RTI Connext DDS

Core Libraries

Release Notes

Version 5.2.0

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

This document includes the following:

For an overview of new features, please see the ***What's New*** document.

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.

Table 2.1 Supported Platforms for Connext DDS 5.2.0 lists the supported platforms. See the *RTI Connext DDS Core Libraries Platform Notes* for more information on each one.

Table 2.1 Supported Platforms for Connext DDS 5.2.0

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

Table 2.1 Supported Platforms for Connext DDS 5.2.0

Operating System	Version
Linux (Intel® CPU)	CentOS 5.4, 5.5, 6.0, 6.2 - 6.4 Red Hat® Enterprise Linux 4.0, 5.0-5.2, 5.4, 5.5, 6.0 - 6.5, 7 SUSE® Linux Enterprise Server 11 SP2, SP3 Ubuntu® Server 12.04 LTS, Ubuntu 14 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
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 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	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
VxWorks	VxWorks 6.7 and 6.8 using JamaicaVM 6.2.1 with gcc 4.1.2

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 Redistribution 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:

  http://www.microsoft.com/download/en/details.aspx?id=5555

- For x64 architectures:

  http://www.microsoft.com/download/en/details.aspx?id=14632

When Using Visual Studio 2012 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2012 Update 3 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 3 from this Microsoft website: http://www.microsoft.com/en-ca/download/details.aspx?id=30679

When Using Visual Studio 2013 — Redistributable Package Requirement

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

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

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

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.

Chapter 2 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 3 Compatibility

3.1 Wire Protocol Compatibility

3.1.1 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.

3.1.2 Release-Specific Information for Connext DDS 5.x

3.1.2.1 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.

3.1.3 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.

3.1.3.1 RTPS Versions

Connext DDS 4.5 and 5.x support RTPS 2.1. Some earlier releases (see Table 3.1 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 3.1 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

3.1.3.2 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.

3.1.3.3 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>
```

3.2 Code Compatibility

3.2.1 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.

3.2.2 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

3.2.2.1 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.

3.2.2.2 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.

3.2.2.3 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).

3.2.2.4 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.

3.2.2.5 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.

3.2.2.6 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*).

3.2.2.7 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.

3.2.2.8 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.

3.2.2.9 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)**

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

3.2.3.1 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_ 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.

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

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.

3.2.3.2 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.

3.3 Extensible Types Compatibility

3.3.1 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.

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

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*.

3.3.2 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.

3.4 ODBC Database Compatibility

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

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*.

3.5 Transport Compatibility

3.5.1 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**.

3.5.2 Transport Compatibility for Connext DDS 5.1.0

3.5.2.1 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.

3.5.2.2 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>
```

3.5.2.3 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 3.2 on the next page) and Shared Memory (Section Table 3.3 on the next page) show the new default transport settings.

Table 3.2 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 3.3 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

3.5.2.4 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]**.

3.5.3 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).

[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.

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.)

3.5.3.1 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: ""
```

3.5.3.2 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>
```

3.5.4 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 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.
```

[a]RTI Issue ID CORE-6777

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>
```

3.6 Other Compatibility Issues

3.6.1 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.

3.6.2 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.

3.6.3 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 4 What's Fixed in 5.2.0

This section describes bugs that have been fixed in this release.

4.1 Fixes Related to Application-Level Acknowledgement

4.1.1 Invalid Value for valid_response_data in AcknowledgmentInfo—.NET API Only

In the .NET API, the **valid_response_data** member in the AcknowledgmentInfo parameter of the DataWriterListener's **on_application_acknowledgment()** callback was always false. This problem has been resolved.

[RTI Issue ID CORE-6592]

4.1.2 REDAInlineList_addNodeToBackEA:!precondition Error Message when Using Application-Level Acknowledgment

When using the Connext DDS debug libraries, the application acknowledgment of samples out-of-order for keyed topics may have generated the following precondition error on the *DataWriter*:

```
REDAInlineList_addNodeToBackEA:!precondition
```

The error message did not impact functional behavior.

When using the release libraries, this issue may have caused DISPOSE or UNREGISTER samples to never be application-level acknowledged.

[RTI Issue ID CORE-6404]

4.1.3 Unable to Send Application-Level Acknowledgement Messages when Using Delegated Reliability

The user-specified response data of an explicit application-level acknowledgment was never removed when the *DataReader* was configured to use delegated reliability. Consequently, the application-level acknowledgment messages may have grown unbounded until they no longer fit in an RTPS packet. Users would have seen errors indicating that the application-level acknowledgment messages could not be sent. This problem has been resolved.

[RTI Issue ID CORE-6540]

4.1.4 Unexpected Memory Growth on DataWriter when Using Batching and Enabling Application-Level Acknowledgement

The memory associated with *DataWriters* using batching and enabling application-level acknowledgements may have grown unbounded over time if **writer_qos.resource_limits.max_samples** was set to UNLIMITED.

Notice that when setting **writer_qos.resource_limits.max_samples** to UNLIMITED, this issue was not reported as a memory leak because all the memory was reclaimed after the *DataWriter* was destroyed.

If *writer_qos.resource_limits.max_samples* was set to a finite number, you would have seen errors on the write operation after *writer_qos.resource_limits.max_samples* was exceeded.

This problem has been resolved.

[RTI Issue ID CORE-6750]

4.1.5 Unexpected Memory Growth when using DataReader's acknowledge_sample() Operation

Using the **acknowledge_sample()** operation to explicitly acknowledge samples in a DataReader resulted in unbounded memory growth, which could eventually exhaust the available memory.

A potential workaround was to use the **acknowledge_all()** operation instead.

This problem has been resolved.

[RTI Issue ID CORE-6277]

4.1.6 Unexpected Timeout Error in DataWriter’s wait_for_acknowledgments() when using Application-Level Acknowledgment

Calling **wait_for_acknowledgments()** on a VOLATILE *DataWriter* may have caused a DDS_RETCODE_TIMEOUT error, even if all the samples in the *DataWriter’s* history were acknowledged by the *DataReaders* that were alive when the samples were published.

This issue was a race condition that only occurred when new *DataReaders* appeared after the samples had been published.

The problem has been resolved.

[RTI Issue ID CORE-6670]

4.2 Fixes Related to Batching

4.2.1 Last value Per Instance Not Guaranteed when Using Batching and No Resources to Satisfy History Depth

If a *DataWriter* is configured with a History **kind** of KEEP_LAST, **max_samples** > **max_instances**, and **max_samples** < **(max_instances *** depth), the documentation[a] states that Connext DDS may discard samples of some other instance, as long as at least one sample remains for such an instance. (**max_ instances** and **max_samples** are members of the ResourceLimitsQosPolicy; **depth** is a member of the HistoryQosPolicy.)

In previous releases, when batching is enabled, Connext DDS may have discarded all the samples for an instance. This problem has been resolved.

[RTI Issue ID CORE-6140]

4.2.2 Unexpected Precondition Error when Using Batching on DataWriter

You may have seen the following precondition errors on *DataWriters* using the batching feature:

```
PRESWriterHistoryDriver_finalizeBatchSample:!precondition
```

The error only occurred when individual samples were invalidated in a batch out-of-order. This error may have prevented batches from being removed from the writer history. This problem has been resolved.

[RTI Usse ID CORE-6669]

4.3 Fixes Related to Conditions and Waitsets

4.3.1 Calling set_query_parameters() with Same Key-Only Expression in QueryCondition Caused Waitset not to be Triggered

In some circumstances, a call to **set_ query_ parameters ()** may have caused a key- only QueryCondition to no longer be triggered. In particular, this issue occurred for instances that had no samples to be read in the *DataReader* when **set_query_parameters()** was called. This problem has been resolved.

[a]This behavior is described in Section 6.3.8.1, Blocking During a write(), in the *RTI Connext DDS Core Libraries User's Manual.*

[RTI Issue ID CORE-6256]

4.3.2 Disposal of Remote DataWriter may have Caused DataReader's Keyed QueryCondition not to Trigger Anymore

Receiving a ‘dispose’ sample as the first sample for an instance may have caused the associated instance not to trigger the QueryCondition until there was a call to **set_query_parameters()**. This problem has been resolved.

[RTI Issue ID CORE-6318]

4.3.3 Memory Leak when Using QueryConditions on Unkeyed DataReader

Using a QueryCondition on unkeyed *DataReaders* may have caused a memory leak. The problem only occurred if the QueryCondition was created while there were samples in the *DataReader's* queue. In addition, this problem only affected the following language bindings:

- .NET
- Java
- DynamicData (regardless of the language)

This problem has been resolved.

[RTI Issue ID CORE-6734]

4.3.4 ReadCondition and Group Access Scope did not Work Correctly

When setting the PresentationQosPolicy’s **access_scope** to GROUP_PRESENTATION_QOS and using a Waitset to wait on a ReadCondition to read historical data, the Waitset may have waited longer than necessary because the ReadCondition was not triggered in a timely manner. This problem has been resolved.

[RTI Issue ID CORE-6237]

4.3.5 ReadCondition not Always Triggered when Samples Available after Losing Remote Participant Liveliness

A ReadCondition was not always properly triggered after loss of liveliness for a remote Participant. In particular, the ReadCondition may have not always triggered as expected in the presence of DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE samples in the *DataReader*. This problem has been resolved.

[RTI Issue ID CORE-6358]

4.3.6 WaitSet Operations get_conditions() and wait() were not Thread Safe

Calling the WaitSet operation **get_conditions()** from two different threads at the same time, or having one thread call **get_ conditions()** and another one call **wait()** may have resulted in undefined behavior. This problem has been resolved and these operations can be safely called concurrently.

Note: Calling **wait()** concurrently from multiple threads is not permitted and returns PRECONDITION_ NOT_MET (as indicated in the API Reference HTML documentation).

[RTI Issue ID CORE-6143]

4.4 Fixes Related to ContentFilters

4.4.1 ContentFilteredTopics may not have Correctly Filtered Some C++ Types with Inheritance

C++ types with some specific memory layouts, generated with rtiddsgen from IDL types that use inheritance, may have not been filtered correctly. This could have caused dropped samples that should have passed the filter. This problem has been resolved.

[RTI Issue ID CORE-6269]

4.4.2 Default Content Filter may have Failed to Filter Some Types with Optional Members–DynamicData, Java, and C# APIs Only

The SQL content filter may have produced undefined behavior (typically non-fatal errors or incorrect evaluation of the sample being filtered) when filtering some non-mutable types containing optional members. The problem affected content filters on IDL types in the Java and .NET APIs, and on DynamicData types in all APIs. IDL types in the C and C++ APIs were not affected. Only some type definitions with a specific layout caused the failure. This problem has been resolved.

[RTI Issue ID CORE-6680]

4.4.3 Filter-Expression Parsing Errors for Variable Names Starting with Reserved Word

Due to an issue in the SQL filter-expression parser, field names starting with a reserved keyword (e.g., ORdinal) were not parsed correctly. This problem has been resolved.

[RTI Issue ID CORE-3971]

4.4.4 Invalid filter_data when ContentFilter's writer_return_loan() Invoked

If a user registers a C Custom Filter that contains a pointer to a user-defined **writer_return_loan()** function, Connext DDS should call the user's function, passing in three parameters: **void *filter_data**, **void**

***writer_filter_data**, and **struct DDS_CookieSeq *cookies**.

In the previous release, the first parameter (**void *filter_data**) contained an invalid pointer. This could result in undefined behavior. This problem has been resolved.

[RTI Issue ID CORE-6201]

4.4.5 Memory Leak if a Custom Content Filter was not Unregistered

You are not required to unregister custom content filters before *DomainParticipant* destruction. However, if a custom content filter was not unregistered, there was a memory leak. This problem has been resolved. All still-registered custom content filters are now explicitly unregistered during *DomainParticipant* destruction, so there is no more memory leak.

[RTI Issue ID CORE-6390]

4.4.6 Memory Leak when Changing Expression Parameters on ContentFilteredTopic not Associated with any Reader

Applications leaked memory when changing the parameters of a ContentFilteredTopic after creating it but before creating a DataReader with it. The memory leak has been resolved.

[RTI Issue ID CORE-6156]

4.4.7 Possible Crash when Using ContentFilteredTopic and Builtin SQL Filter

If an error occurred while creating the SQL filter for a ContentFilteredTopic, there was a possibility of a crash. There are a number of reasons why the creation might fail, for example if there was a memory allocation failure. This problem has been resolved.

[RTI Issue ID CORE-6816]

4.4.8 Possible Incorrect Filter Results in 'AND' Expressions with Optional Members, Sequences, or Unions

A problem evaluating AND expressions in the SQL content filter caused the following kinds of comparisons, which should be false, to be mistakenly evaluated as true:

- Comparisons involving unset (null) optional members
- Comparisons involving sequence members above the actual length
- Comparisons involving unselected union members

For example, given the following IDL type:

```
struct A {
    string required_str;
    string optional_str; //@Optional
};
```

The expression "required_str = 'Hi' AND optional_str = 'Bye'" would evaluate as true when required_str = 'Hi' and optional_str = NULL.

Only AND expressions had this problem. Expressions such as "required_str = 'Hi' OR optional_str = 'Bye'" and "optional_str = 'Bye'" were not affected.

This problem has been resolved and all those kind of comparisons will evaluate to false as expected.

[RTI Issue ID CORE-6684]

4.4.9 Possible Segmentation Fault from ContentFilteredTopic Evaluation for Samples of Types with wstrings

The evaluation of a content filter may have caused a segmentation fault if all of these conditions were met:

- The *DataReader* was a DynamicDataReader, a *DataReader* in the Java API, or a *DataReader* explicitly configured to filter on serialized data (this last case is not common)
- The data type contained one or more wstrings
- The C memory layout of the data type and the wstring member had a specific alignment

This problem has been resolved and applications can now use wstrings normally.

[RTI Issue ID CORE-6287]

4.4.10 Segmentation Fault when Creating ContentFilteredTopic with Long Filter Expression

There may have been a segmentation fault when using ContentFilteredTopics. In particular, this issue may have occurred after creating a ContentFilteredTopic with an expression that resulted in a content-filter property larger than the maximum specified by DomainParticipantResourceLimitsQosPolicy's **contentfilter_property_max_length**. This problem has been resolved.

[RTI Issue ID CORE-5872]

4.4.11 Segmentation fault when Using SQL ContentFilteredTopic on Type with Large Maximum Serialized Size

Using a SQL ContentFilteredTopic on a Type whose maximum serialized size was greater than 2^32/2-1 may have caused a segmentation fault.

For example, this problem may have occurred when using a type with unbounded sequences for which the code generation was done with the -unboundedSupport option:

```
struct MyType {
    sequence<long> m1;
};
```

This problem has been resolved.

[RTI Issue ID CORE-6420]

4.4.12 Unexpected Memory Growth on DataWriter when Performing Writer-Side Filtering on Metadata Fields

Performing writer-side filtering on metadata fields using the built-in SQL filter may have led to unbounded memory growth when new *DataReaders* using ContentFilteredTopics on the metadata fields were created or destroyed.

This bug affected two wrapper APIs:

- Request-Reply API: The Replier's *DataReader* installs a ContentFilteredTopic on the metadata field **related_sample_identity**.
- Queuing API: The QueueProducer, QueueConsumer, QueueRequester, and QueueReplier *DataReaders* install a ContentFilteredTopic on the new metadata fields **related_source_guid** and **related_source_guid**.

Notice that this issue was not reported as a memory leak because all the memory was reclaimed after the affected *DataWriters* were destroyed. This problem has been resolved.

[RTI Issue ID CORE-6738]

4.5 Fixes Related to Discovery

4.5.1 Discovery Failed on Publisher Side if contentfilter_property_max_length not Long Enough to Deserialize Filter Information

Discovery failed if a publishing application set the **contentfilter_property_max_length** to a value too small for deserializing the filter information from a matching *DataReader* of a subscriber application. No communication would occur.

This situation has been solved by allowing discovery to occur in the publisher application. Since the filter information cannot be retrieved, writer-side filtering will not be available and the filtering will occur in the subscriber publication.

[RTI Issue ID CORE-5875]

4.5.2 Discovery Packets Sent over Unicast had Incorrect Priority

The **metatraffic_transport_priority** field of the DiscoveryQosPolicy was not being used to set the priority of discovery packets sent over unicast. (Instead, these packets used a priority of 0.) This problem has been resolved; now **metatraffic_transport_priority** sets the transport priority of all discovery packets, instead of just Heartbeat and Liveliness packets.

[RTI Issue ID CORE-6180]

4.5.3 DomainParticipantResourceLimitsQosPolicy's ignored_entity_allocation.max_count had no Effect

Connext DDS provides a way to ignore remote entities by invoking any of the following *DomainParticipant* operations: **ignore_participant()**, **ignore_publication()**, and **ignore_subscription()**. When an entity is ignored, Connext DDS adds it to an internal ‘ignore’ table whose resource limits are configured using the DomainParticipantResourceLimitsQosPolicy's **ignored_entity_allocation.max_count**.

There was an issue that prevented **ignored_entity_allocation.max_count** from being enforced. Consequently, allocated memory could grow to an unlimited size. This problem has been resolved.

[RTI Issue ID CORE-6176]

4.5.4 Memory Leak upon Remote Participant Discovery

If the DomainParticipantResourceLimitsQosPolicy’s **transport_info_list_max_length** was greater than zero, receiving remote Participant announcements may have triggered a memory leak. This problem has been resolved.

[RTI Issue ID CORE-6562]

4.5.5 Modified RTPS Wire Protocol Built-in Endpoint Mask to Match OMG Specification

The builtin endpoint mask, serialized as part of the RTPS Wire Protocol Participant Data, has been modified to match the OMG specification. In particular, the bits BUILTIN_ENDPOINT_PARTICIPANT_MESSAGE_DATA_WRITER and BUILTIN_ENDPOINT_PARTICIPANT_MESSAGE_DATA_READER are now set to notify other peers about the Liveliness *DataWriter* and *DataReader*.

[RTI Issue ID CORE-6452]

4.5.6 No Communication with Peer that was Removed, Re-Added

If a peer was removed by calling the DomainParticipant's **remove_peer()** operation and later re-added with **add_peer()**, communication with that peer did not always resume. This was a timing issue that only occurred in some cases. This problem has been resolved.

[RTI Issue ID CORE-6128]

4.5.7 ParticipantBuiltinTopicData was Missing Domain ID—.NET API Only

The field **domain_id** was missing from the ParticipantBuiltinTopicData type in the .NET API. This problem has been resolved; the **domain_id** has been added.

[RTI Issue ID CORE-6598]

4.5.8 Possible Segmentation Fault upon Failure Asserting Remote Reader

There may have been a segmentation fault upon discovery of a remote reader. This was only reproducible when there was a failure asserting the remote reader. This problem has been resolved.

[RTI Issue ID CORE-6702]

4.5.9 Potential Bus Error when Discovering Remote Participants with Properties Set—Solaris Platforms on 64-bit Sparc CPUs Only

Applications may have generated a bus error when discovering remote DomainParticipants if the PropertyQosPolicy was not empty. This only occurred on Solaris platforms using 64-bit Sparc CPUs. This problem has been resolved.

[RTI Issue ID CORE-6749]

4.5.10 Potential Unexpected Memory Growth when Asserting a Remote Reader

There may have been unexpected memory growth when a new remote reader was discovered. This memory growth would only have occurred if the remote reader provided a locator that the local writer could not reach.

This issue has been resolved.

[RTI Issue ID CORE-6701]

4.5.11 Unable to Set remote_participant_allocation with incremental_count of 0

If the incremental_count of **remote_participant_allocation** (in the DomainParticipantResourceLimits QoS policy) was set to 0, participant creation failed. This problem has been resolved.

[RTI Issue ID CORE-6556]

4.5.12 Unbounded Memory Growth after Failed Attempt to Ignore Participant

A failed attempt to ignore a remote Participant that had a DomainParticipantResourceLimitsQosPolicy **transport_info_list_max_length** greater than zero may have triggered unbounded memory growth of the transport info list. This problem has been resolved.

[RTI Issue ID CORE-6735]

4.5.13 Unexpected Memory Growth if Remote Endpoint Changed QoS that is Propagated with Built-in Topics

If a remote endpoint changed any QoS values that are propagated as part of the built-in topics, this resulted in unbounded memory growth in the matching local endpoint, which could eventually exhaust the available memory. This only happened if the remote endpoint had a non-NULL value for the EntityQosPolicy's **name** or **role_name** fields. This problem has been resolved.

[RTI Issue ID CORE-6724]

4.5.14 Unexpected Memory Growth when Discovering Remote Participants

When a remote *DomainParticipant* was deleted from the discovery tables of the local *DomainParticipant*, not all the memory associated with the remote *DomainParticipant* was released. This could have led to unbounded memory growth if a lot of remote *DomainParticipants* were discovered and disappeared while the local *DomainParticipant* was running.

Notice that all memory was properly reclaimed when the local *DomainParticipant* was deleted. This is why tools such as valgrind did not report this issue as a memory leak.

This problem has been resolved.

[RTI Issue ID CORE-5589]

4.5.15 Unexpected Memory Growth when Discovering Remote Participants with Role Name

The **participant_name.role_nam**e of a discovered remote *DomainParticipant* was not released when the remote *DomainParticipant* was destroyed or disappeared. This could have led to unbounded memory growth if a lot of remote *DomainParticipants* with **participant_name.role_name** set were discovered and disappeared while the local *DomainParticipant* was running.

Notice that all memory was properly reclaimed when the local *DomainParticipant* was deleted. This is why tools such as valgrind did not report this issue as a memory leak.

This problem has been resolved.

[RTI Issue ID CORE-6723]

4.5.16 Unexpected "PRESPsService_ destroyLocalEndpointWithCursor:!remove remote endpoint" Warning Message

You may have seen the following warning message when using **DomainParticipant::ignore_participant()** in a *DomainParticipant* to ignore itself:

```
PRESPsService_destroyLocalEndpointWithCursor:!remove remote endpoint
```

This warning message was misleading and it has been removed.

[RTI Issue ID CORE-6802]

4.5.17 Unnecessary Participant Announcements after Participant Leaves the System

Under certain circumstances, a participant may have continued to send Participant Announcements to destinations associated with remote participants that had already left the system, even if those destinations were not part of the initial discovery peers.

In particular, this issue may have occurred for multicast or unicast destinations when the DiscoveryQosPolicy's **accept_unknown_peers** was set to true (the default).

This problem has been resolved.

[RTI Issue ID CORE-6555]

4.6 Fixes Related to Distributed Logger

4.6.1 Default Filter Level for Java

The default setting for Distributed Logger's filter level was TRACE, when it should have been INFO. This problem has been resolved.

[RTI Issue ID DISTLOG-137]

4.6.2 Deletion of C/C++ Distributed Logger's DataWriters not Propagated via Discovery

The deletion of the Distributed Logger's DataWriters after invoking the API **RTI_DLDistLogger::finalizeInstance()** was not propagated via discovery. For example, if you ran *RTI Monito*r or *rtiddsspy*, these tools still showed the *DataWriters* as alive. This issue only occurred with the C and C++ APIs.

These *DataWriters* were affected:

- Administration Command Response *DataWriter*
- Administration State *DataWriter*
- Log Message *DataWriter*

This problem has been resolved.

[RTI Issue ID DISTLOG-133]

4.6.3 Possible Warning when Compiling Distributed Logger Examples on UNIX-Based Systems

You may have seen a warning when compiling Distributed Logger C and C++ examples on UNIX-based systems. This problem has been resolved.

[RTI Issue ID DISTLOG-138]

4.7 Fixes Related to Durable Writer History

4.7.1 DataWriter Configured to Use Durable Writer History did not Communicate with Persistence Service

A *DataWriter* using Durable Writer History could not communicate with Persistence Service. You may have seen the following errors on the DataWriter side:

```
WriterHistoryRemoteReaderManager_assertRemoteReader:!precondition
[FATAL] WriterHistoryOdbcPlugin_assertRemoteReader:!assert virtual read
PRESWriterHistoryDriver_assertRemoteReader:!assert_remote_reader
PRESPsService_linkToRemoteReader:!whDriver->assertRemoteReader
PRESPsService_onLinkToRemoteEndpointEvent:!create linkToRemoteReader
```

This problem has been resolved.

[RTI Issue ID CORE-6426]

4.7.2 Long Delay Receiving Data from DataWriter with Durable Writer History

A late-joiner *DataReader* may have taken a long time to receive all historical data from a durable DataWriter, even if there were few historical samples. For example, assume the durable *DataWriter* is configured to keep the last sample for each instance (last-value cache). Consider the following sequence of samples:

S1 (Instance 1), S2 (Instance 2), ..., S1000000 (Instance 2)

In this case, the durable *DataWriter* will keep only two samples for late-joiners: S1 (Instance 1) and S1000000 (Instance 2).

The problem was that when a late-joiner started up, it received S1 from the *DataWriter* immediately, but it took a while to receive S1000000. The durable *DataWriter* did not manage sample GAP messages efficiently; it generated significant RTPS GAP traffic to declare that it did not have samples S2 to S999999. This problem has been resolved.

[RTI Issue ID CORE-6308]

4.7.3 Samples not Acknowledged after Restarting DataWriter in Some Cases

In certain scenarios, some samples remained unacknowledged after restarting the *DataWriter*. This problem only occurred when all of the following conditions were true:

- The *DataWriter* was configured with application acknowledgments enabled
- The *DataWriter* was configured with durable subscriptions enabled
- The *DataWriter* used the ODBC sample history
- The sample history was restored from the database

In that situation, samples in the writer that met all the criteria to be considered acknowledged by the *DataReaders* may not have been marked as such. This caused unnecessary data retransmissions and prevented the release of those samples from the database if needed.

[RTI Issue ID CORE-6580]

4.7.4 Unexpected Memory Growth when using Durable Writer History and Keyed Topics

In previous releases, there may have been unbounded memory growth when using Durable Writer History on a keyed *DataWriter*. This problem has been resolved.

[RTI Issue ID CORE-6784]

4.8 Fixes Related to DynamicData

4.8.1 Cannot Create DynamicDataTypeSupport Object with Typecodes Larger than 64k—Java API Only

Attempting to create a DynamicDataTypeSupport object with a typecode with a size larger than 64k while using the Java API would fail. This problem has been resolved. There is no longer a limit imposed on the size of the typecode that can be used to created a DynamicDataTypeSupport object in the Java API.

[RTI Issue ID CORE-6427]

4.8.2 DDS_DynamicData_copy() could Create Incorrect DynamicData Object

Under a few circumstances, a call to **DDS_DynamicData_copy()** created a misaligned DynamicData object that could not be used. This problem has been resolved.

[RTI Issue ID CORE-6522]

4.8.3 DDS::DYNAMIC_DATA_PROPERTY_DEFAULT, DDS::DYNAMIC_ DATA_TYPE_PROPERTY_DEFAULT Missing from C++ API

Previous releases were missing the definitions for DDS::DYNAMIC_DATA_PROPERTY_DEFAULT, DDS::DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT constants. This issue has been resolved.

[RTI Issue ID CORE-6349]

4.8.4 Failure when Printing DynamicData Object with Type that used Inheritance

Calling print on a DynamicData object whose underlying type used inheritance caused the print function to print incorrect values for the members of the derived type or to fail with the error code DDS_ RETCODE_ERROR. This problem has been resolved.

[RTI Issue ID CORE-6455]

4.8.5 Inconsistencies between DynamicData's set_char() and get_char() -- Java API Only

The value that was set for a char using any of the following APIs in Java was not always the value that was returned when using the DynamicData APIs to retrieve the set values:

- **DynamicData.set_char()**
- **DynamicData.set_char_array()**

- **DynamicData.set_char_seq()**

The problem was due to the fact that a char in Java is a 16-bit, unsigned value, but the OMG IDL specification says that the char data type is 8 bits. Therefore, only char values between 0 and 255 are supported in Java. The problem has been resolved and the value set will be the same value that is returned from the 'get' APIs as long as the values that are set fall within the valid range [0, 255].

[RTI Issue ID CORE-6639]

4.8.6 Incorrect Instance Handles on DynamicData DataReaders Subscribing to Topics with Mutable Types

The instance handle obtained by invoking the DynamicData *DataReader's* **lookup_instance()** operation, or by accessing the field **SampleInfo::instance_handle**, may have been incorrect when DynamicData *DataReaders* subscribed to a *Topic* using mutable types with keys containing non-primitive fields.

This issue affected users who tried to republish the samples received by a DynamicData *DataReader* using the instance handles provided by the same DynamicData *DataReader*. The issue also affected *RTI Routing Service*.

The consequence of using the instance handles provided by DynamicData *DataReaders* to publish new samples is that the same instance may have been identified by two different instance handles in a DDS domain. This would cause correctness issues when applying per-instance QoS policies, such as the HistoryQosPolicy, DeadlineQosPolicy, and others.

The problem has been resolved.

[RTI Issue ID CORE-6674]

4.8.7 Possible Crash when Calling DDS_DynamicData_equals() if Types Contained Arrays—C and C++ APIs Only

A rare crash may have occurred when calling **DDS_DynamicData_equals()** on samples of a type that contained arrays. This problem has been resolved.

[RTI Issue ID CORE-6528]

4.8.8 Possible Crash when Calling DynamicData get_char_seq() if Sequence Passed In was Too Small—Java API Only

When using the DynamicData **get_char_seq()** method in Java, if the sequence used to retrieve the value of the char sequence was not large enough to hold the returned sequence, this caused a crash. This issue has been resolved.

[RTI Issue ID CORE-6632]

4.8.9 Possible Memory Leak when using DynamicData with Java API

There was a memory leak in applications that used the Java API to create DynamicData. This problem has been resolved.

[RTI Issue ID CORE-6282]

4.8.10 Potential Failure Writing DynamicData Object with Unset Optional Primitive Members

A write operation on a DynamicDataWriter with a type containing optional primitive members may have failed if the written sample had unset optional primitive members. This problem has been resolved.

[RTI Issue ID CORE-6776]

4.8.11 Potential Serialization Errors on DynamicData DataWriters Publishing Types Containing Wide Strings

Writing a sample with a DynamicData *DataWriter* may have failed in some cases if the associated type contained members that were wide strings. For example:

```
struct MyStruct {
    wstring<100> m1;
};
```

This problem has been resolved.

[RTI Issue ID CORE-6525]

4.8.12 Segmentation Fault when Setting pool_buffer_max_size to Zero in DataWriter using Dynamic Data

Setting the *DataWriter* property **dds.data_writer.history.memory_manager.fast_pool.pool_buffer_ max_size** to zero when the *DataWriter* was using Dynamic Data may have caused a segmentation fault. The problem has been resolved.

[RTI Issue ID CORE-6675]

4.8.13 Some DynamicData Serialization Properties Missing–Java and .NET APIs Only

The DynamicData serialization properties **min_size_serialized** and **trim_to_size** under DDS_DynamicDataProperty_t::serialization were not present in the Java and .NET APIs; therefore they could not be configured.

For more information on these properties, see Chapter 20, Sample-Data and Instance-Data Memory Management, in the *RTI Connext DDS Core Libraries User's Manual*.

This problem has been resolved.

[RTI Issue ID CORE-6531]

4.8.14 Some Operations on DynamicData Objects with Optional Members Failed

The DDS_DynamicData's **unbind_complex_member()** or **set_complex_member()** operations may have failed if the DynamicData object had unset optional members. This problem has been resolved.

[RTI Issue ID CORE-6594]

4.8.15 Some TypeCode Information Lost when Using DynamicData–Java API Only

When using DynamicDataTypeSupport in the Java API, some of the TypeCode information was lost. Specifically, information about user-specified member IDs and optional members was lost, possibly causing communication between Java applications and other language (C/C++/.NET) applications to fail due to mismatched TypeCodes. This problem has been resolved.

[RTI Issue ID CORE-6177]

4.8.16 Unset, Complex, Mutable Optional Members not Correctly Serialized using DynamicData

When sending a type with an unset optional member that was a struct with mutable extensibility, the serialization to CDR was incorrect, causing incorrect values to be deserialized on the receiving side. This problem has been resolved.

[RTI Issue ID CORE-6498]

4.8.17 Unset, Complex, Optional Members not Correctly Printed in DynamicData

Calling the print function on a DynamicData object with an unset, complex, optional member would print incorrect values for all members following the unset member in the type. This problem has been resolved.

[RTI Issue ID CORE-6497]

4.8.18 Wide String Support was Missing for DynamicData in Java API

The Java API lacked support for wide strings in the DynamicData type. The **set_string()** and **get_string()** operations on wide string members failed with the following error:

```
DDS_DynamicData_set_wstring:type mismatch for field wstr (id=0)
```

where 'wstr' is the name of the wide string member.

Support for wide strings has been added. To map a Java String to a wide string, create a member of kind WSTRING and use the existing **set_string()** and **get_string()** operations.

A wide character may contain more information than a String character. In this case, obtaining a value with **get_string()** may result in a loss of information since each wide string character is truncated to fit a Java String character.

For more information, see to API Reference HTML documentation.

[RTI Issue ID CORE-6235]

4.9 Fixes Related to Instance Management

4.9.1 Errors after Reregistering an Instance and Exceeding max_total_ instances

In some cases, after bringing a deleted instance back to the ALIVE instance state and then exceeding the DataReaderResourceLimit **max_total_instances**, you may have seen incorrect behavior and errors such as:

```
"PRESCstReaderCollator_addInstanceEntry: unexpected error".
```

This problem has been resolved.

[RTI Issue ID CORE-6577]

4.9.2 FooDataWriter::get_key_value() Reported Incorrect Error when Instance did not Exist

If the instance handle passed to **FooDataWriter::get_key_value()** was not registered, this function reported DDS_RETCODE_ERROR instead of DDS_RETCODE_BAD_PARAMETER. This problem has been resolved. Now the function returns the more specific DDS_RETCODE_BAD_PARAMETER.

[RTI Issue ID CORE-6096]

4.9.3 Invalid Instance Handle Provided by on_instance_replaced() Callback if Replacement of Empty Instances was Enabled

In certain circumstances, the instance handle provided by the *DataWriter*'s **on_instance_replaced()** callback was invalid. In particular, this issue was triggered when the replacement of empty instances in the *DataWriter* was enabled (that is, if the **replace_empty_instances** member of the DataWriter-ResourceLimitsQosPolicy was true (which is not the default setting)). This problem has been resolved.

[RTI Issue ID CORE-6284]

4.9.4 Possible Segmentation Fault or Precondition Error when Reading Samples of Previously Disposed Instance

When reading keyed samples of an instance that was previously disposed, a *DataReader* may have experienced a segmentation fault in the function **PRESCstReaderCollator_removeNotAliveSample()** when using the release libraries, or the following error when using the debug libraries:

```
REDAInlineList_removeNodeEA:!precondition: list ==
((void *)0) || node == ((void *)0) || node->inlineList != list
```

This issue has been resolved.

[RTI Issue ID CORE-6261]

4.10 Fixes Related to Linking

4.10.1 Failure when Linking with Other Libraries that Contain Expat-Based Parsers

In some scenarios, the RTI XML parser failed to initialize when linking with third-party libraries that included the expat library, due a symbol conflict. This issue has been resolved by removing the symbol conflicts.

[RTI Issue ID CORE-6043]

4.10.2 Linker Errors in C Request-Reply API

Several linker errors (undefined and/or duplicate symbols) may have occurred when using the C API of the Request-Reply communication pattern when linking user-defined Requesters and Repliers.

For example, defining two Requesters or two Repliers of different types in the same object file would have caused several duplicate-symbol errors. There were also some undefined symbols such as <MyReplier>_return_loan (where <MyReplier> was the name given to a requester declared by using the macro RTI_CONNEXT_REQUESTER_DECL and defined by including the file **connext_c/generic/connext_c_requestreply_TReqTRepReplier.gen**).

These problems have been resolved. Now it is possible to declare more than one Requester or Replier in the same object file. The missing symbols have also been fixed.

[RTI Issue ID REQREPLY-18]

4.11 Fixes Related to Request-Reply Communication

4.11.1 Requester's receiveReply() and Similar APIs Returned Immediately when Infinite Specified—C# API Only

C# Requester APIs that allow waiting for the reception of samples with a specified timeout would return immediately if an infinite timeout was specified. This problem has been resolved.

[RTI Issue ID REQREPLY-15]

4.11.2 Requesters may not have Received Replies of Some Types with Optional Members—Java and C# APIs

Some immutable reply types (for instance, extensible or final types) containing optional members and a specific memory layout may have caused a non-fatal error. This may have occurred when sending or receiving a reply sample. The non-fatal error prevented the application from receiving the sample. This problem has been resolved.

[RTI Issue ID REQREPLY-23]

4.11.3 Requester or Replier Write Timeout Caused Exception of Wrong Class—C++ API Only

If the **Requester::send_request()** or **Replier::send_reply()** operations failed because the write operation timed out, they threw an AlreadyDeletedException instead of a TimeoutException.

This problem has been resolved; now they throw TimeoutException.

[RTI Issue ID REQREPLY-27]

4.12 Fixes Related to Transports

4.12.1 Bus Error when Using Shared Memory Transport on Sparc Solaris 64-bit Platforms

Connext DDS may have generated a bus error when using the shared memory transport on Sparc Solaris 64-bit platforms. This problem has been resolved. Notice that by doing this, out-the-box backward compatibility with previous releases on Sparc Solaris platforms (32-bit and 64-bit) using shared memory transports is broken.

[RTI Issue ID CORE-6777]

4.12.2 Data Corruption when Sending Large Data at a High Rate from a TCP Transport Plugin with Windows IOCP

There may have been data corruption when sending large data at a high rate from a TCP Transport plugin with Windows IOCP. You may have seen the following message from the remote TCP Transport plugin:

```
NDDS_Transport_TCPv4_Connection_nonBlockingRead:wrong msg signature:
recv=00010000, expected=dd54dd55
```

This problem has been resolved.

[RTI Issue ID COREPLG-313]

4.12.3 DataWriter's Transport Priority had no Effect in TCP Transport Plugin Server Sockets

The *DataWriter*'s Transport Priority had no effect on TCP Transport Plugin server sockets. This problem has been resolved.

[RTI Issue ID COREPLG-298]

4.12.4 DomainParticipant's add_peer and remove_peer not Supported by TCP Transport

The TCP transport plugin did not support the DomainParticipant's **add_peer()** and **remove_peer()** operations, resulting in outstanding TCP connections that were not closed. This problem has been resolved.

[RTI Issue ID COREPLG-203]

4.12.5 Duplicated Transaction ID in TCP Transport Plugin Control Messages–Windows Platforms Only

There was an issue that may have caused the TCP Transport plugin to send the same transaction ID in different control message exchanges. Consequently, this may have caused errors establishing new connections. This issue, which only affected Windows platforms, has been resolved.

[RTI Issue ID COREPLG-341]

4.12.6 Error Establishing TCP Transport Client Connection could Cause Communication not *to* Recover

There was an issue that may have caused communication between a TCP transport plugin client and server not to recover. In particular, this issue may have occurred when a client dropped a connection before it was fully established.

To address this issue, the TCP transport plugin now includes a new server connection negotiation timeout feature. This feature will close server connections that have not completed the TCP transport negotiation after a time period specified by the new TCP transport property, **server_connection_negotiation_timeout**. By default, this property is set to 10 seconds.

[RTI Issue ID COREPLG-311]

4.12.7 Failure Enabling DomainParticipant when using UDPv6 Transport in Some Cases

When using the built-in UDPv6 transport plugin, inclusion of a UDPv4 multicast address without an alias (e.g., 239.255.0.1) in NDDS_DISCOVERY_PEERS (file or environment variable) or in the **multicast_receive_addresses** list (in the DiscoveryQosPolicy) may have resulted in a failure during when enabling a DomainParticipant. The following error may have been reported:

```
[D0000|ENABLE]DDS_DomainParticipant_enableI:Automatic participant index
failed to initialize. PLEASE VERIFY CONSISTENT TRANSPORT / DISCOVERY
CONFIGURATION.
[NOTE: On Windows, verify at least one network interface is enabled.]
DDSDomainParticipant_impl::createI:ERROR: Failed to auto-enable entity
DomainParticipantFactory_impl::create_participant():!create failure
creating participant
create_participant error
```

This problem has been resolved.

[RTI Issue ID CORE-2002]

4.12.8 Host Improperly Identified when no IPv4 Interfaces Available

The host identity was not properly determined when there were no IPv4 interfaces available. This typically manifested when only IPv6 interfaces were available on the host. This problem has been addressed.

[RTI Issue ID MONITOR-170]

4.12.9 Logging Level for TCP Transport Connect Errors Changed from Exception to Warning

When a TCP Transport plugin Client disconnected from the Server (for example, after a cable disconnection), the resulting error messages were too verbose. In this release, the logging verbosity for those errors has been changed from Exception to Warning.

[RTI Issue ID COREPLG-307]

4.12.10 Memory Issue upon TCP Transport Plugin Server Disconnection

There was a race condition that may have provoked a segmentation fault in the TCP Transport Plugin server upon a client disconnection. As a consequence of this race condition, the TCP Transport plugin server may have also run into unbounded memory growth. This issue has been resolved.

[RTI Issue ID COREPLG-320]

4.12.11 Memory Leak when using UDPv4 Transport Plugin on VxWorks Architectures

On a VxWorks system, a call to **inet_ntoa()** allocates memory. The Connext DDS core did not properly release the allocated memory, which caused a memory leak. This problem has been resolved; allocated memory is now properly released.

[RTI Issue ID CORE-6368]

4.12.12 Memory Leak in Applications using TCP Transport in Asymmetric Mode

There was a memory leak in applications that used the TCP transport in asymmetric mode (**server_bind_port** = 0). The size of the memory leak depended on the number of TCP connections opened by the transport and the value of the transport property, **parent.message_size_max**. This problem has been resolved.

[RTI Issue ID COREPLG-96]

4.12.13 Memory Issue upon TCP Transport Plugin Server Disconnection

There was a race condition that may have provoked a segmentation fault in the TCP Transport Plugin server upon a client disconnection. As a consequence of this race condition, the TCP Transport plugin server may have also run into unbounded memory growth. This issue has been resolved.

[RTI Issue ID COREPLG-320]

4.12.14 Message Deserialization Errors in TCP Transport Plugin

There was an issue that may have caused errors when deserializing control messages sent by the TCP Transport plugin. This problem has been resolved.

[RTI Issue ID COREPLG-297]

4.12.15 Non RTPS-Compliant UDPv6 Locator kind

Connext DDS 5.1.0 and earlier releases used a UDPv6 locator kind that was not compliant with the value provided by the RTPS specification. The value used in Connext DDS 5.1.0 was 5, while the RTPS specification specifies a value of 2. Because of this issue, Connext DDS could not interoperate with other DDS vendors over UDPv6. This problem has been resolved. Notice that by doing this, out-the-box backward compatibility with previous releases using both the UDPv6 and SHMEM transports is broken. See Transport Compatibility for Connext DDS 5.2.0 (Section 3.5.3 on page 20) for additional details.

[RTI Issue ID CORE-5788]

4.12.16 Participant Creation Failed if All Enabled Network Interfaces were IPv6–Non-Linux Systems Only

To determine part of a *DomainParticipant's* GUID, Connext DDS will, by default, use the host ID of one enabled, non-loopback IPv4 network interface. If all the interfaces on a machine were configured with IPv6 only, the participant creation failed.

Applications can alternatively use the MAC address to determine the GUID (**DDS::WireProtocolQosPolicy::rtps_auto_id_kind** in the DomainParticipantQos). However, this mechanism also required at least one enabled, non-loopback IPv4 interface. The only way to work around this situation was to manually set the host ID (via the **DDS::WireProtocolQosPolicy::rtps_host_id**).

This problem was already resolved for Linux-systems. In this release, the problem is also resolved for non-Linux systems. Now all the options to assign the GUID work in systems where only IPv6 interfaces are available, although using the MAC address is recommended to increase uniqueness.

[RTI Issue ID CORE-6160]

4.12.17 Participant Creation Failed if Shared Memory Resources could not be Allocated

If shared memory resources could not be allocated, participation creation failed.

This problem has been resolved. Now if this situation occurs, the participant will be successfully created and will use other transports that have been installed, and the shared memory transport will be disabled.

[RTI Issue ID CORE-6518]

4.12.18 Participant Creation Failed when All Enabled Network Interfaces were IPv6

To determine part of a *DomainParticipant*'s GUID, Connext DDS uses by default the host ID of one enabled, non-loopback IPv4 network interface. If all the interfaces on a machine were configured with IPv6 only, the participant creation failed.

Applications can alternatively use the MAC address to determine the GUID (**DDS::WireProtocolQosPolicy::rtps_auto_id_kind** in DomainParticipantQos). However, this mechanism also required at least one enabled, non-loopback IPv4 interface.

The only way to work around this situation was to manually set the host id (via (**DDS::WireProtocolQosPolicy::rtps_host_id**).

This problem has been resolved. Now all the options to assign the GUID work in systems where only IPv6 interfaces are available, although using the MAC address is recommended to increase uniqueness.

[RTI Issue ID CORE-5955 and CORE-6160]

4.12.19 Participant may have Hung on Shutdown if Multicast Enabled or in Presence of Some Firewalls/Antivirus

A *DomainParticipant* may have hung on shutdown under one or a combination of the following scenarios:

- The *DomainParticipant* and/or its *Entities* received data over multicast
- The system on which the *DomainParticipant* ran used a firewall

This problem has been resolved.

[RTI Issue ID CORE-5954]

4.12.20 Port Reservation did not Account for All Enabled Transports

In previous releases, RTPS port reservation incorrectly used the DiscoveryQosPolicy's **enabled_transports** configuration for reserving user data ports (instead of using the TransportUnicastQosPolicy). As a consequence, two applications running on the same machine may have failed to open receive ports.

When this issue was triggered, the following messages (or similar ones) were printed:

```
COMMENDLocalReaderRO_init:!create unicast entryPort
COMMENDSrReaderService_createReader:!init ro
PRESPsService_enableLocalEndpointWithCursor:!create Reader in srrService
PRESPsService_enableLocalEndpoint:!enable local endpoint
```

This problem has been resolved.

[RTI Issue ID CORE-6464]

4.12.21 Possible Inconsistent State in the TCP Transport Plugin after Remote Client Reconnection

In some cases when reconnecting a TCP Transport plugin Client, the TCP Transport plugin Server may have ended up in an inconsistent state. Consequently, it may have taken longer than expected to to reestablish communication. This problem has been resolved.

[RTI Issue ID COREPLG-299]

4.12.22 Possible Segmentation Fault due to Race Condition in DTLS Transport

DomainParticipants communicating with each other using the DTLS transport may have experienced occasional segmentation faults in the function **SSL_do_handshake()**. This issue was due to a race condition and has been resolved.

[RTI Issue ID COREPLG-250]

4.12.23 Possible Segmentation Fault during Clean Up of TCP Transport Plugin Allocated Resources

Under certain circumstances, cleaning up resources in the TCP Transport plugin may have caused a segmentation fault. For example, this issue may have been triggered during TCP Transport plugin destruction or after the removal of a remote endpoint. This problem has been resolved.

[RTI Issue ID COREPLG-185]

4.12.24 Possible Segmentation Fault if TCP Transport Plugin Creation Failed

Under certain conditions, a failure in the creation of the TCP transport plugin could have caused a segmentation fault. This problem has been resolved.

[RTI Issue ID COREPLG-245]

4.12.25 Possible Segmentation Fault on Outgoing Traffic Connection Initialization

Under certain circumstances, a failure when initializing resources associated with the TCP Transport plugin's outgoing traffic may have provoked a segmentation fault. In particular, this issue may have been triggered if there was a failure opening a connection in a TCP Client. It may also have been triggered after a failure allocating the queue used by a TCP Server running with **force_asynchronous_send** enabled. This problem has been resolved content.

[RTI Issue ID COREPLG-289]

4.12.26 Possible Segmentation Fault on TCP Transport Plugin Shutdown

In the 5.1.0.38 release, a race condition may have lead to a segmentation fault. In particular, this issue may have been triggered during TCP Transport plugin shutdown. This issue has been resolved.

[RTI Issue ID COREPLG-294]

4.12.27 Potential Segmentation Fault when Outstanding Asynchronous Writes Exceeded write_buffer_allocation Limits

There was a potential segmentation fault when using TCP Transport plugin with Windows IOCP socket monitoring kind. In particular, this issue may have triggered when the number of outstanding asynchronous writes exceeded the limit set by the property **write_buffer_allocation.max_count**. This issue only affected TCPv4_LAN and TCPv4_WAN modes; it did not affect TLS-over-TCP modes. This problem has been resolved.

[RTI Issue ID COREPLG-354]

4.12.28 Potential Data Corruption when Using TCP Transport Plugin with force_asynchronous_send Enabled

There may have been data corruption if you enabled the TCP Transport plugin's **force_asynchronous_send** property while running in LAN or WAN-symmetric mode. This problem has been resolved.

[RTI Issue ID COREPLG-316]

4.12.29 RTPS Auto ID from MAC did not Use MAC Address--Darwin and QNX Systems Only

There was an issue that prevented generation of the GUID from the MAC address on Darwin and QNX systems. In particular, when the application set the WireProtocolQosPolicy's **rtps_auto_id_kind** to RTPS_AUTO_ID_FROM_MAC, the middleware used to retrieve an incorrect value for the GUID that incidentally matched the IP address and some garbage instead of the MAC address.

This problem has been resolved.

[RTI Issue ID CORE-6165]

4.12.30 Segmentation Fault on TCP Transport if TCP Client Connection Failed to Connect

Under certain circumstances, a failed connection from a TCP Transport plugin client connection may have resulted in a segmentation fault. In particular, this may have been triggered as a result of disabling all the interfaces on the system while running the TCP Transport plugin. This problem has been resolved.

[RTI Issue ID COREPLG-278]

4.12.31 Segmentation Fault when Creating DTLS DomainParticipants in Multiple Threads

The creation of DTLS-enabled DomainParticipants was not thread-safe and may have led 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]

4.12.32 Sockets not Completely Closed when Using TCP Transport Plugin with TLS Support

Sockets were not completely closed after a connection was disconnected. This issue only affected the TCP transport plugin connections when using RTI TLS Support.

[RTI Issue ID COREPLG-331]

4.12.33 TCP Connections for Server Outgoing Traffic may not have Reopened After Short TCP Client Disconnection

Under certain circumstances, a short disconnection of a TCP Transport plugin client from the server may have caused communication not to recover. In particular, the connections associated with the traffic flowing from the TCP server to the TCP client may have never been restablished. This problem has been resolved.

[RTI Issue ID COREPLG-279]

4.12.34 TCP Transport Communication Did not Recover after Client Disconnected

There was a timing issue that may have prevented the TCP Transport plugin Client from reestablishing communication after a disconnection. This problem has been resolved.

[RTI Issue ID COREPLG-306]

4.12.35 TCP Transport's enable_keep_alive Property had no Effect on Server Sockets

When the TCP Transport plugin's **enable_keep_alive** property was set to 1, the sockets created by the TCP server did not enable the TCP "keep alive" socket option. This issue did not affect TCP client sockets. This problem has been resolved.

[RTI Issue ID COREPLG-254]

4.12.36 TCP Transport's enable_keep_alive Property had no Effect on Server Sockets when Using TLS Support

When using TLS Support with the TCP transport, and the **enable_keep_alive** property was set to 1, the sockets created by the TCP server did not enable the TCP “keep alive” socket option. This issue did not affect TCP client sockets. This problem has been resolved.

[RTI Issue ID COREPLG-273]

4.12.37 TCP Transport’s ‘keep_alive_*’ Properties had no Effect–Linux Platforms Only

When the TCP Transport plugin’s **keep_alive_time**, **keep_alive_interval**, and **keep_alive_retry_count** properties (only supported on Linux platforms) were set, they were not properly applied to TCP sockets. This problem has been resolved.

[RTI Issue ID COREPLG-255]

4.12.38 TCP Transport’s ‘keep_alive_*’ Properties had no Effect when Using TLS Support–Linux Platforms Only

When using TLS Support with the TCP transport, and the TCP transport’s **keep_alive_time**, **keep_alive_interval**, and **keep_alive_retry_count** properties (only supported on Linux platforms) were set, they were not properly applied to TCP sockets. This problem has been resolved.

[RTI Issue ID COREPLG-274]

4.12.39 TCP Transport Plugin's Connection Liveliness Stopped Working

In the 5.1.0.38 release, under certain circumstances the connection-liveliness feature may have stopped working after a random amount of time. This problem has been resolved.

[RTI Issue ID COREPLG-308]

4.12.40 TCP Transport Plugin Opened UDPv4 Sockets when Running in WAN Mode

In previous releases, the TCP transport plugin created unnecessary UDPv4 sockets when running in WAN mode. This problem has been resolved.

[RTI Issue ID COREPLG-174]

4.12.41 TCP Transport Plugin did not Send Large Samples when Using TLS, 'force_asynchronous_send,' and Windows IOCP

In the 5.1.0.5 release, when used with TLS Support, the TCP Transport plugin could not send large messages if the Windows IOCP **socket_monitoring_kind** and **force_asynchronous_send** properties were both enabled. This problem has been resolved.

[RTI Issue ID COREPLG-312]

4.12.42 TCP Transport Plugin Hangs when using Transport Priority

The TCP Transport plugin would hang when the **transport_priority_mask** property was set to a value higher than 0x7fffffff. This problem has been resolved.

[RTI Issue ID COREPLG-322]

4.12.43 TCP Transport Plugin More Robust Against a Failure in SSL Handshake

When using TLS support, the TCP Transport plugin would shut down upon a failure in the SSL handshake. In this release, the TCP Transport plugin is more robust. In particular, if the plugin fails to complete the SSL handshake for a newly accepted connection, it will print an error message and close the associated connection, but it will not trigger a shutdown.

[RTI Issue ID COREPLG-342]

4.12.44 TCP Transport Plugin Reconnection Problem after Disconnecting Network Interface–Linux Systems Only

A disconnection of the network interface (e.g., unplugging the network cable) may have left the TCP Transport plugin in an inconsistent state, preventing communication from reestablishing once the network interface reconnected. This problem has been resolved.

[RTI Issue ID COREPLG-280]

4.12.45 TCP Transport Plugin Reported Invalid-State Error when Running in Debug Mode

Under certain circumstances, a TCP Transport plugin running in debug mode may have erroneously logged an error and failed to send data. In particular, this issue may have been triggered after a connection was disconnected. The following error was printed:

```
NDDS_Transport_TCPv4_Plugin_verifySendResourceState_
clientEA:detected send resource in an invalid
state: SR=DISCONNECTED, CC=BOUND, DC=BOUND
```

```
NDDS_Transport_TCPv4_send:sendresource state validation
failed
```

This problem has been resolved.

[RTI Issue ID COREPLG-290]

4.12.46 TCP Transport Plugin Stopped on Asynchronous Send Failure

When using RTI TLS Support and the **force_asynchronous_send** property (in NDDS_Transport_ TCPv4_Property_t), or when using TCP and a **socket_monitoring_kind** (also in NDDS_Transport_ TCPv4_Property_t) of SELECT, an error in the send was incorrectly treated as fatal. This could have caused the TCP Transport plugin to shut down. This problem has been resolved.

[RTI Issue ID COREPLG-252]

4.12.47 TCP Transport Properties parent.allow_interfaces_list and parent.deny_interfaces_list properties had no Effect

The TCP Transport plugin properties **parent.allow_interfaces_list** and **parent.deny_interfaces_list** had no effect. This problem has been resolved.

[RTI Issue ID COREPLG-344]

4.12.48 TCP Transport SSL Handshake could Hang Forever

The TCP Transport plugin was not robust against failures in the negotiation of the initial handshake required when enabling TLS Support. Consequently, a connection may have remained forever in a "handshake in progress" state, thus preventing communication.

To address this issue, the TCP Transport plugin now supports a new property: **initial_handshake_ timeou**t. This property controls the maximum time (in seconds) the initial handshake for a connection can remain in progress. The default is 10 seconds. If the handshake has not completed after the specified timeout, the connection will be closed. This way, the TCP Transport plugin can restart the process of establishing and handshaking that connection.

[RTI Issue ID COREPLG-303]

4.12.49 Transport Plugin Resources not Cleaned Up when Remote Endpoints Removed

Transport plugins may not have cleaned up resources when remote endpoints were removed. For example, with the TCP transport plugin this may have resulted in connections that were not closed. This problem, which only occurred with reliable unicast communication, has been resolved.

[RTI Issue ID CORE-6083]

4.12.50 Two Participants on Same Node did not Communicate when One Disabled Shared Memory but still Used UDP Loopback

Two participants on the same node may not have communicated under certain circumstances. In particular, this issue was triggered when one of the participants had disabled shared memory and the other participant had enabled both shared memory and a UDP transport with **ignore_loopback_interface** = -1 (auto). In this scenario, the second participant ignored the loopback interface, therefore there was no communication.

Starting with this release, the behavior for UDP transports with i**gnore_loopback_interface** = -1 has changed. Specifically, Connext DDS will not longer ignore the loopback interface, even if shared memory is enabled on the sender. Instead, in order to avoid redundant traffic, Connext DDS will not send traffic to UDP loopback destinations that are reachable through shared memory.

[RTI Issue ID CORE-4052]

4.12.51 Unbounded Memory Growth upon I/O Failure when Using Windows IOCP

An issue in the TCP Transport plugin may have caused unbounded memory growth after a failure when calling I/O-related OS APIs. This issue only affected Windows architectures when the Windows IOCP socket monitoring kind was enabled. This issue has been resolved.

[RTI Issue ID COREPLG-353]

4.12.52 Unexpected Error Message when Enabling TCP Transport Connection-Liveliness

The following error message appeared when a static application loaded a TCP transport plugin through QoS and the TCP transport plugin was configured to use the connection-liveliness feature:

```
[D0065|ENABLE]DDS_DomainParticipantGlobals_initializeWorkerFactoryI:
!Potential library mismatch.
```

This may have happened if your application used the static and shared RTI core libraries simultaneously (such as using the shared RTI Monitoring library and linking statically with the RTI core libraries). This problem has been resolved.

[RTI Issue ID COREPLG-319]

4.12.53 Unbounded Memory Growth during TCP Transport Plugin Execution

There was an issue that may have provoked an unbounded memory growth during TCP Transport plugin execution. In particular, this issue may have been triggered when the TCP Transport plugin reconnected to a remote peer. This problem has been resolved.

[RTI Issue ID COREPLG-296]

4.12.54 Unexpected Memory Growth in TCP Transport Client upon Connection Close

There may have been unexpected memory growth of ~160 bytes in the TCP Transport plugin upon closure of a client data connection. This issue affected the TCP Transport plugin running in WAN Asymmetric mode as a client. Note this was not a leak, since the memory was properly returned upon TCP Transport plugin destruction. This problem has been resolved.

[RTI Issue ID COREPLG-310]

4.12.55 Unexpected Memory Growth in TCP Transport Server upon Connection Close

There may have been unexpected memory growth of ~160 bytes in the TCP Transport plugin upon closing a server connection. In particular, this was a timing issue that may have triggered in a scenario where a TCP Transport plugin was accepting and closing connections at a very high rate. Note this was not a leak, since the memory was properly returned upon TCP Transport plugin destruction. This problem has been resolved.

[RTI Issue ID COREPLG-305]

4.12.56 Unexpected Messages when Using TCP Transport's 'Force Asynchronous Send'

The TCP Transport plugin had an issue during the clean up of connections when **force_asynchronous_send** was enabled. Consequently, the TCP Transport plugin Client may have delivered unexpected messages to the remote TCP Transport plugin Server. This problem has been resolved.

[RTI Issue ID COREPLG-304]

4.12.57 Unexpected TCP Transport Plugin Error on Packet Reception when Using IOCP Mode

When enabling IOCP monitoring with the property **socket_monitoring_kind**, the TCP transport plugin did not properly catch WSAECONNRESET errors. This may have caused a lost connection to never recover.

When this problem occurred, the following messages were continuously printed:

```
NDDS_Transport_TCPv4_receive_rEA:SocketGroup.wait failed
NDDS_Transport_TCP_SocketGroup_waitForCompletionPacket:error
returned by GetQueuedCompletionStatus in SocketGroup wait
issuing recvZero: (errno: 10054) - An existing connection was
forcibly closed by the remote host.
```

This problem has been resolved.

[RTI Issue ID COREPLG-241]

4.12.58 Unexpected TCP Transport Plugin Error when TCP Server Sent Data Asynchronously

As a result of losing a TCP connection while sending a TCP packet, the TCP server may have ended up in a state that was not properly handled. This was a timing issue that only occurred in some cases while using asymmetric mode together with the property **force_asynchronous_send**.

When this problem occurred, the following message was continuously printed:

```
NDDS_Transport_TCPv4_Plugin_processControlEvent:control
protocol error: unexpected WRITE event on a NULL SR
```

This problem has been resolved.

[RTI Issue ID COREPLG-247]

4.12.59 Unexpected TCP Transport Plugin WSAECONNABORTED Error on Packet Reception when Using IOCP Mode

When enabling IOCP monitoring with the property **socket_monitoring_kind**, the TCP transport plugin did not properly catch WSAECONNABORTED errors. This may have caused a crash in the TCP Transport plugin.

When this problem occurred, the following message was printed:

```
NDDS_Transport_TCP_SocketGroup_waitForCompletionPacket:
error returned by GetQueuedCompletionStatus in
SocketGroup wait issuing recvZero: (errno: 10053) -
An established connection was aborted by the software
in your host machine.
```

This problem has been resolved.

[RTI Issue ID COREPLG-291]

4.12.60 Unnecessary System Resources Allocated when Transport Priority Left at Default Value

In the 5.0.0.35 release, there was an issue that may have caused unnecessary waste of system resources—in particular, memory and sockets. This issue was triggered when the TransportPriorityQosPolicy of a *DataWriter* or the **metatraffic_transport_priority** field in the DiscoveryQosPolicy was left at the default value. This problem has been resolved.

[RTI Issue ID CORE-6774]

4.12.61 Unreadable Characters in Log Messages After Some Errors in TCP Transport

Errors in the TCP transport may have generated log messages with unreadable characters. For example:

```
RTIOsapiUtility_getErrorString:!input buffer too small
NDDS_Transport_TCP_SocketGroup_modify_socket:error returned by
asynchronous operation recvZero in socket group: (errno: 10057)
- ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠I'
NDDS_Transport_TCPv4_Plugin_clientProcessControlConnect:attempt
to modify state of socket 1012 in socket group 0574FD40 failed
```

This problem has been resolved.

[RTI Issue ID COREPLG-336]

4.12.62 Wrong Results for NDDS_Transport_Support_lookup_transport API

There was an issue that prevented **NDDS_Transport_Support_lookup_transport()** from returning a transport that matched the passed in transport_in parameter. This issue has been resolved.

[RTI Issue ID CORE-6642]

4.13 Fixes Related to Other Issues

4.13.1 Access Violation when Calling get() on Loaned Sequence if Provided Index Out of Bounds of the Sequence–Java API Only

Calling a loaned sequence's **get()** operation with a parameter of an invalid index (that is, an index out of the bounds of the sequence), resulted in a crash when using the Java API. This problem has been resolved. Now, instead of causing a crash, a java.lang.IndexOutOfBoundsException will be thrown.

[RTI Issue ID CORE-6198]

4.13.2 Built-in Profiles Overrode DomainParticipantFactoryQos Values Set in Application Code

The values for the DomainParticipantFactoryQos may have been overridden by the built-in QoS profiles if they were set in code before the DomainParticipantFactory was created and the default QoS profile inherited from any of the built-in QoS profiles. This problem has been resolved. The DomainParticipantFactory's default QoS values are no longer set in the Baseline.Root built-in profile (but have been kept as a comment in the profile for reference).

[RTI Issue ID CORE-6075]

4.13.3 Copy_from() Method in Bytes and KeyedBytes Built-in Types may have Thrown Exception–.NET API Only

The .NET **copy_from()** method in the Bytes and KeyedBytes built-in types may have thrown an exception if the **src** parameter had a length of zero elements. This problem has been resolved.

[RTI Issue ID CORE-6560]

4.13.4 Crash when Using XML-based Application Creation for Multiple Participants

If an application attempted to concurrently create two or more *DomainParticipants* using XML-based Application Creation, a crash may have occurred. These operations were not thread-safe. This problem has been resolved.

[RTI Issue ID CORE-6178]

4.13.5 Custom Flow Controller on Built-in Discovery DataWriters caused Participant Deletion to Fail

Installing a custom flow controller on the built-in discovery *DataWriters* by setting the **flow_controller_name** of the **publication_writer_publish_mode** and/or **subscription_writer_publish_mode** fields in the DiscoveryConfigQosPolicy caused the call to **DDS_DomainParticipant_delete_contained_entities()** to fail. The flow controllers could not be deleted because the built-in *DataWriters* had not been deleted yet. All flow controllers are now deleted after the built-in *DataWriters* are deleted, allowing participant destruction to complete successfully.

[RTI Issue ID CORE-6511]

4.13.6 DataReader using GROUP_PRESENTATION_QOS Stopped Providing Samples to Application in Some Cases

A *DataReader* using GROUP_PRESENTATION_QOS may have stopped providing samples to the application in some cases. The issue occurred when the virtual heartbeat period (configured with **writer_qos.protocol.rtps_reliable_writer.virtual_heartbeat_period**) was smaller than the RTPS heartbeat period (configured with **writer_qos.protocol.rtps_reliable_writer.(_fast.)heartbeat_period**).

Workarounds to this problem included:

- Setting the virtual heartbeat period smaller than the RTPS fast heartbeat period
- Setting **writer_qos.protocol.availability.max_data_availability_waiting_time** to a finite value to unblock the reader. This workaround introduced extra latency.

This problem has been resolved.

[RTI Issue ID CORE-6499]

4.13.7 DDS_TypeCode_equals did not Compare Extensibility Kind of Two Types

Given two types that are identical except in their extensibility kind (for example, one Mutable and one Extensible), **DDS_TypeCode_equals()** returned true.

This problem has been resolved; now **DDS_TypeCode_equals()** will return false if two types have different extensibility kinds.

[RTI Issue ID CORE-6640]

4.13.8 Deleting Participants was not Thread Safe–C++ and .NET APIs Only

When using the C++ or .NET APIs, deleting multiple *DomainParticipants* from different threads was not thread-safe. This may have caused System.AccessViolationExceptions when using the .NET API and segmentation faults when using the C++ API. This problem has been resolved.

[RTI Issue ID CORE-6033]

4.13.9 Deserialization of Samples with Sequences Took a Long Time in Java

The deserialization of samples with sequences took a long time in Java.

When a type is marked as EXTENSIBLE or MUTABLE, the deserialize function calls clear on a sample before it is deserialized. If the sample type has a sequence, the clear method for the sequence is also called.

In previous releases, calling clear on a sequence took a long time because the implementation required a quadratic time. This problem has been resolved.

[RTI Issue ID CORE-6222]

4.13.10 Deserialization of ByteSeq using ObjectInputStream Failed if Longer than 1024 Bytes - Java API Only

Using an ObjectInputStream to deserialize a type with a ByteSeq failed if the sequence was longer than 1024 bytes. This use case may have applied, for example, when trying to write and read DDS samples to and from a file. The error message read:

```
Exception in thread "main" java.io.IOException: unable to read all expected
data
```

This problem has been resolved.

[RTI Issue ID CORE-5375]

4.13.11 DomainParticipant.enable() Threw Exception with No Message when It Failed – Java API Only

If a failure occurred while attempting **to enable()** a *DomainParticipant*, the resulting Java Exception contained an empty message instead of the description of the error. This problem has been resolved.

[RTI Issue ID CORE-6325]

4.13.12 Error Printing Unsigned Longs with FooSupport_print_data()

The **FooSupport_print_data()** operation may have printed incorrect values for unsigned long variables on some platforms. For example, consider the following IDL type:

```
struct MyType {
    unsigned long ul;
};
```

If you set **ul** to 1500, **FooSupport_print_data()** may have printed 0 instead. This problem has been resolved.

[RTI Issue ID CORE-6154]

4.13.13 Error When Enabling DomainParticipant

A *DomainParticipant* could not be enabled if it was configured to use Secure WAN Transport. When this issue occurred, the following message was displayed:

```
NDDS_Transport_WAN_plugin_property_from_DDS_property:Unexpected property:
dds.transport.wan_plugin.wan.parent.domain_participant_ptr.
Closest valid property:
dds.transport.wan_plugin.wan.parent.parent.domain_participant_ptr
```

This problem has been resolved.

[RTI Issue ID COREPLG-315]

4.13.14 Error Writing Primitive Sequences with Maximum of Zero Elements --.NET API Only

In previous releases when using the .NET API, if you published a sample in which one of the members was a primitive sequence with a maximum of zero elements, the write operation would have failed with an error like this:

```
at DDS.CdrStream.serialize_primitive_sequence(Sequence`1 seq) in
dds_dotnet.1.0\srccpp\managed\managed_cdr.cpp:line 1870
PRESWriterHistoryDriver_initializeSample:!serialize
WriterHistoryMemoryPlugin_addEntryToSessions:!initialize sample
WriterHistoryMemoryPlugin_getEntry:!add virtual sample to sessions
WriterHistoryMemoryPlugin_addSample:!get entry
PRESWriterHistoryDriver_addWrite:!add_sample
PRESPsWriter_writeInternal:!collator addWrite
write error DDS.Retcode_Error: Exception of type 'DDS.Retcode_Error'
was thrown.
```

This problem has been resolved.

[RTI Issue ID CORE-6137]

4.13.15 Exception in Java while Using HashMap with DomainParticipantQos as a Key

Use of the **java.util.HashMap** class with the DomainParticipantQos as a key produced a NullPointerException. This problem has been resolved.

[RTI Issue ID CORE-6111]

4.13.16 Failure Copying Sample after Calling take/read not Reported as Error—.NET API Only

After calling **take()** or **read()** on a DataReader, a failure when copying a sample was not reported as an error. Consequently, the user may have received invalid data. This problem, which only affected the .NET API, has been resolved.

[RTI Issue ID CORE-6118]

4.13.17 FlowController::set_property() Returned Wrong Error Codes in Some Cases

FlowController::set_property() did not return the correct error codes in some cases.

- When attempting to modify an immutable property, **FlowController::set_property()** returned DDS_RETCODE_INCONSISTENT_POLICY instead of DDS_RETCODE_IMMUTABLE_POLICY.
- When attempting to pass inconsistent properties, it returned DDS_RETCODE_BAD_PARAM instead of DDS_RETURN_CODE_INCONSISTENT_POLICY.

This problem has been resolved and the correct error codes will be returned.

[RTI Issue ID CORE-6483]

4.13.18 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]

4.13.19 Incorrect Conversion of Time_t and Duration_t due to Overflow

The Time_t and Duration_t conversion methods, such as **from_micros()**, **from_nanos()**, and **from_millis ()**, returned an incorrect value if the converted value was greater than the maximum value that can be stored in the Time_t and Duration_t objects.

This problem has been resolved. The Time_t and Duration_t conversion methods will return TIME_MAX and DURATION_INFINITE, respectively, if the passed in parameter produces an overflow.

[RTI Issue ID CORE-6438]

4.13.20 Incorrect IDL char Deserialization by Java DataReaders

A Java *DataReader* did not correctly deserialize members of type 'char' when the value was in the range [128,255]. For example:

```
struct MyType {
    char m1;
};
```

Consequently, the value sent by a *DataWriter* was not that same as the value the receiving *DataReader* provided to the application. This problem has been resolved.

[RTI Issue ID CORE-6758]

4.13.21 Initialize and Copy operations for DDS_WriteParams_t did not Process All Members–C/C++ APIs Only

Some members in the DDS_WriteParams_t structure were not initialized properly by the constructor in C++ and by **DDS_WriteParams_t_initialize()** in C. Similarly, some members were not copied by the copy operator in C++ and by **DDS_WriteParams_t_copy()** in C. This may have resulted in wrong behavior when using the *DataWriter's* **write_w_params()** operation. This issue have been resolved.

[RTI Issue ID CORE-6134]

4.13.22 JVM Crash when Passing Null Type or Topic Names to Participant's create_topic()

If the *DomainParticipant's* **create_topic()** function passed a null value for the type name of topic name, this caused a crash in the Java Virtual Machine. This problem has been resolved.

[RTI Issue ID CORE-5558]

4.13.23 Large Samples Never Repaired in Some Scenarios

In scenarios in which the transport property **message_size_max** was greater than DDS_RtpsReliableWriterProtocol's **max_bytes_per_nack_response**, historical data samples may not have been delivered. When this issue was triggered, communication may have been blocked, so that the writer could not send data to the reader. This issue has been resolved.

[RTI Issue ID CORE-6533]

4.13.24 Late-Joiner DataReader may not have Received All Historical Data

A late-joiner *DataReader* with Durability set to TRANSIENT_LOCAL, TRANSIENT, or PERSISTENT may not have received all the historical data that was available from a matching *DataWriter*. This problem has been resolved.

[RTI Issue ID CORE-6288]

4.13.25 Logging sent Non-Printable (non-ASCII) Data to Console

Under certain circumstances, some of the logging messages printed by Connext DDS included non-printable characters. In particular, this issue was triggered when logging verbosity was set to STATUS_ALL and logging category was set to COMMUNICATION.

When this issue was triggered, the value for the key associated to certain events was not correctly printed, as in the following example:

```
COMMENDAnonWriterService_onDomainBroadcastEvent:writing periodic
keyed data: SN=0x000000001, key=(16)
╙7, 0 bytes
```

This problem has been resolved.

[RTI Issue ID CORE-6181]

4.13.26 Memory Leak in Unmanaged Code when Using .NET API

Deleting a *DataWriter*, *DataReader*, or *DomainParticipant* in the .NET API caused memory leaks in unmanaged code. These memory leaks were related to built-in types and type registration. This problem has been resolved.

[RTI Issue ID CORE-6780]

4.13.27 Monitoring Library Generated Incorrect Values for CPU and Memory Usage - OS X Platforms Only

On OS X platforms, the CPU and memory usage values generated by RTI Monitoring Library were always zero. This problem has been resolved.

[RTI Issue ID MONITOR-199]

4.13.28 .NET Instances of Some Classes were not Garbage Collected

Instances of certain classes were pinned in memory and could not be released by the garbage collector. This may have caused memory growth if an application repeatedly created and deleted DDS entities during the application's life cycle. This problem has been resolved; no instances will remain pinned.

[RTI Issue ID CORE-6703]

4.13.29 Non-Recoverable Inconsistent State in Participant after Failed Call to Enable Participant

If you called enable() on a disabled Participant and that call failed, the Participant may have ended up in an unrecoverable state. That is, any further calls to enable the Participant would never succeed, even if the conditions that provoked the initial failure were no longer present. This problem has been resolved.

[RTI Issue ID CORE-6264]

4.13.30 Operations Involving StatusCondition::GetHashCode() may have Thrown OverflowException–C# API Only

The StatusCondition overrode the **System.Object ::GetHashCode()** in way that may have thrown an OverflowException in 64-bit architectures. This issue has been resolved.

[RTI Issue ID CORE-6761]

4.13.31 Poor Performance of DataWriter's wait_for_acknowledgments()–Windows Platforms Only

The *DataWriter's* **wait_for_acknowledgments()** operation had poor performance poor on Windows platforms. The performance was limited by the Windows timer resolution (which is 15 msec by default). This problem has been resolved.

[RTI Issue ID CORE-6619]

4.13.32 Possible Crash when Calling take_next_sample() After Failed Registration of Builtin Type–Java API Only

Under certain circumstances, a call to a DynamicData *DataReader's* **take_next_sample()** may have caused a JRE crash. In particular, this issue may have been triggered when **take_next_sample()** was called for a built-in type after a failed call to register_type() for that same built-in type.

This problem has been resolved.

[RTI Issue ID CORE-6240]

4.13.33 Possible Crash if Type Unregistered from Participant while Another Participant Still using that TypeSupport–C++ API Only

If the same TypeSupport was registered to more than one participant and it was unregistered from one of the participants, any other participant that tried to use the TypeSupport would crash.

This scenario could also become a problem with built-in topics and types. If one participant was deleted and another participant tried to access the built-in readers or use the built-in types, the application would crash.

This problem has been resolved.

[RTI Issue ID CORE-6695]

4.13.34 Possible Failure of copy_from/copy_from_no_alloc in a .NET Loanable Sequence

The operations **copy_from()** and **copy_from_no_alloc()** in a .NET Loanable sequence may fail in the maximum number of elements in the source sequence is different than the maximum in the destination sequence. A Loanable sequence is a sequence of a type that can be published with Connext DDS.

[RTI Issue ID CORE-6119]

4.13.35 Possible Failure when Registering Same Type Twice in Two Threads

Registering a type for a second time on the same DomainParticipant is valid and doesn't have any effect. For example:

```
FooSupport_register_type(participant, "Foo"); // returns DDS_RETCODE_OK\
FooSupport_register_type(participant, "Foo"); // returns DDS_RETCODE_OK (but doesn't do
anything)
```

However, if both calls to **FooSupport_register_type()** happened concurrently, sometimes one succeeded and the other one failed.

This problem has been resolved; now both calls will succeed, even if they are executed concurrently.

[RTI Issue ID CORE-6167]

4.13.36 Possible Memory Leak if Finalizing DomainParticipantFactory Failed

If finalizing the DomainParticipantFactory failed, there may have been a memory leak (depending on why the finalization failed). This problem has been resolved.

[RTI Issue ID CORE-6362]

4.13.37 Possible Memory Leak upon Failed DomainParticipant Creation

Under certain circumstances, a failed *DomainParticipant* creation may have caused a memory leak. This problem has been resolved.

[RTI Issue ID CORE-6377]

4.13.38 Possible Segmentation Fault on Keyed Reader when Using Collaborative DataWriters or Group Ordered Access

When using collaborative *DataWriters* or group ordered access, a *DataReader* reading keyed samples may have experienced a segmentation fault in the function **PRESCstReaderCollator_addCollatorEntryToVirtualWriterQueue()**.This problem has been resolved.

[RTI Issue ID CORE-6501]

4.13.39 Possible Race Condition during Participant Deletion may have Caused "deadlock risk" Error

In limited scenarios, deletion of *DomainParticipants* may have caused a "deadlock risk" error message and prevented completion of DDS operations that were being executed in other threads. This issue only occurred in applications that had created multiple *DomainParticipants*. The specific error message was similar to:

```
REDAWorker_enterExclusiveArea:worker U70f73cc0 deadlock risk:
cannot enter 0x10182a180 of level 30 from level 50
```

This problem has been resolved.

[RTI Issue ID CORE-6045]

4.13.40 Potential Segmentation Fault after Participant Deletion

Creating multiple *DomainParticipants* in parallel may have caused a segmentation fault when those *DomainParticipants* were deleted. This problem has been resolved.

[RTI Issue ID CORE-6805]

4.13.41 Potential Segmentation Fault in Unkeyed DataReaders when Trying to Remove Expired Samples

An unkeyed *DataReader* may have issued a segmentation fault when trying to remove expired samples from its cache. Samples can be expired because the associated remote *DataWriter* set a lifespan or because of delays set in ReaderDataLifecycleQosPolicy. The segmentation fault only occurred when the *DataReader* tried to remove an expired sample while the user application was holding a loan on the sample.

Notice that when using the debug libraries, the *DataReader* would not issue a segmentation fault, but would enter an infinite loop printing the following message:

```
REDAInlineList_removeNodeEA:!precondition: list == ((void *)0) || node ==
((void *)0) || node->inlineList != list
```

This problem has been resolved.

[RTI Issue ID CORE-6775]

4.13.42 Potential Segmentation Fault when Creating DataWriters and DataReaders

The creation of a *DataWriter* and/or *DataReader* may have caused a rare segmentation fault. This problem has been resolved.

[RTI Issue ID CORE-6571]

4.13.43 Potential Segmentation Fault when Setting AvailabilityQosPolicy's required_matched_endpoint_groups on a DataReader

Setting the AvailabilityQosPolicy's **required_matched_endpoint_groups** to a value other than empty on a *DataReader* may have caused a segmentation fault. This problem has been resolved.

[RTI Issue ID CORE-6719]

4.13.44 Registering a Type Twice with Same Name, Different Type Plugin should have Failed

Registering a type twice with the same name and same type plugin is valid. However, using a different type plugin is an application error and should return DDS_RETCODE_PRECONDITION_NOT_MET. In previous releases, this sequence should have failed but didn't:

```
DDSDomainParticipant participant = ...;
FooTypeSupport_register_type(participant, "Foo");
BarTypeSupport_register_type(participant, "Foo");
```

This problem has been resolved. Now the second call to **register_type()** in the above example will return DDS_RETCODE_PRECONDITION_NOT_MET.

[RTI Issue ID CORE-3962]

4.13.45 RTI DDS Ping did not Print Data Values when using -useKeys and Default Verbosity

RTI DDS Ping (rtiddsping) did not print data values when using the **-useKey**s option and default verbosity. In addition, the output was not formatted correctly. You would have seen the following messages:

```
Sending data...   key: 0000001Sending data...   key: 0000002Sending data...
key: 0000003Sending data...   key: 0000004Sending data...   key: 0000005Sending
data...   key: 0000006Sending data...   key: 0000007Sending data...
```

This problem has been resolved.

[RTI Issue ID CORE-6820]

4.13.46 Segmentation Fault in Java if Default name or role_name in EntityNameQosPolicy not NULL

A Java application may have produced a segmentation fault if the default **name** or **role_name** in the EntityNameQosPolicy for a DDS *Entity* (*DomainParticipant*, *Publisher*, *Subscriber*, *DataWriter*, or *DataReader*) was set to a non-NULL value and the user overwrote this value during entity creation to set it to NULL. This problem has been resolved.

[RTI Issue ID CORE-6772]

4.13.47 Segmentation Fault when DataWriter Creation Failed because System Running Out of Memory

There was an issue that may have caused a segmentation fault when DataWriter creation failed as a consequence of the system running out of memory. In particular, this issue may have occurred when using unbounded strings. This problem has been resolved.

If a *DataWriter* could not be created because the system was running out of memory, this may have caused a segmentation fault. In particular, this issue may have occurred when using unbounded strings. This problem has been resolved.

[RTI Issue ID CORE-6728]

4.13.48 TypeCode::equal Considered TypeCodes with Arrays of Different Dimensions as Equal

TypeCode::equal() may have returned true for two typecodes even if they were different. This occurred when the TypeCodes contained arrays with the same type but different dimensions. This problem has been resolved.

[RTI Issue ID CORE-6527]

4.13.49 Unbounded Memory Growth when Calling NDDS Config Logger Set Output APIs

Calling the NDDS Config Logger APIs **set_output_file()**, **set_output_file_name()**, or **set_output_device()** may have caused unbounded memory growth. The same issue also affected **DDS_DomainParticipantFactory_set_qos()**. Note that this issue was not a memory leak, since all the used memory was released upon application shutdown. This problem has been resolved.

[RTI Issue ID CORE-6706]

4.13.50 Unbounded Memory Growth when Calling NDDS Config Logger Set Output APIs

Calling the NDDS Config Logger APIs **set_output_file()**, **set_output_file_name()**, or **set_output_device()** may have caused unbounded memory growth. The same issue also affected **DDS_DomainParticipantFactory_set_qos()**. Note that this issue was not a memory leak, since all the used memory was released upon application shutdown. This problem has been resolved.

[RTI Issue ID CORE-6706]

4.13.51 Unexpected Error in Publishing Application when Writing Coherent Set of Samples

When piggyback heartbeats are enabled (this is, if the DataWriterProtocolQosPolicy's **rtps_reliable_writer.heartbeart_per_max_samples** != 0), writing a coherent set of samples may have printed the following error upon a call to **end_coherent_changes**():

```
PRESPsService_writerSampleListenerOnQueueUpdate:!get pres psWriter
```

This error prevented the pruning of expired samples during the piggyback heartbeat send. In this sense, if the LifespanQosPolicy's **duration** was set to INFINITE, this problem had no consequences beyond printing the error.

This problem has been resolved.

[RTI Issue ID CORE-6203]

4.13.52 Unexpected Error when get_matched_subscriptions/publications() Called Right After Ignoring Subscription/Publication

A race condition may have caused the **get_matched_subscriptions()**/**get_matched_publications()** operations to fail if they were called right after ignoring an entity (with the **ignore_publication()**/**ignore_subscription()** operations).

This problem has been resolved.

[RTI Issue ID CORE-6271]

4.13.53 Unexpected Memory Growth When Using DDS_GROUP_PRESENTATION_QOS on a DataReader

DataReaders using DDS_GROUP_PRESENTATION_QOS failed to remove all the information about a remote *DataWriter* when that *DataWriter* was deleted. This may have caused unbounded memory growth over time as new *DataWriters* were created and deleted. Notice that this issue was not reported as a memory leak because all the memory was reclaimed after the *DataReader* was destroyed. This problem has been resolved.

[RTI Issue ID CORE-6721]

4.13.54 Unexpected Timeout from DataReader's wait_for_historical_data() when using Delegated Reliability

When a *DataWriter* and matching *DataReader* were configured for delegated reliability with RTI Persistence Service, the *DataReader's* **wait_for_historical_data()** operation always returned a TIMEOUT error, even if the *DataReader* had not received all historical data from RTI Persistence Service. This problem has been resolved.

[RTI Issue ID CORE-6351]

4.13.55 Unexpected Memory Growth when Setting Entity QoS name or role_name

Calling **set_qos()** for an *Entity* that had a non-NULL EntityQosPolicy **name** or **role_name** resulted in unbounded memory growth, which could eventually exhaust the available memory. This problem has been resolved.

[RTI Issue ID CORE-6722]

4.13.56 Version Number was Missing from Libraries for Windows Platforms

Libraries for Windows platforms did not include the version number in the version field. This problem has been resolved. The version number is now included.

To see the version number, run the DUMPBIN utility that comes with Visual Studio®. For example:

```
DUMPBIN/HEADERS nddscore.dll
```

You will find the version number encoded in the 'image version' in the ' OPTIONAL HEADER VALUES ' section:

```
OPTIONAL HEADER VALUES
<snip>
50200.00 image version
<snip>
```

Since this field must be formatted as number.number, the format is <major_version><minor_version><terciary_version>.<patch_version>. For example, Connext DDS version 5.2.1.3 would appear as image version 50201.03.

[RTI Issue ID CORE-4010]

4.13.57 XSD Validation Failed when type_object_max_deserialized_length was UNLIMITED

The XSD validation of QoS profiles using the file **rti_dds_qos_profiles.xsd** failed when **type_object_max_deserialized_length** (under <participant_qos>/<resource_limits>) was set to UNLIMITED. This problem has been resolved.

[RTI Issue ID CORE-6189]

Chapter 5 Known Issues

5.1 AppAck Messages Cannot be Greater than Underlying Transport Message Size

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

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

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

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

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

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

[RTI Issue ID CORE-5329]

5.2 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]

5.3 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]

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

Given a Connext DDS 5.2.0 application with the shared memory transport enabled, a Connext DDS Micro 2.4.x application will fail to discover it. This is due to a bug in Connext DDS Micro that prevents a received participant discovery message from being correctly processed. This bug will be fixed in a future release of Connext DDS Micro. As a workaround, you can disable the shared memory transport in the Connext DDS application and use UDPv4 instead.

[RTI Issue ID EDDY-1615]

5.5 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.

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

If you inherit from either the **BuiltinQosLibExp::Generic.StrictReliable.HighThroughput** or the **BuiltinQosLibExp::Generic.AutoTuning** built-in QoS profiles, your *DataWriters* and *DataReaders* will fail to communicate if you are writing small samples.

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

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

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

[RTI Issue ID CORE-6411]

5.7 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]

5.8 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]

5.9 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-5.2.0
```

5.10 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]

5.11 Writer-Side Filtering May Cause Missed Deadline

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

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

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

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

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

5.13 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]

5.14 Known Issues in RTI Monitoring Library

5.14.1 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).

5.14.2 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*.

5.14.3 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 6 Experimental Features

Experimental features 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.

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);
```

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

Experimental features are clearly documented as such in the RTI Connext DDS Core Libraries What's New document or the Release Notes for the component in which they are included, as well as in the component's User's Manual.

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/).