

Starting with Connext DDS 5.1.0, the constant DDS_AUTO_NAME_ENTITY, which was used to assign the name "[ENTITY]" to a participant, has been removed. References to DDS_AUTO_NAME_ENTITY must be removed from Connext DDS applications.

4.2.2.9 Changes to Time_t and Duration_t Methods

Starting with Connext DDS 5.2.0, the signatures for some of the Time_t and Duration_t methods have changed:

Traditional C++:

Old: **from_micros(DDS_UnsignedLong microseconds);**
New: **from_micros(DDS_UnsignedLongLong microseconds);**

Old: **from_millis(DDS_UnsignedLong milliseconds);**
New: **from_millis(DDS_UnsignedLongLong milliseconds);**

Old: **from_nanos(DDS_UnsignedLong nanoseconds);**
New: **from_nanos(DDS_UnsignedLongLong nanoseconds);**

Old: **from_seconds(DDS_Long seconds);**
New: **from_seconds(DDS_UnsignedLong seconds);**

Java:

Old: **from_seconds(long seconds)**
New: **from_seconds(int seconds)**

.NET:

Old: **from_micros(long microseconds)**
New: **from_micros(System::UInt64 microseconds)**

Old: **from_millis(System::UInt32 milliseconds)**
New: **from_millis(System::UInt64 milliseconds)**

Old: **from_nanos(long nanoseconds)**
New: **from_nanos(System::UInt64 nanoseconds)**

4.2.3 Release-Specific Information for RTI Data Distribution Service 4.x, Connext DDS 4.5 and 5.x

4.2.3.1 Type Support and Generated Code Compatibility

long long Native Data Type Support:

In Connext DDS (and RTI Data Distribution Service 4.5c,d,e), we assume all platforms natively support the 'long long' data type. This was not the case in older versions of RTI Data Distribution Service. There is no longer a need to define RTI_CDR_SIZEOF_LONG_LONG to be 8 on some platforms in order to map the DDS 'long long' data type to a native 'long long' type.

double, long long and unsigned long long Code Generation:

If your Connext DDS (or RTI Data Distribution Service 4.3-4.5e) application's data type uses a 'double,' 'long long,' 'unsigned long long,' or 'long double,' it will not be backwards compatible with applications built with RTI Data Distribution Service 4.2e or lower, unless you use the **-use42eAlignment** flag when generating code with *rtiddsgen*.

Changes in Generated Type Support Code:

The *rtiddsgen*-generated type-support code for user-defined data type changed in 4.5 to facilitate some new features. **If you have code that was generated using *rtiddsgen* 4.4 or lower, you must regenerate that code** using the version of *rtiddsgen* provided with this release.

Cross-Language Instance Lookup when Using Keyed Data Types:

This issue only impacts systems using RTI Data Distribution Service 4.3.

In RTI Data Distribution Service 4.3, keys were serialized with the incorrect byte order when using the Java and .NET[a] APIs for the user-defined data type, resulting in incorrect behavior in the **lookup_**

[a]The Connext DDS .NET language binding is currently supported for C# and C++/CLI.

instance() and **get_key()** methods when using keyed data-types to communicate between applications in these languages and other programming languages. This issue was resolved in Java starting in RTI Data Distribution Service 4.3e rev. 01 and starting in .NET in *RTI Data Distribution Service* 4.4b.

As a result of this change, systems using keyed data that incorporate Java or .NET applications using both *RTI Data Distribution Service* 4.3 and this Connext DDS release could experience problems in the **lookup_instance()** and **get_key()** methods. If you are affected by this limitation, please contact RTI Support.

Data Types with Variable-Size Keys:

If your data type contains more than one key field and at least one of the key fields except the last one is of variable size (for example, if you use a string followed by a long as the key):

RTI Data Distribution Service 4.3e, 4.4b or 4.4c *DataWriters* may not be compatible with RTI Data Distribution Service 4.4d or higher *DataReaders*.

RTI Data Distribution Service 4.3e, 4.4b or 4.4c *DataReaders* may not be compatible with RTI Data Distribution Service 4.4d or higher *DataWriters*.

Specifically, all samples will be received in those cases, but you may experience the following problems:

Samples with the same key may be identified as different instances. (For the case in which the *DataWriter* uses RTI Data Distribution Service 4.4d-4.5e or Connext DDS, this can only occur if the *DataWriter's* **disable_inline_keyhash** field (in the DataWriterProtocolQosPolicy) is true (this is not the default case).

Calling **lookup_instance()** on the *DataReader* may return HANDLE_NIL even if the instance exists.

Please note that you probably would have had the same problem with this kind of data type already, even if both your *DataWriter* and *DataReader* were built with RTI Data Distribution Service 4.3e, 4.4b or 4.4c.

If you are using a C/C++ or Java IDL type that belongs to this data type category in your RTI Data Distribution Service 4.3e, 4.4b or 4.4c application, you can resolve the backwards compatibility problem by regenerating the code with version of rtiddsgen distributed with RTI Data Distribution Service 4.4d. You can also upgrade your whole system to this release.

4.2.3.2 Other API and Behavior Changes

Code Compatibility Issue in C++ Applications using Dynamic Data:

If you are upgrading from a release prior to 4.5f and use Dynamic Data in a C++ application, you may need to make a minor code change to avoid a compilation error.

The error would be similar to:

```
MyFile.cpp:1060: error: could not convert '{0u, 4294967295u, 4294967295u, 0u}' to
'DDS_DynamicDataTypeSerializationProperty_t
MyFile.cpp:1060: warning: extended initializer lists only available with -std=c++0x or -
std=gnu++0
```

```
MyFile.cpp:1060: warning: extended initializer lists only available with -std=c++0x or -
std=gnu++0x
MyFile.cpp:1060: error: could not convert '{0l, 65536l, 1024l}' to 'DDS_DynamicDataProperty_
t'
MyFile.cpp:1060: error: could not convert '{0u, 4294967295u, 4294967295u, 0u}' to
'DDS_DynamicDataTypeSerializationProperty_t
```

The code change involves using a constructor instead of a static initializer. Therefore if you have code like this:

```
DDS_DynamicDataTypeProperty_t properties =
    DDS_DynamicDataTypeProperty_t_INITIALIZER;
...
typeSupport = new DDSDynamicDataTypeSupport(typeCode, properties);
```

Replace the above with this:

```
DDS_DynamicDataTypeProperty_t properties;
...
typeSupport = new DDSDynamicDataTypeSupport(typeCode, properties);
```

New on_instance_replaced() method on DataWriterListener:

Starting with RTI Data Distribution Service 4.5c (and thereby included in Connext DDS), there is a new DataWriterListener method, **on_instance_replaced()**, which supports the new instance replacement feature. This method provides notification that the maximum instances have been used and need to be replaced. If you are using a DataWriterListener from an older release, you may need to add this new method to your listener.

Counts in Cache Status and Protocol Status changed from Long to Long Long:

Starting with RTI Data Distribution Service 4.5c (and thereby included in Connext DDS), all the 'count' data types in DataReaderCacheStatus, DataReaderProtocolStatus, DataWriterCacheStatus and DataWriterProtocolStatus changed from 'long' to 'long long' in the C, C++ and .NETAPIs in order to report the correct value for Connext DDS applications that run for very long periods of time. If you have an application written with a previous release of RTI Data Distribution Service that is accessing those fields, data-type changes may be necessary.

Changes in RtpsReliableWriterProtocol_t:

Starting with RTI Data Distribution Service 4.4c (and thereby included in Connext DDS), two fields in DDS_RtpsReliableWriterProtocol_t have been renamed:

Old name: disable_positive_acks_decrease_sample_keep_duration_**scaler**
New name:disable_positive_acks_decrease_sample_keep_duration_**factor**

Old name: disable_positive_acks_increase_sample_keep_duration_**scaler**
New name: disable_positive_acks_increase_sample_keep_duration_**factor**

In releases prior to 4.4c, the NACK-only feature was not supported on platforms without floating-point support. Older versions of RTI Data Distribution Service will not run on these platforms because floats and doubles are used in the implementation of the NACK-only feature. In releases 4.4c and above, the NACK-only feature uses fixed-point arithmetic and the new DDS_Long "factor" fields noted above, which replace the DDS_Double "scaler" fields.

Tolerance for Destination-Ordering by Source-Timestamp:

Starting with RTI Data Distribution Service 4.4b (and thereby included in Connext DDS), by default, the middleware is less restrictive (compared to older releases) on the writer side with regards to timestamps between consecutive samples: if the timestamp of the current sample is less than the timestamp of the previous sample by a small tolerance amount, **write()** will succeed.

If you are upgrading from RTI Data Distribution Service 4.4a or lower, and the application you are upgrading relied on the middleware to reject timestamps that ‘went backwards’ on the writer side (that is, when a sample’s timestamp was earlier than the previous sample’s), there are two ways to keep the previous, more restrictive behavior:

- If your DestinationOrderQosPolicy’s **kind** is BY_SOURCE_TIMESTAMP: set the new field in the DestinationOrderQosPolicy, **source_timestamp_tolerance**, to 0.
- If your DestinationOrderQosPolicy's **kind** is BY_RECEPTION_TIMESTAMP on the writer side, consider changing it to BY_SOURCE_TIMESTAMP instead and setting **source_timestamp_tolerance** to 0. However, this may not be desirable if you had a particular reason for using BY_RECEPTION_TIMESTAMP (perhaps because you did not want to match readers with BY_SOURCE_TIMESTAMP). If you need to keep the BY_RECEPTION_TIMESTAMP setting, there is no QoS setting that will give you the exact same behavior on the writer side as the previous release.

Starting with RTI Data Distribution Service 4.4b (and thereby included in Connext DDS), by default, the middleware is more restrictive (compared to older releases) on the reader side with regards to source and reception timestamps of a sample if DestinationOrderQosPolicy **kind** is set to BY_SOURCE_TIMESTAMP: if the reception timestamp of the sample is less than the source timestamp by more than the tolerance amount, the sample will be rejected.

If you are upgrading from RTI Data Distribution Service 4.4a or lower, your reader is using BY_SOURCE_TIMESTAMP, and you need the previous less restrictive behavior, set **source_timestamp_tolerance** to infinite on the reader side.

New Location and Name for Default XML QoS Profiles File (formerly NDDS_QOS_PROFILES.xml):

Starting with RTI Data Distribution Service 4.4d (and thereby included in Connext DDS) the default XML QoS Profiles file has been renamed and is installed in a new directory:

- Old location/name: **$NDDSHOME/resource/xml/NDDS_QOS_PROFILES.xml**
- New location/name: **$NDDSHOME/resource/xml/NDDS_QOS_PROFILES.example.xml**

If you want to use this QoS profile, you need to set up your **NDDSHOME** environment variable at run time and rename the file **NDDS_QOS_PROFILES.example.xml** to **NDDS_QOS_PROFILES.xml** (i.e., by default, even if your **NDDSHOME** environment variable is set, this QoS profile is not used.) See Section 17.2, How to Load XML-Specified QoS Settings, in the *RTI Connext DDS Core Libraries User's Manual* for details.

Changes in the default value for max_objects_per_thread:

Starting with RTI Data Distribution Service 4.4d (and thereby included in Connext DDS), the default value for the **max_objects_per_thread** field in the SystemResourceLimitsQosPolicy has been changed from 512 to 1024.

Type Change in Constructor for SampleInfoSeq—.NET Only:

Starting with RTI Data Distribution Service 4.5c (and thereby included in Connext DDS), the constructor for SampleInfoSeq has been changed from SampleInfoSeq(UInt32 maxSamples) to SampleInfoSeq(Int32 maxSamples). This was to make it consistent with other sequences.

Default Send Window Sizes Changed to Infinite:

Starting with RTI Data Distribution Service 4.5d (and thereby included in Connext DDS), the send window size of a *DataWriter* is set to infinite by default. This is done by changing the default values of two fields in DDS_RtpsReliableWriterProtocol_t (**min_send_window_size, max_send_window_size**) to DDS_LENGTH_UNLIMITED.

In RTI Data Distribution Service 4.4d, the send window feature was introduced and was enabled by default in 4.5c (with **min_send_window_size** = 32, **max_send_window_size** = 256). For *DataWriters* with a HistoryQosPolicy **kind** of KEEP_LAST, enabling the send window could cause writes to block, and possibly fail due to blocking timeout. This blocking behavior changed the expected behavior of applications using default QoS. To preserve that preestablished non-blocking default behavior, the send window size has been changed to be infinite by default starting in release 4.5d.

Users wanting the performance benefits of a finite send window will now have to configure the send window explicitly.

4.3 Extensible Types Compatibility

4.3.1 General Compatibility Issues

Connext DDS 5.x includes partial support for the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification[a] from the Object Management Group (OMG). 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 *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

4.3.2 Java Enumeration Incompatibility Issues

Connext DDS 5.2.0 fixed a bug (CODEGENII-397) in Java to resolve an interoperability issue with other languages when using enumerations with unordered enumeration values. For example:

```
enum MyEnum{
    THREE= 3,
    TWO= 2,
    ONE= 1
};
```

Because of this fix, a Java DataWriter built with Connext DDS 5.1.0 or lower will not match a *DataReader* of Connext DDS 5.2.0 or higher if the Topic contains an enumeration of the previous kind (with unordered enumeration values), and vice versa.

4.3.3 Interoperability Issues when Using Keyed Mutable Types

Starting with Connext DDS 5.2.2, the Keyhash calculation for a keyed mutable type in Java and .NET has changed in order to fix a language interoperability issue (see CODEGENII-501 in the Release Notes).

If you need your Java/.NET application built with 5.2.2 to communicate with a Java/.NET application built with an older version of Connext DDS, use the **-use52JavaKeyhash** flag to generate the code for your application.

4.4 ODBC Database Compatibility

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

[a]http://www.omg.org/spec/DDS-XTypes/

In principle, you can use any database that provides an ODBC driver, since ODBC is a standard. However, not all ODBC databases support the same feature set. Therefore, there is no guarantee that the persistent durability features will work with an arbitrary ODBC driver.

We have tested the following driver: MySQL ODBC 5.1.44.

Starting with 4.5e, support for the TimesTen database has been removed.

To use MySQL, you also need MySQL ODBC 5.1.6 (or higher). For non-Windows platforms, UnixODBC 2.2.12 (or higher) is also required.

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*.

For more information on database setup, please see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Database Setup*.

4.5 Transport Compatibility

4.5.1 Shared-Memory Transport Compatibility for Connext DDS 4.5f and 5.x

The shared-memory transport in Connext DDS 4.5f and higher does not interoperate with the shared-memory transport in previous releases of RTI Data Distribution Service.

If two applications, one using Connext DDS and one using RTI Data Distribution Service, run on the same node and they have the shared-memory transport enabled, they will fail with the following error:

```
[D0004|CREATE Participant|D0004|ENABLE]
NDDS_Transport_Shmem_is_segment_compatible:incompatible shared memory protocol detected.
Current version 1.0 not compatible with 2.0.
```

A possible workaround for this interoperability issue is to disable the shared-memory transport and use local communications over UDPv4 by setting **participant_qos.transport_builtin** to DDS_TRANSPORTBUILTIN_UDPv4.

If you have an interoperability requirement and you cannot switch to UDPv4, please contact **support@rti.com**.

4.5.2 Transport Compatibility for Connext DDS 5.1.0

4.5.2.1 Changes to message_size_max

In Connext DDS 5.1.0, the default **message_size_max** for the UDPv4, UDPv6, TCP, Secure WAN, and shared-memory transports changed to provide better out-of-the-box performance. Consequently, Connext DDS 5.1.0 is not out-of-the-box compatible with applications running older versions of Connext DDS or RTI Data Distribution Service.

To guarantee that communication between two applications always occurs: for a given transport, keep a consistent value for **message_size_max** in all applications within a Connext DDS system.

4.5.2.2 How to Change Transport Settings in Connext DDS 5.1.0 Applications for Compatibility with 5.0.0

If you need compatibility with a previous release, you can easily revert to the transport settings used in Connext DDS 5.0.0. The new built-in **Baseline.5.0.0** QoS profile contains all of the default QoS values from Connext DDS 5.0.0. Therefore, using it in a Connext DDS 5.1.0 application will ensure that Connext DDS 5.0.0 and 5.1.0 applications have compatible transport settings. Below is an example of how to inherit from this profile when configuring QoS settings:

```
<qos_profile name="MyProfile"
 base_name="BuiltinQosLib::Baseline.5.0.0">
   ...
</qos_profile>
```

4.5.2.3 How to Change message_size_max in Connext DDS 5.0.0 Applications for Compatibility with 5.1.0

The transport configuration can be adjusted programmatically or by using XML configuration. The following XML snippet shows how to change **message_size_max** for the built-in UDPv4 transport to match the Connext DDS 5.1.0 default setting of 65,507:

```
<participant_qos>
  <property>
    <value>
      <element>
        <name>
         dds.transport.UDPv4.builtin.parent.message_size_max
        </name>
        <value>65507</value>
      </element>
    </value>
  </property>
</participant_qos>
```

See Chapter 15, Transport Plugins, in the *RTI Connext DDS Core Libraries User's Manual* for more details on how to change a transport's configuration.

To help detect misconfigured transport settings, Connext DDS 5.1.0 will send the transport information, specifically the **message_size_max**, during participant discovery. Sharing this information will also make it easier for tools to report on incompatible applications in the system.

If two Connext DDS 5.1.0 *DomainParticipants* that discover each other have a common transport with different values for **message_size_max**, the Connext DDS core will print a warning message about that condition. Notice that older Connext DDS applications do not propagate transport information, therefore this checking is not done.

You can access a remote *DomainParticipant*'s transport properties by inspecting the new **transport_info** field in the DDS_ParticipantBuiltinTopicData structure. See Chapter 16, Built-in Topics, in the *RTI Connext DDS Core Libraries User's Manual* for more details about this field. There is a related new field, **transport_info_list_max_length**, in the DomainParticipantResourceLimitsQosPolicy. See Table 8.12 in the *RTI Connext DDS Core Libraries User's Manual* for more details about this field.

UDPv4, UDPv6, WAN, and TCP (Section Table 4.2 below) and Shared Memory (Section Table 4.3 below) show the new default transport settings.

Table 4.2 UDPv4, UDPv6, WAN, and TCP

	Old Default (bytes)	New Default (bytes)	
		Non-INTEGRITY Platforms	INTEGRITY Platforms[a]
message_size_max	9,216	65,507[b]	9,216
send_socket_buffer_size	9,216	131,072	131,072
recv_socket_buffer_size	9,216	131,072	131,072

Table 4.3 Shared Memory

	Old Default (bytes)	New Default (bytes)	
		Non-INTEGRITY Platforms	INTEGRITY Platforms
message_size_max	9,216	65,536	9,216
received_message_count_max	32	64	8
receive_buffer_size	73,728	1,048,576	18,432

[a]Due to limits imposed by the INTEGRITY platform, the new default settings for all INTEGRITY platforms are treated differently than other platforms. Please see the ***RTI Connext DDS Core Libraries Platform Notes*** for more information on the issues with increasing the **message_size_max** default values on INTEGRITY platforms. Notice that interoperation with INTEGRITY platforms will require updating the transport property **message_size_max** so that it is consistent across all platforms.

[b]The value 65507 represents the maximum user payload size that can be sent as part of a UDP packet.

4.5.2.4 Changes to Peer Descriptor Format

In Connext DDS 5.1.0, the way in which you provide a participant ID interval changed from **[a,b]** to **[a-b]**.

4.5.3 Transport Compatibility for Connext DDS 5.2.0

In Connext DDS 5.2.0, the UDPv6 and SHMEM transport kinds changed to address an RTPS-compliance issue (RTI Issue ID CORE-5788).

Transport	5.1.0 and lower	5.2.0 and higher
UDPv6	5	2
SHMEM	2	0x01000000 (16777216)

Because of this change, out-the-box compatibility with previous Connext DDS releases using both UDPv6 and SHMEM transports is broken. Connext DDS 5.1.0 and earlier applications will not communicate out-of-the-box with Connext DDS 5.2.0 applications over UDPv6 and SHMEM. (See below for how to resolve this.)

4.5.3.1 Observed Error Messages

You may see the following error messages when the UDPv6 transport is not enabled:

5.1.0 Application:

```
can't reach:
locator:
transport: 16777216
address: 0000:0000:0100:0000:0000:0000:0000:0000
port: 21410
encapsulation:
transport_priority: 0
aliasList: ""
```

5.2.0 Application:

```
PRESParticipant_checkTransportInfoMatching:Warning:
discovered remote participant 'yyyyy' using the 'shmem'
transport with class ID 2.
This class ID does not match the class ID 16777216 of the
same transport in the local participant 'xxxxx'.
These two participants will not communicate over the 'shmem'
transport.
Check the value of the property
'dds.transport.use_510_compatible_locator_kinds' in the
local participant.
COMMENDBeWriterService_assertRemoteReader:Discovered remote
reader using a non-addressable locator for a transport with
```

```
class ID 2.
This can occur if the transport is not installed and/or
enabled in the local participant. See
https://community.rti.com/kb/what-does-cant-reach-locator-error-message-mean
for additional info.
can't reach:
locator:
transport: 2
address: 0002:0000:0100:0000:0000:0000:0000:0000
port: 21412
encapsulation:
transport_priority: 0
aliasList: ""
```

4.5.3.2 How to Change Transport Settings in 5.2.0 Applications for Compatibility with 5.1.0

If you need compatibility with a previous release, there are two ways to do so:

- By setting the participant property **dds.transport.use_510_compatible_locator_kinds** to 1 in the Connext DDS 5.2.0 applications. For example:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>
            dds.transport.use_510_compatible_locator_kinds
                </name>
                <value>1</value>
            </element>
        </value>
    </property>
</participant_qos>
```

- By using the new built-in Generic.510TransportCompatibility profile. Below is an example of how to inherit from this profile when configuring QoS settings:

```
<qos_profile name="MyProfile"
 base_name="Generic.510TransportCompatibility">
...
</qos_profile>
```

4.5.4 Transport Compatibility for Connext DDS 5.2.0 on Solaris Platforms with Sparc CPUs

In Connext DDS 5.2.0, the SHMEM transport has changed to address a potential bus error on Solaris platforms on Sparc 64-bit CPUs[a] (). The change is not backward compatible with previous Sparc Solaris

[a]RTI Issue ID CORE-6777

releases (32-bit or 64-bit).

If a 5.2.0 application tries to communicate with an older version using the shared memory transport, you will see this error:

```
[D0004|CREATE Participant|D0004|ENABLE]
NDDS_Transport_Shmem_is_segment_compatible:
incompatible shared memory protocol detected.
Current version 3.0 not compatible with x.0.
```

To avoid this error, disable the shared memory transport by setting the QoS policy **participant_qos.transport_builtin** do that it does not contain the shared memory transport. For example:

```
<participant_qos>
    <transport_builtin>
        <mask>UDPv4</mask>
    </transport_builtin>
</participant_qos>
```

4.6 Other Compatibility Issues

4.6.1 ContentFilteredTopics

- Starting with 5.1.0, Connext DDS includes a change in the generated typecode name when *rtiddsgen's* **-package** option is used in Java. This change introduces the following backward-compatibility issue.

  DataWriters in 4.5x and below will not be able to perform writer-side filtering for Connext DDS 5.1.0 *DataReaders* using ContentFilteredTopics. You will get a ContentFilteredTopic (CFT) compilation error message on the DataWriter side. Notice that this compatibility issue does not affect correctness, as the filtering will be done on the *DataReader* side.

4.6.2 Built-in Topics

Due to the addition of new values in the enumeration DDS::ServiceQosPolicyKind under DDS::ServiceQosPolicy, Connext DDS 5.1.0 or lower applications will not be able to get information about *DataWriters* and *DataReaders* created by Connext DDS 5.2.0 or higher Infrastructure Services by reading from the Publication and Subscription built-in topic DataReaders.

These infrastructure services are affected:

- RTI Routing Service
- RTI Recording Service (Record and Replay tools)

- RTI Database Integration Service (formerly, RTI Real-Time Connect)
- RTI Queuing Service

The 5.1.0 applications monitoring the built-in topics will print the following error message:

```
DDS_ServiceQosPolicy_from_presentation_service_kind:ERROR:
  Failed to get service (unknown kind)
DDS_SubscriptionBuiltinTopicDataTransform:ERROR:
  Failed to get service
PRESCstReaderCollator_addSample:!transform
```

Notice that this compatibility issue does not affect matching between the 5.1.0 *DataReaders/DataWriters* and the corresponding 5.2.0 Infrastructure Service *DataReader/DataWriters*. Within the middleware, discovery will still occur and these entities will communicate with each other.

4.6.3 Some Monitoring Types are not Backwards Compatible

Due to the way in which the type-assignability algorithm handled enumerations in Connext DDS 5.1.0, some of the monitoring types are not compatible with newer versions of the types. This means Connext DDS 5.1.0 applications using the monitoring library may fail to match with the newer versions of the monitoring types. Newer versions of the monitoring library, however, will match with older versions of the monitoring types because of updates that have been made to the type assignability algorithm.

The only incompatibility as of the release of 5.2.0 is with the DataReaderDescription and DataWriter-Description monitoring types. Monitoring applications built with 5.1.0 will not be able to subscribe to the DataReaderDescription and DataWriterDescription topics published by applications that are using version 5.2.0 and later.

[RTI Issue ID MONITOR-188]

Chapter 5 What's Fixed in 5.2.3

Release 5.2.3 is a maintenance release based on the feature release 5.2.0.[a] This section describes bugs fixed in 5.2.3.

For an overview of new features and improvements in 5.2.3, please see the separate ***What's New*** document for 5.2.3.

5.1 Fixes Related to Content Filters

5.1.1 Creating ContentFilteredTopic with SQL Filter Crashed if Provided Expression had Empty String as Predicate

Creating a ContentFilteredTopic using the built-in SQL filter would crash if the provided expression contained empty predicates, such as {{"x = 1 OR"}}, {{"OR"}}, {{""}}, etc. This is grammatically incorrect, so the issue has been resolved by reporting an error on invalid grammar, as shown below:

```
D0060|Reader(80000004)|T=Topic|CREATE READCONDITION]DDS_SqlFilterparse:syntax error,
unexpected $end
[D0060|Reader(80000004)|T=Topic|CREATE READCONDITION]DDS_SqlFilter_compile:SQL
compiler failed with error-code: -1 (Syntax error)
...
```

[RTI Issue ID CORE-6917]

5.1.2 Potential Memory Corruption when Using ContentFilteredTopic with Filter Expression Parameters

There may have been memory corruption when using ContentFilteredTopics with expression parameters. In particular, this issue may have occurred after creating a ContentFilteredTopic whose

[a]For What's Fixed in 5.2.0, see the *RTI Connext DDS Core Libraries Release Notes* provided with 5.2.0.

parameters resulted in a content filter property length equal to the maximum specified by DomainParticipantResourceLimitsQosPolicy's **contentfilter_property_max_length**. This problem has been resolved.

[RTI Issue ID CORE-7351]

5.1.3 SQL ContentFilteredTopic Incorrectly Discarded Samples -- Java and C# APIs Only

When using the Java or C# APIs, a SQL ContentFilteredTopic may have incorrectly discarded samples. In particular, this may have occurred for complex types with at least one 8-byte type and one variable-length type (such as a DDS_LongLong plus a sequence). This problem has been resolved.

[RTI Issue ID CORE-7296]

5.1.4 SQL Content Filter could Incorrectly Evaluate Expression or Crash if HEX Function used in Some Combinations

This issue occurred if you created a ContentFilteredTopic that used the SQL filter class and:

- the expression included more than one Hex function. For example:

```
_octet = &hex(00) OR _octet = &hex(01)
```

- the expression was complex and included at least one Hex function. For example:

```
octet_array=&hex(EE) OR x=1
```

In this case, the evaluation of such a type of expression may have been incorrect, depending on the sample content, or even have caused a crash on ContentFilteredTopic creation. This problem has been resolved.

[RTI Issue ID CORE-6913]

5.1.5 Wrong Result when Calling ContentFilteredTopic's get_expression()

Calling a ContentFilteredTopic's **get_expression()** API may have returned a result that did not match the actual currently active filter. In particular, this issue was triggered by a failed call to **set_expression()**. This problem has been resolved.

[RTI Issue ID CORE-6883]

5.1.6 Potential Memory Corruption when Using ContentFilteredTopic with Filter Expression Parameters

There may have been memory corruption when using ContentFilteredTopics with expression parameters. In particular, this issue may have occurred after creating a ContentFilteredTopic whose parameters resulted in a content filter property length equal to the maximum specified by DomainParticipantResourceLimitsQosPolicy's **contentfilter_property_max_length**. This problem has been resolved.

[RTI Issue ID CORE-7351]

5.1.7 Wrong Value Returned from ContentFilteredTopic set_expression() and set_expression_parameters()

A call to **set_expression()** or **set_expression_parameters()** may have incorrectly returned DDS_ RETCODE_OK upon failure. For example, this issue may have occurred as a consequence of trying to update the filter with a filter expression that was too long. When this issue triggered, the following error was printed:

```
PRESContentFilteredTopic_updateFilterExpression:!copy sequence for content filtered property
data
```

This problem has been resolved.

[RTI Issue ID CORE-7398]

5.2 Fixes Related to Discovery

5.2.1 Setting NDDS_DISCOVERY_PEERS Incorrectly Modified multicast_ receive_addresses when initial_peers Specified in QoS XML Profile

If NDDS_DISCOVERY_PEERS is set using the environment variable or file, the **initial_peers** and **multicast_receive_addresses** fields (in the Discovery QoS policy) are modified according to the value set. However, if an XML QoS profile is used, the values of i**nitial_peers** and **multicast_receive_addresses** in the profile should override the values set by NDDS_DISCOVERY_PEERS.

Previously, the **multicast_receive_addresses** value would be set to the value derived from NDDS_ DISCOVERY_PEERS when an XML QoS profile explicitly set the value of **initial_peers** but did not set the value of **multicast_receive_addresses**. So even though the XML QoS profile's setting of **initial_peers** is supposed to override all effects of NDDS_DISCOVERY_PEERS, **multicast_receive_addresses** was still being modified by the environment variable or file.

The behavior has been modified such that if an XML QoS profile explicitly sets **initial_peers** but not **multicast_receive_addresses**, the value of **multicast_receive_addresses** will be the default value of "builtin.udpv4://239.255.0.1”. Of course, if an XML QoS profile explicitly sets **multicast_receive_addresses**, that value will override the default value.

[RTI Issue ID CORE-6817]

5.2.2 Default Domain Announcement could not be Completely Disabled

If you disabled default domain announcement by setting **discovery_config.default_domain_announcement_period** to UNLIMITED, you may have seen errors such as the following:

```
[D0010|CREATE Participant|D0010|ENABLE]COMMENDAnonWriterService_setWriterProperty:!create
domain broadcast multicast destination
```

The above error was generated if you disabled transport multicast by setting the transport property **dds.transport.UDPv4.builtin.multicast_enabled** to 0. This error should not have been generated because the default domain announcement feature was disabled. Notice that although the feature was not completely disabled, announcements were not being sent on the wire. Therefore, the error message was harmless from a bandwidth-utilization point of view. The problem has been resolved.

[RTI Issue ID CORE-7196]

5.2.3 Potential Interoperability Issue with Remote Participants Announcing Too Many Locators

Discovering a remote Participant that was announcing more than four locators triggered deserialization errors on the local Participant. When these errors occurred, discovery of the remote Participant failed.

This problem has been resolved. Instead of preventing discovery, the reception of too many locators will trigger a warning, and the local Participant will ignore the excessive locators.

[RTI Issue ID CORE-6872]

5.2.4 Unexpected "remote endpoint not previously asserted" Warnings

You may have seen these unexpected warnings during Discovery if liveliness with a remote Participant was lost:

```
DISCEndpointDiscoveryPlugin_unregisterRemoteReader:RTI0xXXXXXXXX:remote endpoint not
previously asserted by plugin
```

The messages were misleading since the middleware was operating as expected. This problem has been resolved by changing the verbosity of the message to NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE and by clarifying the message.

[RTI Issue ID CORE-6799]

5.3 Fixes Related to Modern C++ API

5.3.1 Certain Functions Receiving Samples of IDL Types with Strings may have Caused Heap Corruption–Modern C++ API only

Using certain functions that received samples of some C++ types generated from IDL may have corrupted the heap. In particular, the problem affected types containing bounded strings and the use of functions that copy the sample, such as **DataReader::key_value()** or **DataReader::take(T&)** (but not the **DataReader::take()** that returns LoanedSamples). This problem has been resolved.

[RTI Issue ID CORE-6887]

5.3.2 ContentFilteredTopic Function to Modify Filter Expression Missing - Modern C++ API Only

A ContentFilteredTopic provides a function to modify the filter expression once it has been created. (This is a Connext DDS extension.) However the Modern C++ API didn't include this function. This problem has been resolved; now dds::topic::ContentFilteredTopic includes the following function:

```
void filter(const dds::topic::Filter& filter)
```

[RTI Issue ID CORE-7309]

5.3.3 DataReader::lookup_instance was not Implemented (Modern C++ API only)

The function **DataReader::lookup_instance()** was not implemented in the Modern C++ API. Applications attempting to call it would have seen a compilation error. The function is now available.

[RTI Issue ID CORE-6877]

5.3.4 DataReader's Read/Take Functions that Make a Copy didn't Indicate if Data was Available or Valid–Modern C++ API Only

The following overloads of the *DataReader's***read()** functions (and equivalent **take()** functions) did not indicate whether data was available to read.

```
void read(T& sample)
void read(T& sample, dds::sub::SampleInfo& sample_info)
```

The first one also did not indicate if the data was valid (i.e. SampleInfo::valid() == true).

This problem has been resolved. These functions now return a boolean that says if data was available and copied into the sample. Also, the first one will now skip any sample with invalid data.

[RTI Issue ID CORE-7035]

5.3.5 DataReader::Selector::condition() only Accepted QueryConditions - Modern C++ API Only

The Selector (obtained from **DataReader::select()**) can receive a *Condition* to read data samples from the *DataReader* that match certain criteria.

In the previous release, the Selector only accepted *QueryConditions*. In this release, it can also accept the base class, *ReadCondition*.

[RTI Issue ID CORE-7207]

5.3.6 DataWriter::is_sample_app_acknowledged() Missing – Modern C++ API Only

The function **DataWriter::is_sample_app_acknowledged()** was missing from the Modern C++ API. This function, which indicates if the subscribing application has acknowledged a given sample, is now available in the Modern C++ API.

[RTI Issue ID CORE-7294]

5.3.7 Entities may have been Deleted Unexpectedly if Listener was Attached–Modern C++ API Only

Entities are reference types and, as such, when the last reference goes out of scope the object is deleted. However, there are several situations in which the *Entity* is retained. For example, if an *Entity* has a Listener attached, the Entity must be retained.

In previous releases, if a Listener was attached after the *Entity* was created, the *Entity* was not retained and may have been deleted unexpectedly.

This problem has been resolved. Now an *Entity* will be retained whenever it has a Listener, regardless of when the Listener is set.

[RTI Issue ID CORE-7345]

5.3.8 rti::domain::find_participant_by_name() API not Properly Exported for DLLs – Windows Only, Modern C++ API Only

The **rti::domain::find_participant_by_name()** API was not properly exported for Windows platforms. Therefore it could not be used when linking against the dynamically linked Connext DDS**nddscpp2.dll** library. This problem has been resolved.

[RTI Issue ID CORE-7238]

5.3.9 Headers rti.hpp and rticore.hpp did not Include Some Headers–Modern C++ API Only

The header **rti/rti.hpp** should include all the APIs and **rti/core/rticore.hpp** should include all the core namespaces, but they were missing two headers: **ListenerBinder.hpp** and **EntityLock.hpp**. This problem has been resolved and these headers are included appropriately.

[RTI Issue ID CORE-7313]

5.3.10 Incorrect Signature of EntityLock Constructor – Modern C++ API only

The signature of **rti::core::EntityLock**'s constructor didn't allow the following code to compile:

```
DomainParticipant participant(0);
EntityLock lock(participant); // Compilation error
```

The constructor couldn't directly receive an *Entity*-derived class, such as *DomainParticipant* in this example. The application would have had to first cast the participant to *Entity*. This problem has been resolved.

[RTI Issue ID CORE-7311]

5.3.11 Many Warnings about Unused Parameters and Unused Local Typedefs – Modern C++ API Only

Applications that included the Connext DDS Modern C++ API may have seen a significant number of warnings about unused parameters and unused local typedefs, depending on the compiler version and the compilation options. All these warnings have been fixed. Note, however, that none of these warnings revealed an underlying defect.

[RTI Issue ID CORE-6918]

5.3.12 Modern C++ API did not Provide Function to Look Up ContentFilteredTopic

The Modern C++ API didn't support looking up ContentFilteredTopics (Topics were fine). This problem has been resolved. Now the standalone function dds::topic::find() works for both Topics and ContentFilteredTopics, as well as the base class, TopicDescription.

The following example shows how to look up instances of these three types:

```
using namespace dds::topic;

auto my_content_filtered_topic =
    find<ContentFilteredTopic<Foo> >(participant, "my_cft");


auto my_topic = find<Topic<Foo> >(participant, "my_topic");
auto my_content_filtered_topic_as_topic_desc =
    find<TopicDescription<Foo> >(participant, "my_cft");
```

[RTI Issue ID CORE-6989]

5.3.13 Modern C++ APIs to_cdr_buffer() and from_cdr_buffer() for Builtin Types were Missing

The **to_cdr_buffer()** and **from_cdr_buffer()** APIs for builtin types were not available in the Modern C++ API. This problem has been resolved by adding these APIs.

[RTI Issue ID CORE-6843]

5.3.14 Modern C++ API did not Allow Use of Functions Receiving std::initializer_list – Visual Studio 2013 Platforms Only

The API headers detect compiler support for several C++ 11 features and enable or disable certain types and functions that use them. However, the API headers incorrectly disabled functions that receive **std::initializer_list** as a parameter when compiling with Visual Studio 2013 (even though this compiler does support this C++ 11 feature). This problem has been resolved. Functions using std::initializer_list will now compile with Visual Studio 2013.

[RTI Issue ID CORE-6856]

5.3.15 Problem in Creation of DynamicData Objects–Modern C++ API Only

The DynamicData object didn't make a copy of the DynamicType it receives. If the scope of the DynamicType was shorter than that of the DynamicData instance, some operations of DynamicData that accessed the type may have caused undefined behavior. This problem has been resolved.

[RTI Issue ID CORE-6960]

5.3.16 ReadConditions and QueryConditions were not Cleaned Up in DataReader::close() -- Modern C++ API Only

In Modern C++, the **close()** operation on a *DataReader* did not close any attached ReadConditions or QueryConditions. This problem has been resolved. Additionally, ReadConditions and QueryConditions now also provide a **close()** operation.

[RTI Issue ID CORE-7241]

5.3.17 Setting DomainParticipantFactoryQos may have Overridden Default QosProvider's QosProviderParam Settings – Modern C++ API Only

Getting and setting the DomainParticipantFactoryQos after setting the Default QosProvider QosProviderParams would clear those settings. This problem has been resolved.

[RTI Issue ID CORE-6900]

5.3.18 StatusCondition didn't Allow Setting Dispatch Handler–Modern C++ API Only

In the Modern C++ API, you can attach a handler to a condition that gets called by **WaitSet::dispatch()** when the condition becomes active. While *GuardCondition* and *ReadCondition* included this capability, *StatusCondition* did not. This made **WaitSet::dispatch()** and handlers unusable when mixing these three conditions in the same WaitSet. This problem has been resolved; now *StatusCondition* also supports a handler.

[RTI Issue ID CORE-6921]

5.3.19 XML Application-Creation APIs could not be Used with Non-Default QosProvider – Modern C++ API Only

The XML Application-Creation APIs only worked if the *default* QosProvider was used to load the configuration files. There is no need for this restriction and this issue has been resolved. Now any user-created QosProvider can be used to load XML Application configurations and then access the entities in those configurations.

[RTI Issue ID CORE-6901]

5.4 Fixes Related to Transports

5.4.1 Connectivity not Recovered on TCP Client after Disconnecting Cable when Using Windows IOCP and TLS Support

Connectivity may not have recovered on a TCP client after disconnecting and reconnecting the cable. In particular, this issue may have been triggered when using Windows IOCP and TLS Support. This problem has been resolved.

[RTI Issue ID COREPLG-380]

5.4.2 Connectivity not Recovered on TCP Client after Disconnecting Cable when Using Windows IOCP and Load Balancer Support

Connectivity may not have recovered on a TCP client after disconnecting and reconnecting the cable. In particular, this issue may have been triggered when using Windows IOCP and Load Balancer support.

This problem has been resolved.

[RTI Issue ID COREPLG-379]

5.4.3 DTLS Transport Plugin Printed Wrong Exception when Using create_function_ptr property

The DTLS Transport plugin **create_function_ptr** property was not properly validated. Therefore, the following message was printed when that property was used:

```
[D0000|CREATE Participant|D0000|ENABLE]NDDS_Transport_DTLS_plugin_property_from_DDS_
property:Unexpected property: dds.transport.DTLS.dtls1.create_function_ptr. Closest valid
property: dds.transport.DTLS.dtls1.create_function
```

Note that, despite of the exception, the property was having effect. This problem has been resolved.

[RTI Issue ID COREPLG-364]

5.4.4 Local Plugin Crashed after Ungracefully Closing Connections on Remote TCP Transport Plugin–OS X Platforms Only

If a remote TCP plugin running as a client ungracefully closed a TCP connection, the local application crashed due to an unhandled SIGPIPE signal. This problem has been resolved.

[RTI Issue ID COREPLG-363]

5.4.5 Misleading Warning when Shared Memory Configured in one DomainParticipant but not in Others

If two *DomainParticipants* were configured so that the Shared Memory transport was enabled in one of them but not in the other, the DomainParticipant without Shared Memory printed the following warning:

```
COMMENDBeWriterService_assertRemoteReader:Discovered remote reader using a non-addressable
locator for a transport with class ID 16777216.
This can occur if the transport is not installed and/or enabled in the local participant.
See https://community.rti.com/kb/what-does-cant-reach-locator-error-message-mean for
additional info.
```

This warning was misleading since both *DomainParticipants* can usually communicate using other transport such as UDPv4. Starting with this release, the warning is suppressed.

[RTI Issue ID CORE-7290]

5.4.6 Possible Crash in TCP Transport Plugin Upon Failed Connection Negotiation

There was an issue in the TCP Transport plugin that may have caused a crash. This occurred when a TCP Transport plugin running as a client received a "connection bind error response" from the TCP Transport plugin running as a server. The issue only affected connections used for receiving data in the client. This problem has been resolved.

[RTI Issue ID COREPLG-370]

5.4.7 Potential Hang During Participant Shutdown – Windows Systems Only

It was possible for an application to hang during participant shutdown if the participant was using the builtin UDPv4 transport. This was a rare race condition that was more likely to occur in a system with high throughput. This problem has been resolved.

[RTI Issue ID CORE-7248]

5.4.8 SIGPIPE Signal upon Connection Reset when using TCP Transport with TLS Support –Linux and QNX Platforms Only

When using TCP Transport with TLS Support on Linux or QNX architectures, a connection reset may have resulted in a SIGPIPE signal. This problem has been resolved.

[RTI Issue IDs COREPLG-302, COREPLG-395]

5.4.9 TCP Transport Plugin Printed Wrong Exception when Using create_function_ptr Property

The TCP Transport plugin's **create_function_ptr** property was not properly validated. Therefore, the following message was printed when that property was used:

```
NDDS_Transport_TCPv4_plugin_property_from_DDS_property:Unexpected property:
dds.transport.TCPv4.tcp1.create_function_ptr. Closest valid property:
dds.transport.TCPv4.tcp1.create_function
```

Note that, despite the exception, the property did have an effect. This problem has been resolved.

[RTI Issue ID COREPLG-362]

5.4.10 Unexpected "no transport for destination request" Warning in Applications using Shared Memory Transport

You may have seen the following warning when using the Shared Memory transport in applications running on different nodes:

```
RTINetioSender_addDestination:no transport for destination request
shmem://0002:0105:0101:0000:0000:0000:0000:0000:12160
```

This warning was misleading, as it is perfectly normal for two applications running on different nodes to not communicate over shared memory.This problem has been resolved.

[RTI Issue ID CORE-6625]

5.4.11 Warning Issued if Loopback was the Only Available UDP Interface

The UDPv4 transport reported the following warning when the only interface available for UDP communication was loopback.

```
WARNING: No interfaces for transport udpv4 match allowed patterns.
```

This problem has been resolved. The warning will no longer appear.

[RTI Issue ID CORE-6380]

5.5 Other Fixes

5.5.1 autopurge_unregister_instances_delay had no Effect when Using Monotonic Clock

Unregistered samples were not purged if the internal clock was set to monotonic, even after **autopurge_unregister_instances_delay** (in the WriterDataLifeCycle QoS policy) expired. This problem has been resolved.

[RTI Issue ID CORE-7066]

5.5.2 Creating DataReader with DATAREADER_QOS_USE_TOPIC_QOS and ContentFilteredTopic Failed–Java API Only

The creation of a *DataReader* with the special QoS that indicates that the Topic QoS should be used (**Subscriber.DATAREADER_QOS_USE_TOPIC_QOS**) failed when using a ContentFilteredTopic. For example, the following call threw the exception com.rti.dds.infrastructure.RETCODE_ERROR.

```
my_subscriber.create_datareader(
        my_content_filtered_topic,
        Subscriber.DATAREADER_QOS_USE_TOPIC_QOS,
        my_listener,
        StatusKind.STATUS_MASK_ALL);
```

This problem has been resolved. Now in this situation, the QoS of the ContentFilteredTopic's associated Topic will be used, as it is in the other language APIs.

[RTI Issue ID CORE-6892]

5.5.3 Database Shutdown Timeout Error during Participant Deletion after Publisher/Subscriber Creation Failure

If the creation of a *Publisher* or *Subscriber* failed, the subsequent *DomainParticipant* deletion produced the following error and lead to memory leaks.

```
[D0156|DELETE Participant]DDS_DomainParticipantDatabase_prefinalize:!database shutdown
timeout
```

```
[D0156|DELETE Participant]DDS_DomainParticipant_destroyI:!shut down database
[D0156|DELETE Participant]DDS_DomainParticipantFactory_delete_participant:!delete participant
```

This problem has been resolved.

[RTI Issue ID CORE-7316]

5.5.4 DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS was not Documented and Implemented in All Language APIs

The DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS was not documented in the C API and was not implemented in the .NET and traditional C++ API DDS:: namespace. This problem has been resolved.

[RTI Issue ID CORE-7298]

5.5.5 DATA_WRITER_INSTANCE_REPLACED_STATUS not Implemented in All Language APIs

The DATA_WRITER_INSTANCE_REPLACED_STATUS was not implemented in the .NET API, the Modern C++ API, and the Traditional C++ API DDS:: namespace. This problem has been resolved.

[RTI Issue ID CORE-7318]

5.5.6 DDS_RTPS_AUTO_ID_FROM_MAC not Used if Set in Code

Previously if using the DDS_RTPS_AUTO_ID_FROM_MAC QoS in code (not XML) the MAC ID was not used as part of the RTPS Globally Unique Identifier (GUID). Prior to this fix, you would have seen the IP address used in the GUID rather than the MAC address. This problem has been resolved.

[RTI Issue ID CORE-6832]

5.5.7 Duplicate Declarations in Header Files

The following duplicate declarations were found in Connext DDS header files:

- PRESPsReaderGroup_getStatusChange (pres/pres_psGroup.h)
- RTINetioLocator_compare (netio/netio_common.h)
- PRESParticipant_setLocalParticipantConfigListener (pres/pres_participant.h)

Depending on the enabled compiler flags, the compiler may have printed warning messages on these declarations. This problem has been resolved and the extra declarations have been removed.

[RTI Issue ID CORE-6541]

5.5.8 Error Creating DataWriters/DataReaders after Many DataWriters/DataReaders Already Created/Deleted

Connext DDS did not check whether an entity GUID (assigned automatically by default) was already in use before assigning it to a new *DataWriter/DataReader*. This may have caused the entity creation to fail with a messages like these:

```
PRESPsService_createLocalEndpoint:!assert pres psWriter
DDS_DataWriter_create_presentation_writerI:!create LocalEndpoint
DDS_DataWriter_createI:!create PRESPsWriter
DDS_Publisher_create_datawriter_disabledI:!create DataWriter
DDSDataWriter_impl::createI:!create writer
```

You would have only run into this problem if you had previously created/deleted thousands of *DataWriter-s/DataReaders* (so that the counter used to build GUIDs rolled over).

This problem has been resolved.

[RTI Issue ID CORE-6996]

5.5.9 Error when Invoking end_coherent_changes() with Batching Enabled

If a *Publisher* called **end_coherent_changes()**, and that *Publisher* contained a *DataWriter* with batching enabled, you may have seen an error such as:

```
[1408109599.718719] [D0200|Pub(308)|END COHERENT CHANGES]WriterHistoryMemoryPlugin_
addEndCoherentChangeSample:empty coherent set
[1408109599.718750] [D0200|Pub(308)|END COHERENT CHANGES]PRESWriterHistoryDriver_flush:!add_
batch_sample_group
[1408109599.718758] [D0200|Pub(308)|END COHERENT CHANGES]PRESPsWriter_
flushBatchWithCursor:!collator addWrite
[1408109599.718765] [D0200|Pub(308)|END COHERENT CHANGES]PRESPsWriterGroup_
endCoherentChanges:!flush batch
```

This error was only returned if the outstanding batch being built for the batching-enabled *DataWriter* was empty. This problem has been resolved.

[RTI Issue ID CORE-6389]

5.5.10 Error Restoring Unkeyed Durable Writer History when autopurge_disposed_instances_delay Set to Zero

In Connext DDS 5.2.0, restoring the history of an unkeyed *DataWriter* configured with Durable Writer History may have failed if the *DataWriter's* QoS **autopurge_disposed_instances_delay** was set 0. You may have seen error messages such as the following when trying to restore:

```
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
beginDisposedInstanceIteration:!find disposed instances - ODBC: invalid handle
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
restoreDisposedInstanceCache:!beginDisposedInstanceIteration
```

```
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
createHistory:!restore disposed-instance cache
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
beginDisposedInstanceIteration:!find disposed instances - ODBC: invalid handle
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
purgeReclaimableDisposedInstancesInDB:!beginDisposedInstanceIteration
[D0256|Pub(408)|T=Example Topic|CREATE Writer][FATAL] WriterHistoryOdbcPlugin_
createHistory:!purge reclaimable disposed instances
[D0256|Pub(408)|T=Example Topic|CREATE Writer]PRESWriterHistoryDriver_new:!create _whHnd
```

This problem has been resolved.

[RTI Issue ID CORE-6998]

5.5.11 IDL Enum did not Implement java.io.Serializable Correctly–Java API Only

The creation of an instance of a subclass of com.rti.dds.util.Enum by serializing it in Java yielded an object whose equal() method would always return false when compared with other instances of the same Enum subclass.

[RTI Issue ID CORE-6604]

5.5.12 Interoperability Problem between C/C++ and Java/.NET Applications when Using Keyed Mutable Types

In Connext DDS version 5.2 and below, the instance keyhash for keyed mutable types was calculated differently in C/C++ and Java/.NET languages. As a result, the subscribing application might have observed some unexpected behavior related to instances. Specifically, the call to **DataReader::lookup_instance()** might have failed and returned HANDLE NIL, even if the instance was received. Because of this failure, RTI Spreadsheet Add-In for Microsoft® Excel® may have failed to show the values for this instance. This problem has been resolved.

[RTI Issue ID CODEGENII-501]

5.5.13 Log Verbosity Settings may have been Overridden by DomainParticipantFactory's set_qos()

An application that called **DDS_DomainParticipantFactory_set_qos()** would always override the log verbosity set by previous calls to **NDDS_Config_Logger_set_verbosity()**. This is not expected if the provided LoggingQosPolicy does not contain changes. This issue has been resolved so the log verbosity settings are overridden by **DDS_DomainParticipantFactory_set_qos()** only when the provided LoggingQosPolicy contains any modifications with regards the latest settings.

[RTI Issue ID CORE-7249]

5.5.14 Memory Leak in Request/Reply APIs that Operated with SampleIdentity -- .NET API Only

In the Request-Reply .NET API, any operation that received a SampleIdentity as an argument would generate a memory leak. This problem has been resolved.

[RTI Issue ID CORE-7005]

5.5.15 Partition Information not Propagated Correctly

This issue was reproduced by creating a *Publisher* with a *DataWriter* or a *Subscriber* with a *DataReader* as disabled entities and after that, changing the partition information in the *Publisher*/*Subscriber*. This partition information was not properly passed from the *Publisher*/*Subscriber* to the endpoint (*DataWriter*/*DataReader*), therefore potentially preventing the communication. This problem has been resolved.

[RTI Issue ID CORE-5827]

5.5.16 Ping, Spy, Prototyper, and rtiddsgen-Generated C/C++ Examples may have Crashed–Android Platforms Only

If you ran **rtiddsping**, **rtiddsspy**, or **rtiprototyper** on an Android platform for some time, it may have crashed. This problem may also have occurred when using the *rtiddsgen*-generated application for Android C/C++ examples. This problem has been resolved.

[RTI Issue ID CORE-7170]

5.5.17 Possible Crash Creating Requester if autoenable_created_entities False

An application creating a *Requester* with a participant that set **participant_qos.entity_factory.autoenable_created_entities** to false may have crashed. While that is not supported, the *Requester* creation should have failed gracefully. This problem has been resolved. Now in that situation, the *Requester* creation will fail with DDS_RETCODE_NOT_ENABLED.

[RTI Issue ID REQREPLY-31]

5.5.18 Possible Segmentation Fault in Receiver Thread during Participant Shutdown

A rare race condition in the Connext DDS internal database may have caused a segmentation fault during the removal of a record. This issue may have occurred during a Participant shutdown and appeared as a segmentation fault in the receive thread. This problem has been resolved.

[RTI Issue ID CORE-6928]

5.5.19 Potential Crash when Calling to Get DataWriter/DataReader QoS

There was an issue that may have caused a crash when calling to get *DataWriter/DataReader* QoS. In particular, this issue may have been triggered by the following sequence of actions:

1. Set the *DataWriter/DataReader* default EntityName QoS to a non-empty string value,
2. Change the *DataWriter/DataReader* EntityName QoS to a larger string,
3. Call to get the *DataWriter/DataReader* QoS.

This problem has been resolved.

[RTI Issue ID CORE-7262]

5.5.20 Reliable DataReader Stopped Receiving Data from Reliable DataWriter after Running Long Time and Losing/Regaining Liveliness

A reliable *DataReader* may have stopped receiving data from a reliable *DataWriter* after running for a long period of time. This issue only occurred when the Participant liveliness between the publishing and subscribing applications was lost and regained after that long period of time. The length of time after which communication stopped depended on the configuration of the RTPS reliability protocol.

The issue only occurred when the *DataReader* sent more than 2 billion ACKNACK messages across all its matching *DataWriters*. In Connext DDSS, ACKNACK messages are sent in response to HEARTBEATS. For example, assuming only one *DataWriter* matching the *DataReader* and the absence of piggyback HEARTBEATs, if the periodic HEARTBEAT was 10 msec., it would have taken 231 days to reach 2 billion ACKNACKs and trigger this issue.

This problem has been resolved. Now in the above situation, communication will still occur, regardless of the number of ACKNACKs.

[RTI Issue ID CORE-7133]

5.5.21 Serialization Errors for Unkeyed Types when Using Batching – Java API Only

There was an issue that may have caused serialization errors for unkeyed types when using batching in the Java API. In particular, this issue was triggered when setting the property **dds.data_writer-.history.memory_manager.java_stream.min_size** to a value smaller than 4. This problem has been resolved.

[RTI Issue ID CORE-7333]

5.5.22 Specifying NULL Function for NDDS_Config_LoggerDeviceCloseFnc caused Segmentation Fault

Specifying a NULL pointer for a NDDS_Config_LoggerDeviceCloseFnc caused a segmentation fault if the logging device was set more than once. This problem has been resolved.

[RTI Issue ID CORE-6896]

5.5.23 Uncommon Race Condition in DataWriter Deletion

An uncommon race condition may have caused the deletion of DataWriters to fail and leak memory or crash. A DataWriter could only have run into the race condition if it enabled batching with a non-infinite flush period. This problem has been resolved.

[RTI Issue ID CORE-6903]

5.5.24 Valid URL Syntax Not Accepted–Windows Systems Only

When providing URLs of XML files to load in the ProfileQosPolicy (or to the QosProvider in the Modern C++ API), valid URLs were not accepted. On Windows systems, the correct syntax for a URL is **file:///c:/path/to/thefile.xml**, with 3 forward slashes ('///') after 'file:', however, the Connext DDS parser was expecting only 2 forward slashes ('//'). This problem has been resolved.Now URLs with either 2 or 3 forward slashes will be accepted in order to preserve backwards compatibility.

[RTI Issue ID CORE-7011]

5.5.25 XSD Validation Failed when Configuring multi_channel on DataWriter

The XSD validation of QoS profiles using the file **rti_dds_qos_profiles.xsd** failed when you tried to configure a <multi_channel> on a <datawriter_qos>. The problem has been resolved.

[RTI Issue ID CORE-7075]

5.5.26 Changed Verbosity for "remote entity ignored by user" Messages

The verbosity of "remote entity ignored by user" warning messages has been changed from WARNING to STATUS_REMOTE. Since these messages inform you of expected behavior, they should not be warnings.

[RTI Issue ID CORE-7197]

5.5.27 Rare Segmentation Fault when Using ODBC Writer History

You may have seen a rare segmentation fault when shutting down a *DomainParticipant* that was used to create *DataWriters* using ODBC writer history. Before the segmentation fault, the application printed this error message:

```
WriterHistoryOdbcPlugin_cleanup:!delete event thread
```

This problem has been resolved.

[RTI Issue ID CORE-7399]

5.5.28 Possible Crash if Reliable DataReader Received Data from DataWriter Running for Long Time

An application may have crashed if a reliable *DataReader* received data from a reliable *DataWriter* that was running for a long period of time.

This issue only occurred if the *DataReader* received a sample with a RTPS Sequence Number (SN) that was at least 2ˆ31 times larger than the RTPS SN of the last received sample. In other words, the *DataWriter* must have generated a GAP message containing 2^31 or more samples.

The computation of the distance between two sequence numbers could have overflowed and returned an invalid value that led to invalid memory access.This problem has been resolved so the overflow is handled properly and communication is not affected.

[RTI Issue ID CORE-7411]

Chapter 6 Known Issues

6.1 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]

6.2 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 Section 12.4, Durable Reader State, in the *RTI Connext DDS Core Libraries User's Manual*.

[RTI Issue ID CORE-5360]

6.3 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]

6.4 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 Chapter 22, Introduction to the Request-Reply Communication Pattern, in the *RTI Connext DDS Core Libraries User's Manual*).

[RTI Issue ID CORE-5181]

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

Given a Connext DDS 5.2.3 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]

6.6 Disabled Interfaces on Windows Systems

The creation of a *DomainParticipant* will fail if no interface is enabled and the **DiscoveryQosPolicy.multicast_receive_addresses** list (specified either programmatically, or through the **NDDS_DISCOVERY_PEERS** file or environment variable) contains a multicast address.

However, if **NDDS_DISCOVERY_PEERS** only contains unicast addresses, the *DomainParticipant* will be successfully created even if all the interfaces are disabled. The creation of a *DataReader* will fail if its TransportMulticastQosPolicy contains a UDPv4 or UPDv6 multicast address.

6.7 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]

6.8 Segmentation Fault when Creating DTLS DomainParticipants in Multiple Threads—Solaris and QNX Platforms Only

On Solaris and QNX platforms, the creation of DTLS-enabled *DomainParticipants* is not thread-safe and may lead to a segmentation fault in the function **RTIOsapiSemaphore_take()**. This issue has been resolved for Windows, Linux, and Android systems.

[RTI Issue ID COREPLG-264]

6.9 Typecodes Required for Request-Reply Communication Pattern

Typecodes are required when using the Request-Reply communication pattern. To use this pattern, do not use *rtiddsgen's* **-noTypeCode** flag. If typecodes are missing, the Requester will log an exception.

[RTI Issue ID REQREPLY-3]

6.10 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-5.2.3
```

6.11 Writer-side Filtering Functions Can be Invoked Even After Filter Unregistered

If you install a ContentFilter that implements the writer-side filtering APIs, Connext DDS can call those APIs even after the ContentFilter has been unregistered.

[RTI Issue ID CORE-5356]

6.12 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]

6.13 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]

6.14 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]

- DynamicData cannot handle a union with a discriminator that is set to a value which is not defined in the type.

  [RTI Issue ID CORE-3142]

- DynamicData may have problems resizing variable-size members that are >= 64k in size. In this case, the method (**set_X()** or **unbind_complex_member()**) will fail with the error: "sparsely stored member exceeds 65535 bytes." Note that it is not possible for a member of a sparse type to be >= 64k.

  [RTI Issue ID CORE-3177]

- 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]

- DynamicData does not support out-of-order assignment of members that are longer than 65,535 bytes. In this situation, the DynamicData API will report the following error:

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

  For example:

  ```
  struct MyStruct {
      string<131072> m1;
      string<131072> m2;
  };
  ```

With the above type, the following sequence of operations will fail because m2 is assigned before m1 and has a length greater than 65,535 characters.

```
str = DDS_String_alloc(131072);
memset(str, 'x', 131072);
str[131071]= 0;
DDS_DynamicData_set_string(
    data, "m2",  DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED, str);
DDS_DynamicData_set_string(
    data, "m1", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED, str);
```

If member m1 is assigned before m2, the sequence of operations will succeed.

[RTI Issue ID CORE-3791, Bug # 13745]

6.15 Known Issues in RTI Monitoring Library

6.15.1 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 work-around, 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).

6.15.2 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*.

6.15.3 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]

Chapter 7 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/).