RTI Connext DDS Core Libraries

Release Notes

Version 5.3.1

Printed in U.S.A. First printing.
March 2018.

Trademarks

Copy and Use Restrictions

Third-Party Copyright Notices

Technical Support

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

Contents

Chapter 1 Introduction

Connext DDS 5.3.1 is a maintenance release based on feature release 5.3.0. This document describes fixes in Connext DDS 5.3.1. These enhancements have been made since 5.3.0. This document includes the following:

For an overview of new features in 5.3.1, see the separate *What's New* document for 5.3.1.

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

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

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

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

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

Table 2.1 Supported Platforms

Operating System	Version
AIX®	AIX 7.1
Android™	Android 2.3 - 4.4, 5.0, 5.1
INTEGRITY® (target only)	INTEGRITY 5.0.11, 10.0.2, and 11.0.4
iOS®	iOS 8.2
Linux® (ARM® CPU)	NI™ Linux 3 Raspbian Wheezy 7.0 Ubuntu® 16.04 LTS

Table 2.1 Supported Platforms

Operating System	Version
Linux (Intel® CPU)	CentOS 6.0, 6.2 - 6.4, 7.0 Red Hat® Enterprise Linux 6.0 - 6.5, 6.7, 6.8, 7.0 Ubuntu 12.04 LTS, 14.04 LTS, 16.04 LTS SUSE 11[a] WindRiver® Linux 7.0
LynxOS® (target only)	LynxOS 4.0, 4.2, 5.0
OS X®	OS X 10.10 - 10.13
QNX® (target only)	QNX Neutrino® 6.4.1, 6.5, 7.0
Solaris™	Solaris 2.10
VxWorks® (target only)	VxWorks 6.9, 6.9.3.2, 6.9.4.x, 7.0 VxWorks 653 2.3
Windows®	Windows 7, 8, 8.1, 10 Windows Server 2008 R2 Windows Server 2012 R2 Windows Server 2016

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 MPC 8349 CPU

[a]Available upon request.

Table 2.2 Custom Supported Platforms

Operating System	Version
Linux	Debian™ Linux 3.12 on ARMv7a Cortex-A9 CPU
	Freescale™ Linux 3.8.13 on QorIQ or P4040/P4080/P4081 CPU
	Red Hat Enterprise Linux 5.2 on x86 CPU with gcc 4.2.1
	RedHawk™ Linux 6.0 on x64 CPU
	Wind River Linux 7 on ARMv7 CPU
	RedHawk™ Linux 6.5 on i86 and x64 CPUs
	Xilinx® Zynq® Linux 3.8.11 on ARMv7 CPU
	Yocto Project® 2.2 on ARM Cortex-A53 CPU
QNX	QNX 6.5 on PPC E500 v2 CPU
	QNX 6.6 on ARMv7 and x86 CPUs
VxWorks	VxWorks 6.9.4.6 on PPC CPU

2.2 Requirements when Using Microsoft Visual Studio

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

When Using Visual Studio 2008 — Service Pack 1 Requirement

You must have Visual Studio 2008 Service Pack 1 or the Microsoft Visual C++ 2008 SP1 Redistributable Package installed on the machine where you are *running* an application linked with dynamic libraries.

This includes dynamically linked C/C++ and all .NET and Java applications. The Microsoft Visual C++ 2008 SP1 Redistributable Package can be downloaded from the following Microsoft websites:

For x86 architectures:

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

For x64 architectures:

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

When Using Visual Studio 2010 — Service Pack 1 Requirement

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

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

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

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

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

When Using Visual Studio 2012 — Update 4 Redistributable Package Requirement

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

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

When Using Visual Studio 2013 — Redistributable Package Requirement

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

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

When Using Visual Studio 2015 — Update 3 Redistributable Package Requirement

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

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

When Using Visual Studio 2017 — Update 2 Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2017 Update 2 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2017 Update 2 from this Microsoft website: https://www.visualstudio.com/downloads. Scroll down to the "Microsoft Visual C++ Redistributable for Visual Studio 2017" section.

2.3 Disk and Memory Usage

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

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

2.4 Networking Support

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

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

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

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

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

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

Chapter 3 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 2.2. 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 5.3.x

3.1.3.1 Discovery Wire Compatibility with Applications using Connext DDS 5.2.x and Earlier

3.1.3.1.1 Discovery Wire Compatibility with Applications using Connext DDS 5.2.0 and Earlier

Starting with Connext DDS 5.3.0 (also in 5.2.7) the numbers of locators that can be announced by a *DomainParticipant* has been increased from 4 to 16. If your Connext DDS application runs on a machine with multiple interfaces, or if you have enabled multiple transports, the discovery information that is announced to other participants may be incompatible with applications running Connext DDS 5.2.0. This problem does not affect applications running Connext DDS 5.2.3.

When this happens, messages similar to these will appear on the remote participant:

```
PRESCstReaderCollator_storeSampleData:!deserialize
COMMENDSrReaderService_assertRemoteWriter:!create reachable destination
```

To ensure backwards compatibility with 5.2.0 and older releases, there is new property, **dds.domain_participant.max_announced_locator_list_size**, in the *DomainParticipant's* Property QoS. This property limits the maximum number of locators *per locator list* the local participant will announce to other participants. The default value for this property is 16. The following snippet shows how to configure it:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>
                  dds.domain_participant.max_announced_locator_list_size
                </name>
                <value>16</value>
            </element>
        </value>
    </property>
</participant_qos>
```

3.1.3.1.2 Discovery Wire Compatibility with Applications using Connext DDS 5.3.0 and Earlier

Starting with Connext DDS 5.3.0 (also in 5.2.7), a *DomainParticipant* is able to propagate changes to IP addresses as part of the discovery traffic. Note that applications using *Connext DDS* 5.2.3 and earlier versions will report errors like the one below if they detect changes to IP addresses.

```
PRESParticipant_assertRemoteParticipant:!assert remote participant ac110001 3a33 1 due to
different ro area
DISCParticipantDiscoveryPlugin_assertRemoteParticipant:!assert remote participant:
0XAC110001,0X3A33,0X1,0X1C1
DISCSimpleParticipantDiscoveryPluginReaderListener_onDataAvailable:!assert remote participant
```

You can disable the notification and propagation of IP address changes to be compatible with 5.2.3 by setting the following transport property: **<<transport prefix>>.disable_interface_tracking**.

For example, for UDPv4 the property name is: **dds.transport.UDPv4.builtin.disable_interface_tracking**.

3.1.4 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.

3.1.4.1 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 5.2 and higher	2.2
Connext DDS 4.5f -5.1	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

3.1.4.2 double, long long, unsigned long long or long double Wire Compatibility

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

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

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

3.1.4.3 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 and Configuration 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, 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 Changes in Generated Type Plugin Code in Connext DDS 5.3.0

The *rtiddsgen*-generated type-plugin code for user-defined data type changed in 5.3.0 to fix some bugs and facilitate some new features. **If you have code that was generated with rtiddsgen 5.2.x or lower, you must regenerate that code** using the version of rtiddsgen provided with this release.

3.2.2.7 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*).

3.2.2.8 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.

3.2.2.9 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.

3.2.2.10 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)**

3.2.2.11 Locator Reachability Configuration

Connext DDS 5.2.2 introduced a new feature that provides the ability to detect unreachable locators. In Connext DDS 5.2.5, the QoS properties that were used to configure this feature have been replaced with QoS values in DDS_DiscoveryConfigQosPolicy and they will be ignored if set.

Specifically, the properties:

- **dds.domain_participant.locator_reachability_assert_period.sec**
- **dds.domain_participant.locator_reachability_assert_period.nanosec**
- **dds.domain_participant.locator_reachability_lease_duration_period.sec**
- **dds.domain_participant.locator_reachability_lease_duration_period.nanosec**
- **dds.domain_participant.locator_reachability_change_detection_period.sec**
- **dds.domain_participant.locator_reachability_change_detection_period.nanosec**

Have been replaced with the following QoS values. The above properties will be ignored if set.

```
struct DDS_DiscoveryConfigQosPolicy {
    ....
    struct DDS_Duration_t locator_reachability_assert_period;
    struct DDS_Duration_t locator_reachability_lease_duration;
    struct DDS_Duration_t locator_reachability_change_detection_period;
    ....
};
```

3.2.2.12 Memory Management Changes for Optional Members in Traditional C++

Starting with Connext DDS 5.2.5, the way memory is managed for optional members in traditional C++ has changed.

In previous releases, applications were advised to allocate and release optional members using the following functions:

- **DDS_Heap_malloc()**
- **DDS_Heap_free()**

The type-support also used these, for example in **FooTypeSupport::create_data()** or **FooTypeSupport::delete_data()**.

Besides being less intuitive than **new()** and **delete()**, these functions posed a problem allocating optional sequences. Sequences have constructors, so **DDS_Heap_malloc()** simply reserved the memory but did not initialize the object. Using **new()** would be inconsistent with the call to **DDS_Heap_free()** from **FooTypeSupport::delete_data()**. The only solution left was to use **DDS_Heap_malloc()**, followed by a call to operator placement new.

In Connext DDS 5.2.5, this problem has been resolved. Now all optional members should be created and destroyed with **new** and **delete**.

Notice that this change may break user code since you may be using **DDS_Heap_malloc()** and **DDS_Heap_free()** to work with optional members. This change requires you to replace calls to **DDS_Heap_malloc()** with **new**, and calls to **DDS_Heap_free()** with **delete**.

3.2.2.13 Changes to the Sequence API in C and Traditional C++

Changes to length():

In Connext DDS 5.3 and 5.2.3.1, the semantic associated with the operation that sets the length of a C/traditional C++ sequence changed.

In previous releases, the **length()** setter in a C/C++ sequence returned an error (false) if the new length exceeded the maximum of the sequence as returned by the **maximum()** operation.

In Connext DDS 5.3 and 5.2.3.1, the **length()** method behavior has changed to allow the sequences to be resized and grow beyond the current maximum if necessary.

For unbounded sequences, the semantic is equivalent to the one provided by the **ensure_length()** operation.

For bounded sequences, the **length()** method allows resizing as long as the new length does not exceed the sequence bound provided in IDL.

Changes to maximum():

In Connext DDS 5.3 and 5.2.3.1 (USACE), you cannot set the maximum of a bounded sequence to a value greater than the value provided in the IDL. This was possible in previous releases.

3.2.2.14 Change to ConfigLogger::set_ouput_device()

Starting with 5.3.0, the .NET API **ConfigLogger::set_ouput_device()** has been renamed to **ConfigLogger::set_output_device()** to fix a typo in the previous name.

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.

3.2.4 Deprecated <participant_library> XML Application Creation Tag

Starting with Connext DDS 5.3.0, the XML tag for the DomainParticipant library has been renamed to <domain_participant_library> so it is compliant with the DDS-WEB OMG standard. The old tag, <participant_library>, is still valid but it maybe removed in future versions of the product.

3.3 Application Binary Interface Compatibility

RTI does not guarantee Application Binary Interface (ABI) compatibility between different versions of Connext DDS.

An application compiled using one version of Connext DDS must be recompiled when moving to a different Connext DDS version.

3.4 Extensible Types Compatibility

3.4.1 General Compatibility Issues

Connext DDS 5.x includes partial support for the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification[a] from the Object Management Group (OMG). This support allows systems to define data types in a more flexible way, and to evolve data types over time without giving up portability, interoperability, or the expressiveness of the DDS type system.

For information related to compatibility issues associated with the Extensible Types support, see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

3.4.2 Java Enumeration Incompatibility Issues

Connext DDS 5.2.0 fixed a bug (RTI Issue ID 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.

3.4.3 Interoperability Issues when Using Keyed Mutable Types

Starting with Connext DDS 5.2.2, the Keyhash calculation for a keyed, mutable type in Java or .NET changed in order to fix a language-interoperability issue (RTI Issue ID CODEGENII-501). If your Java/.NET application built with 5.2.2, 5.2.5, or 5.2.6 needs to communicate with a Java/.NET application built with Connext DDS 5.2.3 or prior General Access Releases (GARs), use the **-use52JavaKeyhash** flag (in 5.2.2, 5.2.5 and 5.2.6) or **-use52Keyhash** (in 5.2.7) to generate the code for your application.

Alternatively, applications built using Connext DDS 5.2.3 can use the flag **-use52CKeyhash** when generating code to be compatible with the 5.2.2, 5.2.5, and 5.2.6 releases.

Starting with Connext DDS 5.2.7 and Connext DDS 5.3.0, the Keyhash calculation for a keyed mutable type inheriting from another keyed mutable type containing key fields in C/C++ and .NET changed in order to fix a language-interoperability issue (RTI Issue ID CODEGENII-693). If your C/C++/.NET application built with Connext DDS 5.2.7 or 5.3.0 needs to communicate with a C/C++/.NET application

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

built with 5.2.6, 5.2.5, or 5.2.2, use the **-use526Keyhash** flag to generate code for your application. Use **use52Keyhash** to communicate with applications built with Connext DDS 5.2.3 or prior GARs.

The following tables clarify what flags to use to generate code for the versions in the first column in order to be compatible with previous releases.

Table 3.2 C/C++ Backward-Compatibility Flags for Keyed Mutable Types

	5.2.2/5.2.5/5.2.6	5.2.3 and Prior GARs
5.2.2/5.2.5/5.2.6	Compatible	Compatible
5.2.7/5.3.0 and future GARs	-use526Keyhash	-use52Keyhash

Table 3.3 Java Backward-Compatibility Flags for Keyed Mutable Types

	5.2.2/5.2.5/5.2.6	5.2.3 and Prior GARs
5.2.2/5.2.5/5.2.6	Compatible	-use52JavaKeyhash
5.2.7/5.3.0 and future GARs	Compatible	-use52Keyhash

Table 3.4 .NET Backward-Compatibility Flags for Keyed Mutable Types

	5.2.2/5.2.5/5.2.6	5.2.3 and Prior GARs
5.2.2/5.2.5/5.2.6	Compatible	-use52JavaKeyHash
5.2.7/5.3.0 and future GARs	-use526Keyhash	-use52Keyhash

3.4.4 IDL String Wire Compatibility

Starting with Connext DDS 5.3.0, the expected wire encoding for IDL strings has changed from ISO-8859-1 to UTF-8.

Because of this change, Java and .NET applications built using an older Connext DDS version may receive wrong values for IDL strings if the strings contain non-ASCII characters and they are published by applications built using 5.3.0 or higher. The opposite is also true.

To be backward compatible, you can configure the IDL wire encoding back to ISO-8859-1 as follows:

- For generated code TypePlugins and builtin types, set the value of the following properties to ISO-8859-1.

  - DataReader: **dds.data_reader.type_support.cdr_string_encoding_kind**
  - DataWriter: **dds.data_writer.type_support.cdr_string_encoding_kind**
- For DynamicData set the member **string_character_encoding** in DynamicDataProperty_t to the following value:
  - For Java: **StandardCharsets.ISO_8859_1**
  - For .NET: **StringEncodingKind::ISO_8859_1**

3.4.5 Compatibility Between IDL, XML, and XSD Generated with RTI Connext DDS 5.3 and Previous Versions

- IDL

  To be compliant with the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification, Connext DDS 5.3.0 adds support for prefix annotations. The IDL files obtained using the **-convertToIDL** option using Code Generator 2.5.0 (the version included with Connext DDS 5.3.0) will fail to compile with previous versions of Code Generator if annotations are used.

  For example, the following XML file will generate the IDL seen below.

  XML:
  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="rti_dds_topic_types.xsd">
      <struct name= "MyType">
          <member name="m1" type="int32" key="true"/>
      </struct>
  </types>
  ```

  IDL:
  ```
  @appendable
  struct MyType {
      @key long m1;
  };
  ```

  The above IDL will fail to compile with previous versions of Code Generator.
- XML

  XML files obtained using Code Generator 2.5.0 (the version included with Connext DDS 5.3.0)and the **-convertToXML** option may fail to compile with previous versions of Code Generator due to the following changes, which were made to align with the latest "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes)" specification.
  - Changes in the XML Type Representation for Basic Types (RTI Issue ID CODEGENII-493)

    For example, for the above long member, Code Generator 2.5.0 generates:

```
<member name="m1" type="int32" key="true"/>
```

while previous versions generated:

```
<member name="m1" type="long" key="true"/>
```

- Changes in default value name for Extensibility annotation

  In 2.5.0, **rti_dds_topic_types.xsd** specifies appendable as the default extensibility. This is not recognized by earlier versions of Code Generator.

Important: In addition, the parsing of XML files generated by Code Generator 2.5.0 may fail in infrastructure services (such as RTI Routing Service and RTI Recording Service) from previous Connext DDS versions that accept XML type definitions.

- XSD

  XSD files obtained using Code Generator 2.5.0 (the version included with Connext DDS 5.3.0) and the **-convertToXSD** option may fail to compile or may produce invalid source code with previous versions of the Code Generator because some of the annotations have changed the name to be compliant with the latest "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes)" specification.

  For example, for a member type that is external (pointer) like this one in IDL:

```
@external long m1;
```

Code Generator 2.5.0 will generate the following:

```
<xsd:element name="m1" minOccurs="1" maxOccurs="1" type="xsd:int"/>
<!-- @external true -->
```

while previous versions generated :

```
<xsd:element name="m1" minOccurs="1" maxOccurs="1" type="xsd:int"/>
<!-- @pointer true -->
```

3.5 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*.

3.6 Transport Compatibility

3.6.1 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**.

3.6.2 Transport Compatibility for Connext DDS 5.1.0

3.6.2.1 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.

3.6.2.2 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>
```

3.6.2.3 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.

Table 3.5 UDPv4, UDPv6, WAN, and TCP and Table 3.6 Shared Memory show the new default transport settings.

Table 3.5 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 3.6 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

3.6.2.4 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]**.

3.6.3 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.)

3.6.3.1 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: ""
```

3.6.3.2 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="BuiltinQosLib::Generic.510TransportCompatibility">
...
</qos_profile>
```

3.6.4 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.
```

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>
```

[a]RTI Issue ID CORE-6777

3.7 Other Compatibility Issues

3.7.1 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.

3.7.2 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.

3.7.3 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]

3.7.4 Linking with Libraries for Windows Platforms

Starting with Connext DDS 5.2.5, all Connext DDS libraries for Windows platforms (static release/debug, dynamic release/debug) now link with the dynamic Windows C Run-Time (CRT). Previously, the static Connext DDS libraries statically linked the CRT.

If you have an existing Windows project that was linking with the Connext DDS static libraries, you will need to change the RunTime Library settings:

- In Visual Studio, select C/C++, Code Generation, Runtime Library and use Multi-threaded DLL (**/MD**) instead of Multi-threaded (**/MT**) for static release libraries, and Multi-threaded Debug DLL (**/MDd**) instead of Multi-threaded Debug (**/MTd**) for static debug libraries.
- For command-line compilation, use **/MD** instead of **/MT** for static release libraries, and **/MDd** instead of **/MTd** for static debug libraries.

In addition, you may need to ignore the static run-time libraries in their static configurations:

- In Visual Studio, select **Linker, Input** in the project properties and add **libcmtd;libcmt** to the **'Ignore Specific Default Libraries**' entry.
- For command-line linking, add **/NODEFAULTLIB:"libcmtd" /NODEFAULTLIB:"libcmt"** to the linker options.

3.7.5 Time Conversion

In Connext DDS 5.1.0, the methods that convert the API time representations (DDS_Time_t and DDS_Duration_t) to the internal time representation (NTP timestamp) changed to address a conversion issue (CORE-5926).

Because of this change, in some cases, it is possible to get a QoS mismatch between publishing applications using Connext DDS 5.1.0 and later with subscribing applications using previous Connext DDS versions for the following QoS policies:

- DeadlineQosPolicy
- LivelinessQosPolicy

The QoS mismatch occurs when you set the time value of the above QoSs to be the same on the *DataWriter* and *DataReader* sides. The workaround consists of setting the time value in the Publishing application to be at least one nanosec smaller than the time value in the Subscribing application.

[RTI Issue ID CORE-7468]

3.7.6 Behavior Change for Config Logger's finalize_instance() – Traditional C++ API Only

Release 5.3.0.2 and higher introduces a behavior change in **NDDSConfigLogger::finalize_instance()** (Traditional C++ API only).

Prior to 5.3.0.2, setting an output device, calling **finalize_instance()**, and obtaining the logger instance again preserved the output device. Now it doesn't. **finalize_instance()** now resets the output device, if any.

[RTI Issue ID CORE-8150]

Chapter 4 Migration

4.1 Transitioning to Connext DDS 5.3.1

4.1.1 Code Generation

In releases 5.3.0.7, 5.3.0.8, and 5.3.1, the generated code for C, C++, and C++03/C++11 (without STL) was changed in order to fix a potential memory leak (see RTI Issue ID CORE-8271).

C, C++, and C++03/C++11 (without STL) applications moving from a previous Connext DDS version to Connext DDS 5.3.1 require code regeneration and recompilation of the application using Connext DDS.

4.1.2 Application Binary Interface Compatibility

Connext DDS 5.3.1 does not provide Application Binary Interface (ABI) compatibility with previous versions of Connext DDS, including Connext DDS 5.3.0.

An application compiled with a previous version of Connext DDS must be recompiled when moving to Connext DDS 5.3.1.

4.2 Transitioning to Connext DDS 5.3

4.2.1 Code Generation

For applications defining types in IDL, XML, or XSD, moving from a previous Connext DDS version to Connext DDS 5.3 requires code regeneration and recompilation of the applications using Connext DDS.

4.2.2 Application Binary Interface Compatibility

Connext DDS 5.3 does not provide Application Binary Interface (ABI) compatibility with previous versions of Connext DDS.

An application compiled with a previous version of Connext DDS must be recompiled when moving to Connext DDS 5.3.

4.2.3 Compatibility

Connext DDS 5.3 breaks backward compatibility with previous Connext DDS releases for some features. For information on backward-compatibility issues, see Compatibility (Chapter 3 on page 8).

4.3 Transitioning to Connext DDS 5.2 or Higher

Please read this before installing. If you are transitioning to 5.2 or higher, 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 5 What's Fixed in 5.3.1

Release 5.3.1 is a maintenance release based on the feature release 5.3.0. This section describes bugs fixed in 5.3.1. These fixes have been made since 5.3.0.

5.1 Fixes Related to Content Filters and Query Conditions

5.1.1 Error Evaluating Query Condition for Unkeyed DataReaders in Some Cases

This issue only affected Connext DDS 5.3.0.

If you created a Query Condition on a *DataReader* after some samples had been received, the evaluation of the Query Condition for the samples on the *DataReader* cache were incorrect. Therefore, **take_w_condition()** or **read_w_condition()** may not have provided samples that should pass the Query Condition.

In addition, the creation of the Query Condition produced an error like this:

```
[D0406|Reader(80000004)|T=MyTopic|CREATE READCONDITION]RTICdrStream_
getCurrentPosition:!precondition: me == ((void *)0) || me->_currentPosition == ((void
*)0) || me->_buffer == ((void *)0)
[D0406|Reader(80000004)|T=MyTopic|CREATE READCONDITION]RTICdrStream_
getRelativeCurrentPositionOffset:!precondition: me == ((void *)0) || me->_
currentPosition == ((void *)0) || me->_relativeBuffer == ((void *)
[D0406|Reader(80000004)|T=MyTopic|CREATE READCONDITION]RTICdrStream_
appendFull:!precondition: me->_currentPosition == ((void *)0)
[D0406|Reader(80000004)|T=MyTopic|CREATE READCONDITION]PRESPsReaderQueue_
evaluateEntryForQueryCondition:serialize failed
[D0406|Reader(80000004)|T=MyTopic|CREATE READCONDITION]PRESPsReaderQueue_
initializeQueryConditionInventory:!evaluate sample for query condition
```

This issue only applied to unkeyed *DataReaders* using DynamicData or in the following languages: Java, .NET, and Modern C++ using stl TypePlugin.

This problem has been resolved.

[RTI Issue ID CORE-8188]

5.1.2 Only First Created Key-Only Query Condition was Evaluated

A key-only filter expression only filters based on key fields of a type. When multiple Query Conditions were created with key-only filter expressions, only the first one created was correctly evaluated against the samples in the DataReader queue. For all the other Query Conditions, the result may have been incorrect. This means your application may have received samples it shouldn't have, or may have not received samples it should have. All non-key-only filters behaved as expected. This problem has been resolved.

[RTI Issue ID CORE-8214]

5.1.3 Incorrect Results if Query Condition Parameters Changed while Samples Received but not Committed to Reader Queue

In rare situations it may have been possible to receive incorrect results for a Query Condition. For this situation to occur, the following conditions must have been met:

- The Topic associated with the Query Condition was created in Java, with the new Modern C++ Type Plugin or through the Dynamic Data API.
- Samples were received out of order due to, for example, network congestion or a late-joining *DataReader* receiving live data at the same time as historical data.
- The parameters of an existing Query Condition were changed at the same time that samples were being received out of order. If the parameters were changed before any samples were received by a *DataReader*, this issue would not have been triggered.

This problem has been resolved.

[RTI Issue ID CORE-8222]

5.1.4 Recurrent Deserialization Failure for Unkeyed Sample Prevented Communication when Using QueryCondition or ContentFilter

When a reliable *DataReader* fails to process a sample due to an error in deserialization, the *DataReader* does not deliver the sample to the application. After this rejection happens, the correct behavior is to skip the sample instead of asking for a repair, because there are scenarios (for example, when using DDS-XTypes or an incorrect TypePlugin code) in which this sample will always fail to be deserialized.

There used to be some scenarios, however, that provoked the reliable *DataReader* to incorrectly ask the *DataWriter* to repair this sample. These scenarios were when the *DataReader* was using a *QueryCondition* or ContentFilteredTopic. This problem has been resolved.

[RTI Issue ID CORE-8402]

5.1.5 Crash when Unregistering Custom Content Filters

A crash would occur when unregistering a custom content filter from a *DomainParticipant* that contained a *DataWriter*, if the *DataWriter* was performing writer-side filtering for multiple *DataReaders* that use different content filters (custom or built-in). This problem has been resolved.

[RTI Issue ID CORE-8392]

5.1.6 Potential Crash on Unkeyed DataReaders when Using Query Conditions and KEEP_LAST HistoryQosPolicy

Continuously creating and deleting new QueryConditions for the same *DataReader* could have triggered a crash upon receiving a sample.This issue affected only unkeyed DataReaders using the KEEP_LAST HistoryQosPolicy. This problem is now resolved.

[RTI Issue ID CORE-8372]

5.1.7 Possible Memory Leak if DataWriter used Writer-Side Filtering and Published Topic with Optional Members–C and Traditional C++ APIs Only

A *DataWriter* may have leaked memory when using writer-side filtering if its *Topic's* data type contained optional members. This only occurred for C and traditional C++ languages when the data type did not use inheritance. This problem has been resolved.

Note: This fix requires regeneration and recompliation of the application using Connext DDS. See Transitioning to Connext DDS 5.3.1 (4.1 on page 33).

[RTI Issue ID CORE-8271]

5.1.8 DataReader using ContentFilteredTopic may not have Received DISPOSE/UNREGISTER Samples from DataWriter using Writer-Side Filtering

A *DataReader* using a ContentFilteredTopic may not have received DISPOSE/UNREGISTER samples from a *DataWriter* using writer-side filtering.

This issue only occurred for repair data (data that had to be resent to the *DataReader* after the *DataReader* sent a NACK) and when **writer_qos.writer_resource_limits.max_remote_reader_filters** was set to UNLIMITED (the default value starting in 5.3.0). When this occurred, the *DataWriter* printed errors such as:

```
PRESWriterHistoryDriver_requestData:!evaluate filter
PRESWriterHistoryDriver_evaluateFilter:!deserialize encapsulation
PRESWriterHistoryDriver_requestData:!evaluate filter
PRESWriterHistoryDriver_evaluateFilter:deserialize sample error in topic 'MyTopic' with type
'MyType'
```

```
PRESWriterHistoryDriver_requestData:!evaluate filter
```

This problem has been resolved.

[RTI Issue ID CORE-8297]

5.1.9 Performance Degradation when Using Filters on ValueTypes (Modern and Traditional C++ APIs)

Applications that filtered topic types defined as the IDL "valuetype" may have experienced decreased performance since Connext DDS 5.2.0. Note that simply declaring the type as "struct" resulted in normal performance. This problem affected the modern and traditional C++ APIs. This problem has been resolved.

[RTI Issue ID CORE-8430]

5.2 Fixes Related to TopicQueries

5.2.1 Unbounded Memory Growth when Continuously Creating/Deleting TopicQueries

Continuously creating/deleting TopicQueries caused unbounded memory growth. Some TopicQuery memory was not released after the TopicQuery was deleted with **DDSDataReader::delete_topic_query ()**. This problem has been resolved.

[RTI Issue ID CORE-8175]

5.2.2 DataWriterListener::on_service_request_accepted Reported Invalid last_request_handle after TopicQuery Deleted

The status provided on the listener callback **DataWriterListener::on_service_request_accepted** reported the wrong **last_request_handle** after a TopicQuery was deleted. This problem has been resolved.

[RTI Issue ID CORE-8230]

5.2.3 WaitSet may have Woken up with Active QueryConditions but no Data

In the presence of TopicQueries, a WaitSet may have woken up with active QueryConditions for which there was no data. This problem has been resolved.

[RTI Issue ID CORE-8398]

5.2.4 MultiChannel and TopicQuery did not Work with Large Data

A MultiChannel *DataWriter* or a *DataWriter* configured to dispatch TopicQueries (by setting **writer_qos.topic_query_dispatch.enable** to true) was not able to send large data (that is, samples bigger than the transport **message_size_max**). The write() call would fail with RETCODE_ERROR. This problem has been resolved.

[RTI Issue ID CORE-8422]

5.3 Fixes Related to Discovery

5.3.1 Potential Deadlock Risk Errors when Rediscovering Remote Participant

There was a rare race condition that may have resulted in deadlock risk error messages. In particular, this issue may have been triggered when rediscovering a remote participant that had been just marked for removal from the local participant. This problem has been resolved.

[RTI Issue ID CORE-8164]

5.3.2 Reader did not Discover Writer when MultiChannel Enabled

Enabling MultiChannel on a *DataWriter* may have incorrectly triggered discovery deserialization errors on remote subscribers. In particular, this issue may have occurred when a *DataWriter's* number of channels and filter expressions lengths were close to (but not exceeding) the limits defined by remote participant's **channel_seq_max_length** and **channel_filter_expression_max_length**. When this issue triggered, the following error was logged on the subscriber side, preventing discovery from occurring:

```
PRESCstReaderCollator_storeSampleData:deserialize sample error in topic 'DISCPublication' with
type 'DISCPublicationParameter'
```

This problem has been resolved.

[RTI Issue ID CORE-8304]

5.3.3 No Communication between DataWriters and DataReaders upon Changing Partition QoS

In previous releases, there was an issue that may have prevented communication between *DataWriters* and *DataReaders* using non-default values for the Partition QoS policy. In particular, this issue may have been triggered by changing the Partition QoS configuration for a *Publisher* or *Subscriber* containing multiple endpoints configured with different values for the EntityName QoS policy. This problem has been resolved.

[RTI Issue ID CORE-8323]

5.3.4 Failure to Receive Builtin Topic Data Containing Unknown DDS::ServiceQosPolicyKind Values

Previous versions of Connext DDS failed to receive builtin topic data containing DDS::ServiceQosPolicyKinds that were introduced in newer versions of RTI Connext DDS. The following error was printed:

```
DDS_ServiceQosPolicy_from_presentation_service_kind:ERROR:Failed to get service (unknown kind)
```

```
DDS_SubscriptionBuiltinTopicDataTransform:ERROR:Failed to get service
PRESCstReaderCollator_addSample:!transform
```

Starting with version 5.3.0.7, DDS::ServiceQosPolicyKind will be set to DDS::NO_SERVICE_QOS and the builtin topic data will not fail to be received.

[RTI Issue ID CORE-8245]

5.4 Fixes Related to DynamicData, TypeCode, and TypeObjects

5.4.1 Error Loading XML Configuration File Containing Type that Inherited from Empty Structure

Loading an XML file containing a type that inherited from an empty structure failed. For example:

```
<struct name= "StructBase">
</struct>
<struct name= "StructDerived" baseType="StructBase">
    <member name="m1" type="boolean"/>
</struct>
```

Parsing the above XML snippet would have failed with the following error:

```
DDS_TypeCodeFactory_initialize_value_tcI:!get member ID
DDS_XMLValueType_initialize:!create valuetype typecode
DDS_XMLValueType_new:!init XML valuetype object
RTIXMLParser_onStartTag:Parse error at line 12: Error processing tag 'struct'
RTIXMLParser_parseFromFile_ex:Parse error in file 'HelloWorld.xml'
DDS_XMLTypeCodeParser_parse_from_file:Error parsing XML
DDS_XMLInclude_initialize:Parse error at line 4: error parsing 'HelloWorld.xml'
DDS_XMLInclude_new:!init XML include object
RTIXMLParser_onStartTag:Parse error at line 4: Error processing tag 'include'
RTIXMLParser_parseFromFile_ex:Parse error in file 'routertest.xml'
DDS_XMLParser_parse_from_file:Error parsing file
ROUTERCfgFileParser_loadCfgFile:error parsing configuration file 'routertest.xml'
ROUTERService_initialize:!load configuration file
ROUTERService_new:!init service
main:!create router service
```

This problem has been resolved. It affected any product/feature that can receive XML type definitions as part of its configuration, such as RTI Routing Service.

[RTI Issue ID CORE-8157]

5.4.2 Error Creating Valuetype/Structure Typecode that Inherited from Empty Valuetype/Structure

Creating a valuetype/struct typecode using the API **DDS:TypeCodeFactory::create_value_tc()** failed if the concrete_base was a typecode representing an empty structure/valuetype. This problem has been resolved.

[RTI Issue ID CORE-8160]

5.4.3 DynamicData::get_string() Allocated Large Amount of Memory when Retrieving Zero-Length String

The **DynamicData::get_string()** API incorrectly allocated 4,294,967,296 bytes of memory when returning a 0-length string. This situation could only occur if a *DataWriter* using generated types explicitly set a non-optional string to NULL (instead of leaving it at the default value of an empty string) and a DynamicDataReader read that sample and retrieved the 0-length string. There is no way to set or send a 0-length string from the DynamicData API itself.

This problem has been resolved. The **DynamicData::get_string()** API now correctly allocates only 1 byte, for the null-terminator, when retrieving a 0-length string.

[RTI Issue ID CORE-8162]

5.4.4 DynamicData Endpoint did not Send/Receive Samples from Generated Endpoint if Type used member_id Greater than 65535

If a type containing a member with a **member_id** value greater than 65535 was used, a DynamicData *DataReader* was unable to receive samples from a generated *DataWriter*. This problem also occurred when a DynamicData *DataWriter* was used to send samples of this type to a generated *DataReader*. This problem has been resolved.

[RTI Issue ID CORE-8220]

5.5 Fixes Related to Transports

5.5.1 UDPv4 Multicast Communication Interrupted if Physical NIC Disabled

This issue did not get fixed in release 5.3.0, but it is fixed in release 5.3.1. See 6.7.3 UDPv4 Multicast Communication Interrupted if Physical NIC Disabled on page 64 for details.

[RTI Issue ID CORE-6908]

5.5.2 Potential Double Free upon UDPv4 Transport Creation Failure—INTEGRITY and VxWorks 653 Platforms Only

There was an issue that may have lead to a double free upon a failure during UDPv4 transport creation. This problem, which only affected INTEGRITY and VxWorks 653 platforms, has been resolved.

[RTI Issue ID CORE-7006]

5.5.3 Possible Segmentation Fault when Using Shared Memory on VxWorks

When using the shared memory transport on VxWorks and repeatedly creating and deleting a DomainParticipant, a segmentation fault was possible in the function RTIOsapiSharedMemorySegment_createOrAttach. This problem has been resolved.

[RTI Issue ID CORE-8324]

5.5.4 Potential Transport Send Failure when Fragmenting Data

There was an issue that may have caused a transport send failure when fragmenting data. This problem has been resolved.

[RTI Issue ID CORE-8253]

5.5.5 Error Negotiating TCP Transport Connection may have Prevented Communication

In previous releases, an error during TCP Transport connection negotiation (for example, as a result of getting an EAGAIN socket error on the sender) may have lead to an unrecoverable state, preventing communication between two participants. This problem is now resolved.

As part of the fix, a new TCP Transport property has been added:

- **client_connection_negotiation_timeout:** This property controls the maximum time (in seconds) a client data connection negotiation can remain in progress. If the negotiation of a connection has not completed after the specified timeout, the negotiation will restart, and if there is an associated data connection, it will be closed. This way, the TCP Transport plugin can retry the process of establishing and negotiating that connection.

And the following property has been adjusted to apply to any TCP client connection (instead of being restricted to enabling TLS):

- **initial_handshake_timeout:** Once a connection is established, TCP Transport will exchange some information to identify itself and the connection. This process is known as the initial handshake of a connection, and if using TLS the TCP Transport plugin will also exchange additional information to secure the connection. This property controls the maximum time (in seconds) the initial handshake for a connection can remain in progress. 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.

For more information, refer to Section 37.6 TCP/TLS Transport Properties in the *RTI Connext DDS Core Libraries User's Manual*.

[RTI Issue ID COREPLG-356]

5.5.6 Improved CRC Error Handling for TCP Transport Control Connections

This release introduces changes to improve how the TCP Transport handles a CRC error on the TCP Transport control connections.

In previous releases, a CRC error on a control connection triggered the close of the connection and cleaning up of the state between the TCP Transport client and server, making it necessary to open and negotiate again all the connections—a costly process.

This release changes this behavior to just drop the corrupted message. This way, the dropped message can be re-sent without restarting the entire communication between the client and the server, resulting in a more stable communication when message corruption occurs.

[RTI Issue ID COREPLG-433]

5.5.7 Possible Deadlock or Segmentation Fault in TCP Transport

An application using the TCP Transport may have deadlocked or had a segmentation fault during the exchange of control messages over the transport.

This problem has been resolved.

[RTI Issue ID COREPLG-437]

5.6 Fixes Related to Modern C++ API

5.6.1 Heap Monitoring Utility could not be Enabled in main() – Modern C++ API Only

The heap monitoring utility included in 5.3.0 could not be enabled in **main()**. This utility needs to be enabled before any middleware allocation, but a global static variable initialized before **main()** allocated memory. The only workaround was to enable heap monitoring as part of the constructor of another global static variable.

This problem has been resolved; now you can enable monitoring in **main()** like this:

```
int main(int argc, char** argv)
{
    rti::util::heap_monitoring::enable();
    // ...
}
```

[RTI Issue ID CORE-8178]

5.6.2 Some Functions in Default QosProvider not Thread-Safe–Modern C++ API Only

The default QosProvider (**dds::core::QosProvider::Default()**) shares some internal state with the DomainParticipantFactory—an implicit entity in this API. Some operations were not mutually protected for multi-threaded access. For example, two threads running the following two operations may have caused undefined behavior, such as a crash:

- **QosProvider::Default()->default_library()**
- **DomainParticipant::participant_factory_qos()**

This problem has been resolved. Now it is safe to access the default QosProvider and operations related to the implicit DomainParticipantFactory in parallel.

[RTI Issue ID CORE-8246]

5.6.3 Default Constructor for safe_enum's did not Initialize Underlying Integer–Modern C++ API Only

The default constructor for dds::core::safe_enum left the underlying integer uninitialized, which may have caused static-code-analysis tools to report an error. To prevent this, the constructor has been modified to initialize the enumeration to zero.

Note that zero may still be an invalid enumerator. It is always recommended to initialize all enumeration variables with a valid enumerator.

[RTI Issue ID CORE-8265]

5.6.4 DomainParticipant::finalize_participant_factory may have not Released all Resources–Modern C++ API only

An application that used **QosProvider::Default()** and then called **DomainParticipant::finalize_participant_factory()** at the end may not have released all resources. You may have seen some memory leaks as the application exited. A second call to **DomainParticipant::finalize_participant_factory()** would have solved the problem.

This problem affected only 5.3.0.7. This problem has been resolved in release 5.3.1.

[RTI Issue ID CORE-8463]

5.7 Fixes Related to Logging

5.7.1 Changing Logging QoS Policy before Creating DomainParticipant may have Stopped Redirecting Log Messages

When an application redirected the *RTI Connext* log messages to a custom device and then changed the Logging QoS policy (in the DomainParticipantFactory), log messages may have been directed to the standard output instead of to that device until the first *DomainParticipant* was created. This problem only affected 5.3.0 and has been resolved in this release.

[RTI Issue ID CORE-8134]

5.7.2 Memory Leak when Application Registered Logging Device and Ended without Creating DomainParticipant

An application that registered a custom logging device (with **NDDSConfigLogger::set_output_device()** and then ended without creating any DomainParticipants may have shown a small non-recurrent memory leak. Valgrind reported it as still-reachable memory. The problem only affected 5.3.0 and has been resolved in this release.

[RTI Issue ID CORE-8150]

5.8 Fixes Related to XML-Based Application Creation

5.8.1 <register_type> Ignored Deprecated 'kind' Attribute if Built-in Type was Specified

Creating a Topic with XML Application Creation failed if the associated type was specified through a <register_type> tag with the deprecated kind attribute set to the value of a built-in type. This resulted in backward incompatiblity with prior versions. This problem only affected 5.3.0 and has been resolved in this release.

[RTI Issue ID CORE-8151]

5.8.2 Creating DomainParticipant from Configuration Mistakenly Succeeded Even Though Creating its Contained DataWriters or DataReaders Failed

Creating a *DomainParticipant* from a configuration (e.g., through **create_participant_from_config()**) should fail if any of its contained *DataWriters* or *DataReaders* cannot be created. In the previous release, the *DomainParticipant* was created anyway. (However, looking up its *DataWriters* or *DataReaders* by name did fail, as expected.)

This problem has been resolved. Now the *DomainParticipant* creation will fail if any of its contained *DataReaders* or *DataWriters* cannot be created.

[RTI Issue ID CORE-8152]

5.8.3 Deadlock when Using Some APIs and Monitoring Simultaneously

Using the following APIs while Monitoring was enabled may have caused a deadlock in the middleware:

```
DDS_DomainParticipantFactory::lookup_participant_by_name()
DDS_DomainParticipantFactory::create_participant_from_config()
DDS_DomainParticipantFactory::create_participant_from_config_w_params()
```

This problem has been resolved.

[RTI Issue ID CORE-8252]

5.9 Fixes Related to Vulnerabilities

This release fixes some potential vulnerabilities and memory corruption when receiving malformed data (including RTI Issue IDs CORE-8396, CORE-8251, CORE-8155, CORE-8038).

5.10 Other Fixes

5.10.1 Crash when Sending AppAck when Remote Endpoint not Alive

Sending an AppAck message for an endpoint that had not fully completed discovery, or whose liveliness had been lost, may have triggered a crash. This problem has been resolved.

[RTI Issue ID CORE-8121]

5.10.2 Potential Inconsistent Behavior or Crash

There was a very rare race condition that may have left the internal Connext DDS database in an inconsistent state. This problem has been resolved.

[RTI Issue ID CORE-8124]

5.10.3 XSD Validation Failed for QoS Profile Files Referring to rti_dds_qos_profiles.xsd

The **rti_dds_qos_profiles.xsd** file included in Connext DDS 5.3.0 did not include the right product version in the version attribute of the DDS tag. Consequently, well-formed QoS profile XML files failed validation against the XSD. This problem has been resolved.

[RTI Issue ID CORE-8233]

5.10.4 Potential Segmentation Fault when Using Multichannel Feature

A subscriber application may have issued a segmentation fault when receiving channel information from a *DataWriter* with one or more channels set in the MultiChannelQosPolicy. This problem has been resolved.

[RTI Issue ID CORE-8238]

5.10.5 Potential Crash during Reliable Writer Deletion

There was an issue that may have triggered a crash during the deletion of a reliable local writer. This problem has been resolved.

[RTI Issue ID CORE-8255]

5.10.6 rti_dds_topic_types.xsd and rti_dds_profiles.xsd Schemas did not Allow Valuetype Modifier

The rti_dds_topic_types.xsd and rti_dds_profiles.xsd schemas provided with release 5.3.0 had an error and did not allow valuetype modifiers such as custom, abstract, or trucatable. You may have seen an error when validating your XML with these schemas if those modifiers were used. This problem has been resolved.

[RTI Issue ID CORE-8267]

5.10.7 Possible Crash after Failing to Create DataWriter or DataReader

When **create_datawriter()** or **create_datareader()** failed, certain failure conditions caused a segmentation fault in the release libraries when calling **DomainParticipant_delete_contained_entities()**.

When using the debug libraries, create_datawriter() or create_datareader() logged this error message:

```
REDACursor_removeRecord:!precondition: !( (c!=((void *)0)) && ((c->_state) & 0x04) && ((c->_
state) & 0x08) )
```

One such failure condition was when using a custom plugin implementation of DDS Security in which **register_local_datawriter()** or **register_local_datareader()** returned NULL.

This problem has been resolved.

[RTI Issue ID CORE-8269]

5.10.8 write() may have Returned RETCODE_OK Even if it Failed

There were some situations in which the **write()** operation failed due to a misconfiguration, and although an error message was logged, **write()** didn't return the error code that indicates a failure (RETCODE_ERROR). You may have seen this error message :

```
MIGGenerator_addDataBatch:serialize buffer too small
```

This problem has been resolved. Now **write()** will return RETCODE_ERROR after logging the error message.

[RTI Issue ID CORE-8291]

5.10.9 Internal Symbol Redeclaration when Using diab Compiler

When using the diab compiler to build an application for the architecture ppce6500Vx6.9.4.6diab5.9.1, the following warning was displayed:

```
dld: warning: Redeclaration of ADVLOGLogger_g_TimestampClock
Defined in Logger.o(<NDDSHOME>/lib/ppce6500Vx6.9.4.6diab5.9.1/libnddscorez.a)
and LoggerFormat.o(<NDDSHOME>/lib/ppce6500Vx6.9.4.6diab5.9.1/libnddscorez.a)
```

This problem has been resolved.

[RTI Issue ID CORE-8355]

5.10.10 WaitSet May Not Have Been Triggered for Active ReadConditions

In some cases, for TopicQuery data, a WaitSet may not have been triggered, although there was data for some of the attached ReadConditions (or QueryConditions). This problem has been resolved.

[RTI Issue ID CORE-8407]

5.10.11 Error When Starting in an Environment with No Network Stack

A Connext DDS application that was started in an environment with no network stack printed out the following error message, even if the application was configured to use only shared memory transport:

```
RTIOsapi_getFirstValidInterface:!create socket
```

The verbosity of this benign error message has been lowered to a warning level verbosity.

[RTI Issue ID CORE-8417]

5.10.12 Monotonic Clock not Supported for Ubuntu 16.04 Platforms

The monotonic clock feature was mistakenly disabled for Ubuntu 16.04 Intel platforms. This feature is now enabled in this release.

[RTI Issue ID PLATFORMS-1101]

5.10.13 DomainParticipant Information not Published when Created with rtps_auto_id_kind DDS_RTPS_AUTO_ID_FROM_UUID

RTI Connext Monitoring failed to publish the data of a *DomainParticipant* that was created with the following QoS setting:

```
WireProtocolQosPolicy::rtps_auto_id_kind = DDS_RTPS_AUTO_ID_FROM_UUID
```

The following message was printed:

```
DDSMonitoring_WireProtocolQosPolicyAutoKindPlugin_serialize:invalid enum value 2 for enumerator
DDSMonitoring_WireProtocolQosPolicyAutoKind
```

This QoS setting was introduced in 5.3.0 as part of support for IP mobility.

This problem has been resolved.

[RTI Issue ID MONITOR-226]

Chapter 6 What's Fixed in 5.3.0

Release 5.3.0 is a general access release based on the maintenance release 5.2.3.[a] This section describes bugs fixed in 5.3.0.

For an overview of new features and improvements in 5.3.0, please see the separate ***What's New*** document for 5.3.0.

6.1 Fixes Related to Content Filters and Query Conditions

6.1.1 Possible Segmentation Fault when Writer-Side Filtering for DataReader with More Than Four Locators

A publisher application may have issued a segmentation fault when doing writer-side filtering for a *DataReader* with more than 4 locators. This only occurred in best-effort configurations. This problem has been resolved.

[RTI Issue ID CORE-6947]

6.1.2 Possible Multi-Threaded Race Condition and Crash after DataWriter Exceeds Resource Limits

A DDS application may have crashed due to a race condition when multiple threads called **write()** on the same reliable, keep-all *DataWriter* communicating with a *DataReader* with a ContentFilteredTopic.

The problem may have happened after one thread blocked on **write()** due to the **max_samples** or **max_instances** resource limits, and a second thread attempted to call **write()** before the first one had woken up.

---

[a]For What's Fixed in 5.2.3, see the *RTI Connext DDS Core Libraries Release Notes* provided with 5.2.3.

As an example, a .NET DDS application running into this problem could have thrown the following exception:

```
Unhandled Exception: System.AccessViolationException: Attempted to
read or write protected memory. This is often an indication that other memory
is corrupt. at DDS.DataWriter.write_untyped(Object instance_data,
InstanceHandle_t& handle)
```

This problem has been resolved; the multi-threaded use of a *DataWriter* is now safe as described in the API documentation.

[RTI Issue ID CORE-3336]

6.1.3 Content Filter Expression did not Provide Way to Check for Unset Optional Members

Content filter expressions that included optional members could be used to check the value of those members, but there was no way to check if the member was unset.

For example, if **my_optional** is an optional member in the topic being filtered, this is legal:

```
"my_optional = 5"
```

A ContentFilteredTopic defined with the above expression would filter out samples in which **my_optional** was unset or was set to a value other than 5. There was, however, no syntax to simply verify if **my_optional** was set or unset.

This release introduces a new keyword, **null**, which allows you to check if a member is set or unset. The following expressions are now supported:

```
"my_optional = null"
"my_optional <> null"
```

[RTI Issue ID CORE-7265]

6.1.4 DataReader using ContentFilteredTopic may not have Matched with DataWriter

In rare situations, a *DataReader* using a ContentFilteredTopic may not have matched with a *DataWriter*. This behavior was more likely to occur in scenarios where a *DataWriter* was writing samples at a high frequency and in the meantime matching *DataReaders* using ContentFilteredTopic were created/destroyed.

A workaround was to disable writer-side filtering on the *DataWriter* by setting **writer_resource_limits.max_remote_reader_filters** to 0.

This problem has been resolved.

[RTI Issue ID CORE-7911]

6.1.5 DataWriter Segmentation Fault when Matching DataReader using ContentFilteredTopic Sets Long Filter Expression

A *DataWriter* matching with a *DataReader* using a ContentFilteredTopic (CFT) may segfault under this scenario:

1. The *DataReader* is created using a CFT where the combination of the expression and parameters does not exceed the DataWriter and DataReader Participants **resource_limits.contentfilter_property_max_length**.
2. The *DataReader* changes the CFT expression and/or parameters so that the *DataReader's* Participant **resource_limits.contentfilter_property_max_length** is not exceeded but the DataWriter's Participant **resource_limits.contentfilter_property_max_length** is exceeded.

This problem has been resolved. When the **resource_limits.contentfilter_property_max_length** is exceeded on the *DataWriter's* Participant, the *DataWriter* will not do writer-side filtering for the *DataReader*; content filtering will be done on the *DataReader* side.

[RTI Issue ID CORE-7918]

6.1.6 Writer-Side Filtering did not Work when Using Durable History and Setting writer_resource_limits.max_remote_reader_filters to Finite Value

Writer side filtering did not work when using durable history and setting **writer_resource_limits.max_remote_reader_filters** to a finite value.

If **writer_resource_limits.max_remote_reader_filters** was set to 32 or less, the results of the filter evaluation may have been wrong.

If **writer_resource_limits.max_remote_reader_filters** was set to 33 or more, the *DataWriter* may have caused a segmentation fault when more than 33 *DataReaders* using ContentFilteredTopics were matching with the *DataWriter*.

This problem has been fixed by not accepting finite values when using durable history. If you configure a finite value, the value will be ignored and the middleware will print a warning message like this:

```
finite max_remote_reader_filters not supported with durable writer history.
The value will be changed to UNLIMITED.
```

[RTI Issue ID CORE-7964]

6.1.7 Possible Content Filter Failure

In some situations, a *DataWriter* applying a writer-side Content Filter may have failed to evaluate a data sample, printing the following message:

```
DDS_SqlFilter_grow_deserialization_buffer:!sample exceeds maximum total length
DDS_SqlFilter_evaluateOnSerialized:deserialization error: sample
```

This problem may have only happened in scenarios where the following three statements were true:

- The Java or .NET API was used, or the C++ API was used with an IDL type that used inheritance;
- The type contained at least one long long, unsigned long long, double or long double member;
- The *DataWriter* was communicating with two or more *DataReaders* at the time of writing the sample.

This problem has been resolved.

[RTI Issue ID CORE-7977]

6.1.8 Incorrect Results if Query Condition Created while Samples have been received but not committed to Reader Queue

In rare situations it may have been possible to receive incorrect results for a Query Condition. For this situation to occur the following conditions had to be met:

- The Topic associated with the Query Condition was created in Java or through the Dynamic Data API.
- Samples were received out-of-order due to, for example, network congestion or a late-joining *DataReader* receiving live data at the same time as historical data.
- The Query Condition was created at the same time that samples were being received out-of-order. If the Query Condition was created before any samples were received by a *DataReader*, this issue would not have been triggered.

This problem has been resolved.

[RTI Issue ID CORE-7976]

6.2 Fixes Related to Asynchronous Publishers

6.2.1 Partial Support for DataWriterProtocolStatus Statistics when Publishing Asynchronously

Some DataWriterProtocolStatus statistics such as **pushed_sample_count** would not update when publishing data asynchronously. The following subset of DataWriterProtocolStatus statistics will now be updated when publishing samples using the asynchronous publisher:

1. pushed_sample_count
2. pushed_sample_count_change

3. pushed_sample_bytes
4. pushed_sample_bytes_change
5. pulled_sample_count
6. pulled_sample_count_change
7. pulled_sample_bytes
8. pulled_sample_bytes_change

pulled_sample_bytes will be updated when a whole fragmented sample, or a single fragment, gets repaired. **pulled_sample_count** will ONLY be updated when the whole sample, and its fragments, are repaired.

[RTI Issue ID CORE-7658]

6.2.2 Samples Published with DataWriter using Asynchronous Publication Mode Possibly not Sent

A race condition may have caused the samples published with a *DataWriter* using Asynchronous publication mode to not be sent.

This only occurred if the affected DataWriter was created on an empty Publisher after all the *DataWriters* that were contained on the *Publisher* were deleted. This problem has been resolved.

[RTI Issue ID CORE-7719]

6.2.3 Asynchronous Publisher's DDSThreadFactory::delete_thread() may have Blocked when Using DDSThreadFactory

Using a user-defined DDSThreadFactory to manage internal threads created by Connext DDS may have caused the **DDSThreadFactory::delete_thread()** operation for the Asynchronous Publisher thread to block or timeout if within that operation the user tried to wait (for instance, using join()) for the thread to finish.

[RTI Issue ID CORE-7886]

6.3 Fixes Related to Discovery

6.3.1 DiscoveryConfig's SEDP Rely On SPDP Only Prevents Simple Endpoint Discovery From Resuming

Simple Endpoint Discovery did not resume when calling the *DomainParticipant's* **resume_endpoint_discovery()** API if the DiscoveryConfig QoS policy's **sedp_rely_on_spdp_only** field was set to true. This problem has been resolved.

[RTI Issue ID CORE-7346]

6.3.2 No Communication Between Two Participants on Same Machine after Change in Network Interfaces

There was an issue that may have prevented communication between two Participants in the same machine when using shared memory and UDPv4 transports in the two Participants. In particular, this issue may have been triggered when each Participant was created when a different set of network interfaces was available in the system. This problem has been resolved.

[RTI Issue ID CORE-7680]

6.4 Fixes Related to DynamicData, TypeCode, and TypeObjects

6.4.1 TypeCode.equals() did not Check for Base Class Equality

The Java method TypeCode.equals() did not check the equality of the base class (if one existed). This resulted in inaccurate return values if the base classes differed. This problem has been resolved.

[RTI Issue ID CORE-7987]

6.4.2 Non-Standard TypeObject Representation of Modules Broke Interoperability

The serialized TypeObject that represents types inside a module is nonstandard. This may prevent Connext DDS from delivering or receiving type information (that is used, for example, to determine the assignability of two topics) from applications using other DDS implementations. This problem has been resolved.

[RTI Issue ID CORE-7624]

6.4.3 Some Fields in Dynamic Data Types not Set Properly

Some fields of complex types were not set properly using the Dynamic Data API. This problem has been resolved.

[RTI Issue ID CORE-6008]

6.4.4 Default Member of Union not Always Correctly Initialized by DynamicData API

If no default member is defined in a union, the default discriminator is the lowest value associated with any member. The DynamicData API incorrectly always chose the first member of a union as the default, regardless of the value of the discriminator.

You may have run into this problem when sending a sample containing a union that was not explicitly set. In this scenario, the union was initialized to its default value and sent; this default may have been different depending on whether generated code or DynamicData was used to send the data. This problem has been resolved.

[RTI Issue ID CORE-6365]

6.4.5 Dynamic Data Print Operation did not Print Unset Members at End of a Type

The **DDS_DynamicData_print()** operation did not display any data for members at the end of a data structure that had not been explicitly set before the operation was called. This problem has been resolved.

[RTI Issue ID CORE-6503]

6.4.6 Error Unbinding Complex Member using DynamicData API

When unbinding a complex member using the DynamicData API, you may have encountered the following error:

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

This may have occurred if members of the DynamicData object were set out of order. It may also have occurred if the DynamicData object being unbound had a nested, complex member that had its members set out of order. This problem has been resolved.

[RTI Issue ID CORE-6952]

6.4.7 Setting Members Out of Order in DynamicData Object may have Caused Data Corruption

In a number of scenarios, if members of a DynamicData object were set out of order, further use of the DynamicData object may have resulted in data corruption and undefined behavior. This situation was most likely to occur if the DynamicData object had nested complex members and the **DynamicData::bind_complex_member()** and **DynamicData::unbind_complex_member()** APIs were used. This problem has been resolved.

[RTI Issue ID CORE-6964]

6.4.8 Unable to Set Default Case in Union using DynamicData API

When setting a union member using the DynamicData API, it was not possible to set the member identified by the default label after any other member of the union had previously been set. This problem has been resolved.

[RTI Issue ID CORE-6905]

6.4.9 Unbounded Memory Growth when Setting and Clearing Optional Members in a DynamicData Object

There was unbounded memory growth if optional members within a DynamicData object were repeatedly set and then cleared. This issue has been resolved.

[RTI Issue ID CORE-6971]

6.4.10 DDS_DynamicData_set_complex_member() could Corrupt DynamicData Object

Calling the **DDS_DynamicData_set_complex_member()** API may have corrupted the contents of the DynamicData object whose member was being set. This could cause subsequent calls that get the data values from the DynamicData object, or write the DynamicData object, to fail or provide incorrect results.

This behavior was most likely to occur if the complex member being set had previously been set or if a member following the complex member in the type had been set before the complex member was set.

This problem has been resolved.

[RTI Issue ID CORE-7083]

6.4.11 DynamicType::member_kind() Returned Wrong Kind in Some Cases (Modern C++ API only)

The getter **member_kind()** returned the wrong kind if the member was a long long, unsigned long long, or long double. This problem has been resolved.

[RTI Issue ID CORE-7428]

6.4.12 Memory Leak when Deleting TypeObject Associated with Empty Valuetype Definition on Some Platforms

Using empty valuetypes produced a memory leak reported as follows by Valgrind™. This leak was reported only by certain platforms, including Windows and Linux.

```
==4287== 0 bytes in 2 blocks are definitely lost in loss record 1 of 1
==4287== at 0x4C2CC70: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==4287== by 0xBBC1D9: RTIOsapiHeap_reallocateMemoryInternal (heap.c:405)
==4287== by 0xB7D66B: RTICdrTypeObject_assertTypeFromTypeCode (typeObject.c:2226)
==4287== by 0xB7E038: RTICdrTypeObject_assertTypeFromTypeCode (typeObject.c:2360)
==4287== by 0xB7E038: RTICdrTypeObject_assertTypeFromTypeCode (typeObject.c:2360)
==4287== by 0xB7F8B6: RTICdrTypeObject_createFromTypeCode (typeObject.c:2708)
==4287== by 0xB75F83: RTICdrTypeObjectFactory_createTypeObjectFromTypeCode
(typeObjectFactory.c:340)
==4287== by 0xB76308: RTICdrTypeObjectFactory_createTypeObjectBufferFromTypeCode
(typeObjectFactory.c:384)
```

```
==4287== by 0x8EE40C: PRESParticipant_assertTypeObjectFromTypeCode
(TypeObjectTable.c:340)
==4287== by 0x8EAD20: PRESParticipant_registerType (Type.c:545)
==4287== by 0x551745: DDS_DomainParticipant_register_type
(DomainParticipant.c:8653)
==4287== by 0x44CE72: DDSDomainParticipant_impl::register_type(char const*,
PRESTypePlugin const*,
  void*, int) (DomainParticipant.cxx:2838)
```

The memory leak corresponded to an array of 0 elements allocated during TypeObject construction. This problem has been resolved.

[RTI Issue ID CORE-7467]

6.4.13 Use of Java Custom Content Filter with DynamicData Caused Crash on DataWriter Side

Using a Java custom filter with DynamicData may have caused the application to crash on the DataWriter side if writer-side filtering was enabled.

A workaround was to disable writer-side filtering by setting **writer_qos.writer_resource_limits.max_remote_reader_filters** to 0.

This problem has been resolved.

[RTI Issue ID CORE-7876]

6.4.14 Unbinding from a Complex Member May Clear Optional Members

Under certain situations, calling **DDS_DynamicData_unbind_complex_member()** may have cleared the value of any optional member that had been previously set in the complex member. This problem has been resolved.

[RTI Issue ID CORE-7919]

6.4.15 Wrong Type Returned from DynamicData get_type(), get_member_type()—Java API Only

In previous releases, invoking the DynamicData APIs **get_type()** and **get_member_type()** on a DynamicData object may have returned a type that did not correspond to the type associated with the DynamicData object.

Specifically:

- Member IDs set explicitly by the user (using the ID annotation) were not provided.
- Information about optional members was not provided. A member that was optional was provided as non-optional.

This problem has been resolved.

[RTI Issue ID CORE-7942]

6.4.16 Wrong Values from DDS::DynamicData::print() in Certain Cases—C/C++ APIs Only

In some cases, the **DynamicData::print()** method may have printed incorrect values in C/C++. This occurred when**DynamicData::print()** was invoked in a DynamicData object built using the type code discovered as part of the SubscriptionBuiltinTopicData or PublicationBuiltinTopicData. This problem has been resolved.

[RTI Issue ID CORE-7595]

6.4.17 TypeCode Associated with Discovered DataWriters and DataReaders was Wrong in Some Cases

The TypeCode associated with discovered *DataWriters* and *DataReaders* (accessible using PublicationBuiltinTopicData and SubscriptionBuiltinTopicData) may have been wrong. This only occurred when the remote type contained a sequence, array, or alias whose elements referred directly or indirectly (nested) to a type that was marked as MUTABLE, contained optional members, or used the **//@ID** annotation to break default ID assignment.

For example:

```
struct test {
    float c;
    string<5> a; //@Optional
}; //@top-level false

struct MyType {
    long theKey; //@key
    long count;
    sequence<test, 30> test;
};
```

In the above example, the sequence test refers to a type containing an optional member. The discovered TypeCode mistakenly did not report the member **a** as optional when it should have. This problem has been resolved.

[RTI Issue ID CORE-7682]

6.4.18 Serialized Sample Size Calculation Incorrect for DynamicData Samples with Many Mutable Members

The serialized sample size calculation for DynamicData samples with types that contained large numbers of mutable members may have been too small. This may have caused errors in any operation that

depended on this calculated size (for example, **DDS_DynamicData_to_cdr_buffer()**) and a number of internal operations. This problem has been resolved.

[RTI Issue ID CORE-7744]

6.4.19 Incorrect Results when using DynamicData and Arrays containing Complex Members with Optional Members

There may have been incorrect results or segmentation faults when using a type with an array of complex members that contained optional members. If only a subset of the complex members in the array were set, the **DDS_DynamicData_equal()** and **DDS_DynamicData_print()** functions reported incorrect results. Also, writing a Dynamic Data sample in which only a subset of the complex members in the array were set resulted in a segmentation fault. All these problems have been resolved.

[RTI Issue ID CORE-7783]

6.4.20 Segmentation Fault when Deserializing Malformed TypeObject for Module Types

Connext DDS may have crashed if it received an malformed TypeObject during the discovery phase. This may have happened when a Connext DDS application discovered other DDS applications. The robustness of TypeObject deserialization has been improved to prevent this problem.

[RTI Issue ID CORE-7648]

6.4.21 Unbinding from Empty Sequence may have Caused Data Corruption

In some situations when using the DynamicData API, unbinding from an empty sequence may have caused data corruption. This could have led to incorrect or missing data. This problem has been resolved.

[RTI Issue ID CORE-8044]

6.5 Fixes Related to Modern C++ API

6.5.1 Passing Incorrect Index to Some DynamicData Getters or Setters caused Undefined Behavior–Modern C++ API Only

Some of the DynamicData::value<T>() getters and setters caused undefined behavior (likely crashing) when they received an out-of-bounds index, instead of throwing dds::core::InvalidArgument.

For example:

```
DynamicData s = ...;
int member_index = 1000; // doesn't exist in the type
auto member_value = s.value<dds::core::string>(member_index); // undefined behavior, should
throw InvalidArgument
```

The problem only affected the overloads of value, and get_values for some types T, not all of them.

This problem has been resolved and now is handled by throwing an InvalidArgumentError.

[RTI Issue ID CORE-7627]

6.5.2 Conditions and WaitSets Missing Some Reference-Type Operations—Modern C++ API Only

All reference types in the API (such as DomainParticipant, FlowController, ...) provide a set of common operations (https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/api/connext_dds/api_cpp2/group__DDSCpp2Conventions.html#a_st_ref_type). However, *Condition* and its derived classes, as well as *WaitSet* did not provide some of them, namely, the creation from, assignment from, and comparison to dds::core::null or nullptr. For example, the following code did not compile:

```
dds::core::cond::GuardCondition c = dds::core::null;
```

This prevented applications from declaring a *Condition* or *WaitSet* variable before initializing it.

This problem has been resolved and now *Conditions* and *WaitSets* provide all the expected operations of a reference type.

[RTI Issue ID CORE-7749]

6.5.3 Renamed Verbosity Level—Modern C++ API Only

The 'ERRORY' enumerator in the rti::config::Verbosity enumeration was named incorrectly, it has been renamed to 'EXCEPTION'.

[RTI Issue ID CORE-7718]

6.5.4 Missing Functions to Get Discovered Participants—Modern C++ API Only

The Modern C++ API did not provide the functions to retrieve information about discovered participants. In the traditional C++ API, these functions are:

- **DomainParticipant::get_discovered_participants()**
- **DomainParticipant::get_discovered_participant_data()**

The following two standalone functions have been added to the modern C++ API:

- **rti::domain::discovered_participants()**
- **rti::domain::discovered_participant_data()**

(They are available by including dds/domain/discovery.hpp)

[RTI Issue ID CORE-7789]

6.5.5 Use of Deprecated std::auto_ptr Removed–Modern C++ Only

The C++ request-replyAPI used a **std::auto_ptr** in one header, which may have caused deprecation warnings on some compilers. This warning has been resolved.

[RTI Issue ID REQREPLY-38]

6.5.6 Publisher's wait_for_acknowledgments() did not Work in Modern C++

It was not possible to use **dds::pub::Publisher::wait_for_acknowledgments()** in the Modern C++ API. This problem has been resolved.

[RTI Issue ID CORE-7901]

6.5.7 Function to_cdr_buffer() may have Returned Larger-than-Needed Buffer–Modern C++ API Only

The function **rti::topic::to_cdr_buffer()** may have returned a vector with more bytes than needed. This problem did not affect correctness, since a later call to **from_cdr_buffer()** with that vector as a parameter would have correctly deserialized and created the correct data-sample.

This problem has been resolved. Now the returned vector will have the exact size needed to contain the CDR buffer.

[RTI Issue ID CORE-7986]

6.5.8 Incorrect dds::core::Error Copy Constructor did not Copy Exception Message–Modern C++ API Only

This problem may have caused **std::rethrow_exception** to strip out the original message in the **dds::core::Error** being rethrown. This problem has been resolved.

[RTI Issue ID CORE-7978]

6.5.9 Memory Leak in Some Read/Take Operations–Modern C++ API Only

The *DataReader's* read and take operations that receive a sample as an argument may have leaked memory when the following circumstances were true:

- The same sample was passed more than once.
- The *DataReader*'s topic-type contained sequences of types that required dynamic memory allocation, such as strings and structs containing other sequences.

The same problem also affected the function **rti::topic::from_cdr_buffer()**.

The other read and take operations (those that return a LoanedSamples collection or receive an iterator range) were not affected.

This problem has been resolved.

[RTI Issue ID CORE-8084]

6.6 Fixes Related to Java API

6.6.1 Crash in Java API when Running Out of Space

When using the Java API, running out of memory during the creation of a *DataReader* resulted in a crash. This problem has been resolved.

[RTI Issue ID CORE-3447]

6.6.2 'last_reason' in Status from DataReaderListener's on_sample_lost () Callback not Populated–Java API Only

When using the Java API, the **last_reason** field in the status provided by the DataReaderListener's **on_sample_lost()** callback was not populated and always had the value NOT_LOST. This problem has been resolved.

[RTI Issue ID CORE-6090]

6.6.3 Java Virtual Machine may have Hung on Shutdown

A race condition may have caused applications using the Connext DDS Java API to hang during the Java Virtual Machine shutdown. The application did not finish normally and had to be killed. This problem was sporadic and more common in applications creating many *DomainParticipants*. This problem has been resolved.

[RTI Issue ID CORE-7427]

6.6.4 Possible Crash in Java Application using IDL Type Defined as Valuetype with No Members

Due to a regression from previous releases, the creation of TypeCode from an IDL type defined as (or containing) a valuetype with no members may have crashed. This affected APIs such as **TypeSupport::register_type()**.

This problem has been resolved and it is again possible to create *DataWriters* and *DataReaders* with this kind of type.

[RTI Issue ID CORE-7659]

6.6.5 Unreported Exception after JVM Ran out of Memory while Creating DataReader

In some circumstances, if the *DataReader* creation method failed due to memory exhaustion, it returned an invalid object, instead of null. This could have led to undefined behavior in the application, which was unaware of the error.

This problem has been resolved. Now if that failure occurs, **Subscriber.create_datareader()** returns null.

[RTI Issue ID CORE-7730]

6.6.6 Undefined Reference to "java.nio.charset.StandardCharsets" in Java API

In Connext DDS 5.2.7, the Java API included a reference to **java.nio.charset.StandardCharsets**, which was introduced in Java 1.7. Because of this, users could no longer run their Java applications with Java 1.5 or 1.6.

This problem has been resolved by replacing the use of **java.nio.charset.StandardCharsets** with **com.rti.dds.infrastructure.StandardCharsets**.

[RTI Issue ID CORE-8071]

6.6.7 Java Method to Configure Public-Key Infrastructure (PKI) Elements of TLS Transport

Configuration of the TLS secure transport requires, among other things, the specification of Public-Key Infrastructure (PKI) elements. These PKI elements include the CA certificates, the certificate chain, and the private key. Configuration of a secure RTI Connext DDS transport is done with a set of properties in the PropertyQosPolicy. There is a new method that configures the required PKI properties from the elements represented as objects in memory, instead of being manually specified as path to files. The new method belongs to the PropertyQosPolicyHelper class:

```
public static void configure_pki_secure_transport_properties(
               PropertyQosPolicy policy,
               String transport_plugin_prefix,
               java.security.cert.Certificate[] root_ca_certificates,
               java.security.cert.Certificate[] certificate_chain,
               java.security.PrivateKey private_key) {
```

To configure the PKI elements of a secure transport, call **configure_pki_elements()** and pass in the PropertyQosPolicy member of the DomainParticipantQos that is used to create a *DomainParticipant*.

The PKI objects are encoded in a set of properties. This may require you to modify the DomainParticipantResourceLimitsQosPolicy so it can support the required set of properties. In order to know the minimum required length, you can call the **get_qos_resource_limits_property_string_max_length()** operation, which belongs to PropertyQosPolicyHelper.

[RTI Issue ID COREPLG-314]

6.6.8 Java Method to Compute Property String Maximum Length of a Given PropertyQosPolicy Instance

If an application needs to add a set of properties to a PropertyQosPolicy, you can get the property string length that is required to represent these properties from a new PropertyQosPolicy Helper operation:

```
int get_qos_resource_limits_property_string_max_length(PropertyQosPolicy property);
```

This is useful when the required value to represent the set of properties is bigger than the default value in the ResourceLimits QoS of the entity. In this situation, you can guarantee correct operation by setting that value to the amount returned by this operation.

[RTI Issue ID COREPLG-314]

6.7 Fixes Related to Transports

6.7.1 Multicast not supported on INTEGRITY 11 Platforms

Connext DDS did not support multicast on INTEGRITY 11 platforms. This problem has been resolved and multicast is now supported.

[RTI Issue ID PLATFORMS-930]

6.7.2 Possible Segmentation Fault when using Shared Memory Transport

Using the Shared Memory transport may have led to a rare segmentation fault. This occurred when there were multiple *DataWriters* in a participant 'A' writing samples with a high frequency that must be received by the *DataReaders* in a second participant 'B'. This problem has been resolved.

[RTI Issue ID CORE-7667]

6.7.3 UDPv4 Multicast Communication Interrupted if Physical NIC Disabled

The communication between two nodes was interrupted if the physical NIC was disabled in the following scenario, on Windows hosts only:

1. Use only a multicast address in the initial peers (for example: builtin.udpv4://239.255.0.1).
2. Start the publisher and subscriber on different nodes.
3. Check that communication occurs.
4. Disable the NIC in one of the nodes.
5. Communication will be interrupted after discovery liveliness expires because multicast traffic is not received by the publisher or subscriber anymore.

This problem has been resolved.

[RTI Issue ID CORE-6908]

6.7.4 Participant may have Hung on Shutdown if Multicast Enabled or in Presence of Some Firewalls/Antivirus–OS X and QNX Platforms Only

On an OS X or QNX platform, 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-7188]

6.7.5 Unexpected Non-Addressable Locator Messages when Starting Two Participants in Same Machine

When starting two participants with different RTPS host IDs within the same machine, with the UDP and SHMEM transports enabled, some unexpected non-addressable SHMEM locator error messages were shown. For instance, if you started a publisher with only one network interface up, and started a subscriber after tearing down the interface, you may have see the following messages:

```
COMMENDSrReaderService_assertRemoteWriter:Discovered remote writer using
a non-addressable locator for a transport with class ID 16777216.
This can occur if the transport is not installed and/or enabled in the local
participant.
See https://community.rti.com/kb/what-does-cant-reach-locator-error-message-mean
for additional info.
can't reach:
  locator:
    transport: 16777216
    address: 0000:0103:0401:0000:0000:0000:0000:0000
    port: 39910
    encapsulation:
  transport_priority: 0
  aliasList: ""
```

This issue only occurred when **participant_qos.wire_protocol.rtps_auto_id_kind** not RTPS_AUTO_ID_FROM_UUID. This issue has been resolved.

[RTI Issue ID CORE-7676]

6.7.6 Problems using Windows Machines with more than 20 Interfaces or IP Addresses

If your Windows machine had more than 20 interfaces or IP addresses you may have seen this error:

```
NDDS_Transport_UDPv4_query_interfaces:error accessing Windows registry: operation:
RegQueryValueEx error: 234
```

In addition, Connext DDS may have been unable to use some of your interfaces, particularly if they were DOWN when the application started. This problem has been resolved.

[RTI Issue ID CORE-7743]

6.7.7 Default Value for message_size_max Exceeded Maximum for UDPv6

The maximum payload size of a message sent using UDPv6 is 65,487 bytes. However, the default value of the transport property **message_size_max** was incorrectly set to 65,507 bytes in previous releases.

In Connext DDS 5.2.5, when the UDPv6 transport was enabled with the default value for **message_size_max**, the following message was printed:

```
Reducing message_size_max from 65507 to 65487.
The current value of message_size_max has been changed to 65487.
```

In other releases, an RTPS message with a size greater than 65,487 may have never been sent over the UDPv6 transport. This problem has been resolved.

[RTI Issue ID CORE-7748]

6.7.8 Transport Priority not Used for Unicast Traffic if Multicast also Used

If discovery traffic or user traffic was sent over multicast and was marked with a transport priority using the **metatraffic_transport_priorit**y field in the DiscoveryQosPolicy or the TransportPriorityQosPolicy, any related heartbeats or ACKNACKs sent over unicast were not marked with a transport priority. This problem has been resolved.

[RTI Issue ID CORE-7788]

6.7.9 Possible Shared Memory Communication Failure When Creating DomainParticipants in Multiple Threads

When creating *DomainParticipants* in multiple threads, if the participants were configured to use shared memory and another transport, it was possible for those participants to fail to communicate with each other over shared memory. This problem was seen on a QNX system, but may have happened on any architecture. This problem has been resolved.

[RTI Issue ID CORE-7983]

6.7.10 TCP Transport Plugin Opened UDPv4 Sockets when Running in LAN Mode

The TCP transport plugin created unnecessary UDPv4 sockets when running in LAN mode. This problem has been resolved.

[RTI Issue ID COREPLG-246]

6.7.11 Wrong Connection Peer Address on TCP Server for Windows IOCP Server Connections

There was an issue in the TCP transport plugin that may have caused an incorrect peer address to be used on the TCP server side when using Windows IOCP. In particular, this issue affected the following TCP Transport Plugin use cases:

- Logging "Connection established from client at:" messages.
- Calling to NDDS_Transport_TCPv4_OnConnectionEstablishedCallback on_connection_established.
- Calling to NDDS_Transport_ConnectionEndpoint_PeerEquals_Fcn peer_equals.

This problem, which did not affect communication between TCP Transport Plugins, has been resolved.

[RTI Issue ID COREPLG-352]

6.7.12 Potential Memory Corruption when Enabling or Disabling Interfaces with UDPv4-Based Transports–Windows Platforms Only

This problem only affected Windows platforms and the following transports: UDPv4, TCPv4, DTLSv4, LBRTPS, ZRTPS, and WAN when the transport property **ignore_nonup_interfaces** was set to 0 or when using version 5.2.2.

When enabling or disabling interfaces, plugging or unplugging Ethernet cables, or disconnecting the router, you may have seen the following error and in rare cases your application may have issued a segmentation fault:

```
RTIOsapiHeap_freeBufferAligned:inconsistant free/alloc
```

The problem has been resolved.

[RTI Issue ID COREPLG-401]

6.7.13 Potential Misbehavior or Segmentation Fault in UDP-Based Transports when Setting Allow/Deny Interface List Properties

Setting the allow/deny interface list properties in UDP-based transports may have led to misbehavior, and a potential segmentation fault. This could have happened when you changed the status of the interfaces (for example, enabling a disabled interface) after the *DomainParticipant* associated with the transport was enabled.

The UDP-based transports are: UDPv4, WAN, DTLSv4 and the Limited Bandwidth Transport Plugins LBRTPS, LBST, and ZRTPS.

The allow/deny interface list properties are:

- **allow_interfaces_list**
- **deny_interfaces_list**
- **allow_multicast_interfaces_list**
- **deny_multicast_interfaces_list**

With Connext DDS 5.2.2, you may have run into this problem at any time, unless you disabled interface tracking using the property: disable_interface_tracking.

With other releases, you may have run into this issue only when the property **ignore_nonup_interfaces** was explicitly to false set on Windows platforms.

This problem has been resolved.

[RTI Issue ID COREPLG-408]

6.7.14 TCP Transport not Robust Against Receiving Unexpected Logical Port Responses

There was an issue affecting the TCP Transport Plugin that may have led to unexpected plugin behavior or a crash. In particular, this issue may have triggered upon receiving a TCP Logical Port Response control message that did not match any of the TCP Logical Port Request control messages for which the local TCP Transport Plugin was expecting a response. This problem has been resolved.

[RTI Issue ID COREPLG-415]

6.7.15 TCP Transport Clients Running in WAN Mode not Robust Against Failed Connect while Enabling DomainParticipant

When using WAN modes, a failed connect() while enabling the *DomainParticipant* may have prevented communication from establishing forever (until the TCP Transport plugin was restarted). This problem has been resolved.

[RTI Issue ID COREPLG-293]

6.7.16 Shared Memory Transport Mismatch Errors for Participants Running on Different Machines

A local Participant may have incorrectly printed shared-memory mismatch errors for remote Participants running on a different machine. An example of those errors is:

```
PRESParticipant_checkTransportInfoMatching:Warning: discovered remote participant
'key: c0a8ce01, a04c, 2' 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 'testParticipant'. 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.
```

This problem has been resolved. Starting with this release, the local Participant will no longer print shared memory mismatch errors for remote Participants running on a different machine if remote Participants are announcing UDPv4/UDPv6 locators.

[RTI Issue ID CORE-6763]

6.7.17 Potential Deadlock upon Failed TCP Transport Plugin Client Initial Connect

There was an issue that may have provoked a deadlock in the TCP Transport Plugin Client upon a failed connect() during the plugin initialization. This problem has been resolved.

[RTI Issue ID COREPLG-386]

6.7.18 Participant may have Hung on Shutdown if Multicast Enabled or in Presence of Some Firewalls/Antivirus–AIX, VxWorks, and INTEGRITY Platforms Only

On an AIX, VxWorks, or INTEGRITY platform, 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-7917]

6.8 Fixes Related to Wire Protocol

6.8.1 Receiving Multiple RTPS Messages on Same UDP Datagram not Supported

If multiple RTPS messages were included as part of the same UDP datagram, only the first one was processed; any subsequent messages were incorrectly ignored. This problem has been resolved.

[RTI Issue ID CORE-5974]

6.8.2 Valid RTPS Endpoint Object ID Skipped during Auto Assignment of Object IDs

A participant may have skipped valid and unused endpoint RTPS Object-IDs during endpoint creation. In particular, this issue was triggered when explicitly assigning an endpoint Object ID through the DataWriterProtocol/DataReaderProtocol QoS policy's **rtps_object_id**. In this scenario, two object IDs were incorrectly marked as "in use"—instead of just one. This problem has been resolved.

[RTI Issue ID CORE-7253]

6.8.3 Multicast Retransmissions not Effective

Repair packets sent through multicast were set with a reader GUID. This prevented a single transmission of multicast packets from being accepted by all matched *DataReaders*. This problem has been resolved by leaving the reader GUID unset. This allows a single multicast retransmission to be accepted by all matched *DataReaders*.

[RTI Issue ID CORE-7598]

6.8.4 Wrong RTPS GAP messages Emitted by Reliable DataWriters in Some Cases

In previous releases, a Reliable *DataWriter* may have sent invalid GAP messages to a VOLATILE Reliable *DataReader* after receiving a preemptive ACK or after receiving a NACK for samples that the *DataReader* has already received.

This message did not cause the *DataReader* to misbehave but it could have led to issues when interoperating with other DDS vendors.

[RTI Issue ID CORE-7836]

6.8.5 Unnecessary Periodic NACK Traffic

This issue only affected applications running Connext DDS 5.2.7.

In applications where the **DiscoveryConfigQosPolicy.locator_reachability_lease_duration** was left at the default setting and no TopicQueries were created, NACK traffic was generated every 5 seconds from a built-in ServiceRequest *DataReader* that had no matching *DataWriter*. This problem has been resolved.

[RTI Issue ID CORE-7930]

6.9 Fixes Related to Logging

6.9.1 Potential Segmentation Fault when Enabling Multichannel for Distributed Logger or Monitoring Library DataWriters

When *DataWriters* for distributed logger or monitoring library were initialized using a QoS profile in which multichannel was enabled, this may have caused a segmentation fault. This problem has been resolved.

[RTI Issue ID CORE-7187]

6.9.2 Distributed Logger Option echoToStdout not Copied with RTI_DL_Options_copy()

RTI_DL_Options_copy() did not completely copy Distributed Logger options; it skipped the RTI_DL_Option's member **echoToStdout**. This problem has been resolved.

[RTI Issue ID DISTLOG-160]

6.9.3 Warning Logged if Some Network Interfaces Disabled when Creating DomainParticipant

In Connext DDS 5.2.7, you may have seen the following warning when creating a *DomainParticipant* in a system where some of the networks interfaces were disabled:

```
RTIOsapiInterfaces_getIPv4Interfaces:skipped
```

The warning has been moved to a higher verbosity level.

[RTI Issue ID CORE-7974]

6.9.4 .NET ConfigLogger set_output_device() was Misspelled

Starting with Connext DDS 5.3.0, the name of this method has changed from **set_ouput_device**() to **set_output_device()**.

[RTI Issue ID CORE-7557]

6.9.5 Possible Deadlock in Distributed Logger during Finalization under High Verbosity Settings

It was possible for a thread to deadlock while finalizing the Distributed Logger instance if messages were logged at a high rate. This problem has been resolved.

[RTI Issue ID DISTLOG-174]

6.10 Fixes Related to Vulnerabilities

This release fixes some potential vulnerabilities and memory corruption when receiving malformed data (including RTI Issue IDs CORE-7101, CORE-7556, CORE-7559, CORE-7872, CORE-7877, CORE-7878, CORE-7879, CORE-7880, CORE-7882, CORE-7883, CORE-7884, CORE-7885, CORE-7902, CORE-7912, CORE-7915, CORE-7926, CORE-7927, CORE-7929, CORE-7940, CORE-7948).

6.11 Other Fixes

6.11.1 Connext DDS Thread Execution Interrupted by Signals

Execution of the threads created by Connext DDS, such as the event thread, could have been interrupted by signals in POSIX-compliant operating systems. If you installed a signal handler and executed DDS code within the handler, this may have lead to errors or deadlocks.

In POSIX-compliant operating systems, such as Linux or OS X, DDS threads are created now with all signals blocked except SIGINT (CTRL-C). The signal blocking affects DDS-created threads but not the user threads that create the DDS entities. If you want to receive signals on a DDS thread such as a listener thread, you need to unblock the signal and set a signal handler from the DDS thread.

[RTI Issue ID CORE-179]

6.11.2 Potential Segmentation Fault when Mixing Static and Dynamic Versions of RTI Libraries

Mixing static and dynamic versions of the Connext DDS libraries may have caused an application to crash when creating a *DomainParticipant*. This may have happened, for instance, when trying to load monitoring libraries dynamically from a statically linked RTI Connext DDS application. This problem has been resolved.

[RTI Issue ID CORE-7933]

6.11.3 Tolerance Check Incorrectly Applied in DestinationOrder QoS Policy

To enable systems with eventual consistency, *DataWriters* and *DataReaders* should be configured with the DestinationOrderQosPolicy's **kind** field set to DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS.

Previously, the tolerance check at the *DataWriter*, while using DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, was incorrectly applied across all instances instead of per instance. This problem has been resolved.

[RTI Issue ID CORE-3571]

6.11.4 Shadow Warnings Reported while Compiling Generated Examples with -Wshadow Switch

On UNIX-like systems, shadow warnings were reported while compiling generated examples with the **-Wshadow** switch. The warnings reported by the compiler gcc 4.8.2 on x64 Red Hat Enterprise Linux 7.0 operating system are fixed.

[RTI Issue ID CORE-5098]

6.11.5 Potential Assertion Raised when Using Debug Libraries on Windows Systems

When using debug libraries on Windows platforms, a possible assertion could have been raised when parsing a QoS XML file that contained a non-ASCII character. This problem has been resolved.

[RTI Issue ID CORE-5269]

6.11.6 Incorrect Warning Reported while Trying to Purge Unregistered Instances Proactively

When **autopurge_unregistered_instances_delay** is not DURATION_INFINITE, Connext DDS will attempt to purge unregistered instances proactively, independently of hitting resource limits.

The warning "WriterHistoryMemoryPlugin_dropFullyAckedUnregisteredInstance:unregistered instances not fully acked" was logged during this process. However this warning is valid only when resource limits are hit, not when instances are proactively removed. This problem has been resolved.

[RTI Issue ID CORE-6272]

6.11.7 DataReader's get_matched_publications() may not have Reported all Matching Publications

The *DataReader's* **get_matched_publications()** did not report a publication as a matched publication if it made a transition from incompatible to compatible. This problem has been resolved.

[RTI Issue ID CORE-7149]

6.11.8 Possible Deadlock when Using User-Defined DDSThreadFactory

Using a user-defined DDSThreadFactory to manage internal threads created by Connext DDS may have caused a deadlock. An internal middleware thread being deleted via a user-defined **DDSThreadFactory::delete_thread(void* ddsThread)** may have never finished and hung due because it needed a mutex held by the thread calling **DDSThreadFactory::delete_thread()**. This only occurred if **DDSThreadFactory::delete_thread()** blocked and waited for the thread it was deleting to finish, for instance via a **join()**. This problem has been resolved.

[RTI Issue ID CORE-7334]

6.11.9 Documentation for .NET API did not Include Enumerations

Documentation for enumeration data types did not show up in the .NET API Reference HTML documentation in previous releases. This problem has been resolved.

[RTI Issue ID CORE-7447]

6.11.10 Operations to Look Up Entity by Name Could Cause Deadlock

The following operations were not thread safe and may have caused a deadlock if they were called concurrently:

- **DomainParticipantFactory::lookup_participant_by_name()**
- **DomainParticipant::lookup_subscriber_by_name()**
- **DomainParticipant::lookup_publisher_by_name()**
- **DomainParticipant::lookup_datareader_by_name()**
- **DomainParticipant::lookup_datawriter_by_name()**

This problem has been resolved so these operations are thread-safe.

[RTI Issue ID CORE-7458]

6.11.11 Heap_free() not Available under DDS Namespace in Traditional C++ API

The operation **Heap_free()** was not available under the DDS namespace in the traditional C++ API. This problem has been resolved.

[RTI Issue ID CORE-7488]

6.11.12 DataWriterSeq Class not Available under DDS Namespace in Traditional C++ API

The definition for DataWriterSeq could not be accessed from the DDS namespace in the traditional C++ API. Users had to use DDS_DataWriterSeq instead. This problem has been resolved.

[RTI Issue ID CORE-7510]

6.11.13 Segmentation Fault when Using Coherent Changes with Disabled DataWriters

Your application may have crashed if you tried to write coherent data with enabled and disabled *DataWriters* present simultaneously in your system. This problem has been resolved.

[RTI Issue ID CORE-7543]

6.11.14 Participant Creation Crashed if Specified XML Configuration was not a Participant Configuration

The creation of a Participant from an XML configuration would crash if the specified configuration name referenced an XML configuration (e.g., a QoS profile) that was not a participant configuration (that is, it did not have a <participant> element). This problem has been resolved. Now an error message will be displayed in the standard output in this situation.

[RTI Issue ID CORE-7544]

6.11.15 Errors Linking with mingw

In previous releases, DDS applications may have failed to link when using mingw. This problem has been resolved.

[RTI Issue ID CORE-7577]

6.11.16 Interoperability Problems Between Connext DDS and CoreDX DDS on Windows Systems

Interoperability between CoreDX DDS and Connext DDS was unreliable on some Windows systems due to a port conflict. You may have noticed that applications either completely failed to communicate or only did so if started in some particular order. RTI DDS Spy and Admin Console were sometimes unable to detect CoreDX DDS entities.

This problem has been resolved. However, the interoperability problem affects both Connext DDS and CoreDX DDS symmetrically, so make sure you are also using CoreDX DDS version 3.6.44 or higher.

[RTI Issue ID CORE-7646]

6.11.17 Memory Leak in DataWriter when Matching Reliable DataReader Deleted

A *DataWriter* kept a small amount of memory around for a matching *DataReader* even if the *DataReader* was deleted and unmatched. This may have lead to unbounded memory growth over time as new *DataReaders* were created and destroyed. Notice that the leaked memory was returned to the heap after the *DataWriter* was deleted. This problem has been resolved.

[RTI Issue ID CORE-7652]

6.11.18 Crash when Setting Topic QoS after Setting Publication or Subscription Name

Creating two endpoints of the same kind (e.g., two *DataWriters* or two *DataReaders*) and setting a publication or subscription name in only one of them may have crashed the application. This has problem has been resolved.

[RTI Issue ID CORE-7780]

6.11.19 Rare Potential Segmentation Fault after Deleting Subscriber

There was an issue that in rare cases may have triggered a segmentation fault after the deletion of a *Subscriber*. In particular, this issue may have only triggered when the *Subscriber* contained at least one reliable *DataReader* that had received at least one sample from a matched *DataWriter*, and only if the ACK response delay's minimum and maximum were both non-zero. This problem has been resolved.

[RTI Issue ID CORE-7781]

6.11.20 WaitSet Unblocked Prematurely when QoS Changed such that a Pair of Matching Entities No Longer Matched

If you had a WaitSet attached to a Status Condition, it may have unblocked prematurely in some cases, which can cause high CPU usage. This issue occurred if the use of a WaitSet attached to a Status Condition was preceded by a change in QoS leading to a pair of matched entities to not match anymore (such as a change to the DDS_PartitionQosPolicy). This problem has been resolved.

[RTI Issue ID CORE-7785]

6.11.21 Warning of Unsafe Functions when Compiling Examples for Windows Platforms

When compiling the examples provided for Windows platforms, you may have seen warnings such as:

```
warning C4996: 'sprintf': This function or variable may be unsafe. Consider using sprintf_s
instead.
To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
1>   c:\program files (x86)\microsoft visual studio 12.0\vc\include\stdio.h(356) : see
declaration of 'sprintf
```

This problem has been resolved.

[RTI Issue ID CORE-7848]

6.11.22 DomainParticipant Creation Failure if Some Resource Limits were Too Small

This issue only affected applications running Connext DDS 5.2.7.

If **DomainParticipantResourceLimitsQosPolicy.reader_property_list_max_length** was set to 0 or **DomainParticipantResourceLimitsQosPolicy.reader_property_string_max_length** was set to a value smaller than 74, the *DomainParticipant* creation would fail.

If **DomainParticipantResourceLimitsQosPolicy.writer_property_list_max_length** was set to 0 or **DomainParticipantResourceLimitsQosPolicy.writer_property_string_max_length** was set to a value smaller than 74 and security was enabled or **DiscoveryConfigQosPolicy.locator_reachability_lease_duration** was set to a finite value, the *DomainParticipant* creation would fail. If security was not enabled, *DomainParticipant* creation would have succeeded but any attempt to create a TopicQuery would have failed.

All of these issues have been resolved.

[RTI Issue ID CORE-7849]

6.11.23 Incorrect DiscoveryConfigQoS Field Used to Create ServiceRequest DataReader

This issue only affected applications running Connext DDS 5.2.7.

The wrong field from the DiscoveryConfigQosPolicy was used to populate the RtpsReliableReaderProtocol values in the ServiceRequest *DataReader's* DataReaderProtocolQosPolicy. The **DiscoveryConfigQosPolicy.publication_reader** field was used instead of **DiscoveryConfigQosPolicy.service_request_reader**. This problem has been resolved and the correct field is now being used.

[RTI Issue ID CORE-7989]

6.11.24 Potential Segmentation Fault if System Running out of Resources

There was an issue that prevented Distributed Logger from recovering from an out-of-resources error during initialization, potentially raising a segmentation fault. This problem has been resolved.

[RTI Issue ID DISTLOG-150]

6.11.25 Installer no Longer Fails when '.rtipkg' Filename Changed

The installer previously expected the name of the rtipkg file to be unchanged after download, and would throw an error if it changed. Now the name of the .rtipkg file can be changed, and the installation will succeed.

[RTI Issue ID INSTALL-307]

6.11.26 Polluted Global Namespace in Traditional C++ Request-Reply API

Some API headers included using namespace DDS directives that may have caused name conflicts if the applications defined symbols that also existed in the DDS namespace. This situation would have caused a compilation error. This problem has been resolved by removing all visible 'using' directives from the global namespace.

[RTI Issue ID REQREPLY-29]

6.11.27 Disabling Positive ACKs with Batching could Cause Incorrect Historical Data

In some cases, a *DataWriter* may have incorrectly sent very old samples to a late-joining *DataReader* that was expecting the very latest samples. This occurred if a *DataWriter* set **disable_positive_acks** to true, enabled batching, used keep-last history, and had a history depth not less than the number of samples that can fit in **max_batches**. In this case, the *DataWriter* may have incorrectly sent very old samples, which had already been sent to an early-joining, early-departing *DataReader*, to a late-joining *DataReader* that was expecting the very latest samples. This problem has been resolved.

[RTI Issue ID CORE-6839]

6.11.28 Optional RTI Package for OpenSSL Run-Time Libraries

Connext DDS now provides an optional RTI Package with OpenSSL's run-time libraries. This provides out-of-the-box support for RTI Security Plugins in infrastructure services and tools. The RTI Package must be installed on top of an existing Connext DDS installation using either the **rtipkginstall** command-line utility or RTI Launcher.

[RTI Issue ID INSTALL-283]

6.11.29 Backup Libraries

In previous releases, not all libraries were backed up during the patching process. This fix ensures that all libraries get backed up in the correct place when patching.

[RTI Issue ID INSTALL-285]

6.11.30 Memory Leak when Registering Types

During type registration of both user and builtin types, some memory allocated during type registration was not deallocated. This problem has been resolved.

[RTI Issue ID CORE-7910]

6.11.31 No Communication when Creating and Deleting Readers

In previous releases 5.2.5, 5.2.6, and 5.2.7, there was an issue that may have prevented communication when creating and deleting Readers on a Participant.

This problem, which only affected QNX and Darwin architectures, has been resolved.

[RTI Issue ID CORE-7968]

6.11.32 Potential Segmentation Fault on QNX Architectures

Previous releases may have issued a segmentation fault on QNX architectures due to the internal usage of **gethostbyname()**, which is a non-reentrant function. This problem may have occurred when simultaneously creating *DomainParticipants* in multiple threads. This problem has been resolved by replacing **gethostbyname()** with **gethostbyname_r()**.

[RTI Issue ID CORE-7971]

6.11.33 Memory Leak when Failing to Enable DataReader due to Unavailable receive_port

If enabling a *DataReader* failed because the desired unicast **receive_port** was not available, the *DataReader* leaked memory in the function **COMMENDFragmentedSampleTableResourcePool_new()**. The memory remained leaked even after deleting the *DataReader*. This problem has been resolved.

[RTI Issue ID CORE-8011]

6.11.34 .NET ThreadSettings_t::mask Field had Wrong Type

In the previous release, the type of the ThreadSettings_t::mask was wrong. In Connext DDS 5.3.0, the type has changed from TheadSettingsKind to System::UInt32.

[RTI Issue ID CORE-8081]

6.11.35 Generic.ConnextMicroCompatibilty Built-in Profile not Compatible with Latest Version of RTI Connext Micro

The built-in profile Generic.ConnextMicroCompatibility was incompatible with Micro releases 2.4.4 and higher because the default LivelinessQosPolicy kind was changed to Automatic, making RTI Connext Micro compatible with RTI Connext DDS out-of-the-box.

The Generic.ConnextMicroCompatibilty profile has been updated to reflect that there are currently no QoS changes necessary for RTI Connext DDS and RTI Connext Micro to communicate out-of-the-box. Also, two new profiles, Generic.ConnextMicroCompatibility.2.4.9 and Generic.ConnextMicroCompatibility.2.4.3, have been added. The names of these profiles identify the latest

known version of RTI Connext Micro that they are compatible with at the time of the most recent release of Connext DDS.

[RTI Issue ID CORE-7925]

6.11.36 RTIEventJobDispatcher_updateAgentPriorities Errors when Using EDF or HPF Scheduling Policy

In previous releases, a user creating multiple flow controllers with EDF or HPF scheduling policy may have seen sporadic errors like these:

```
RTIEventJobDispatcher_updateAgentPriorities:!re-sorted job agent already exists
RTIEventJobDispatcher_updateAgentPriorities:could not remove re-sorting agent
```

After the errors, communication may have stopped.

This problem has been resolved.

[RTI Issue ID CORE-8074]

6.11.37 Publication of Entity Description Using Monitoring Library may have Failed in Some Cases

This issue only affected Connext DDS 5.2.3.1, 5.2.3.18, and 5.2.3.21. In those releases, the entity description data may not have been published if the user changed some of the *DomainParticipant* resource limits used by the monitoring library.

For example, if the user updated **participant_qos.resource_limits.writer_user_data_max_length** to be greater than the default value 256, the publication of *DataWriter* description data when using the monitoring library would have failed with an error like this:

```
[D0000|Pub(80000008)|T=Example MyType|CREATE Writer|D0000|Writer(80000003)|GET_QOS]DDS_
OctetSeq_set_maximum:!assert new max cannot be larger than absolute maximum
[D0000|Pub(80000008)|T=Example MyType|CREATE Writer|D0000|Writer(80000003)|GET_QOS]DDS_
UserDataQosPolicy_setup_presentation_policyI:ERROR: Failed to set maximum
[D0000|Pub(80000008)|T=Example MyType|CREATE Writer|D0000|Writer(80000003)|GET_QOS]DDS_
DataWriter_get_qos:!prepare QoS
[D0000|Pub(80000008)|T=Example MyType|CREATE Writer]RTIDefaultMonitorParticipantObject_
sampleAndPublishWriterDesc:!get qos
[D0000|Pub(80000008)|T=Example MyType|CREATE Writer]RTIDefaultMonitorPublisher_
onEventNotify:!publish writer desc
```

This problem has been resolved.

[RTI Issue ID MONITOR-224]

Chapter 7 Known Issues

7.1 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]

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

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

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

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

You can open the file manually from here:

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

This issue does not affect the functionality of the example.

[RTI Issue ID CODEGENII-743]

7.3 DataReader Cannot Persist AppAck Messages Greater Than 32767 Bytes

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

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

For more information, see Section 12.4, Durable Reader State, in the *RTI Connext DDS Core Libraries User's Manual*.

[RTI Issue ID CORE-5360]

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

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

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

[RTI Issue ID CORE-7284]

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

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

If your application requires acknowledgment of message receipt by late-joiners, use the Request/Reply communication pattern with an Acknowledgment type (see Chapter 22, Introduction to the Request-Reply Communication Pattern, in the *RTI Connext DDS Core Libraries User's Manual*).

[RTI Issue ID CORE-5181]

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

Given a Connext DDS 5.3.1 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]

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

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

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

[RTI Issue ID CODEGENII-800]

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

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

to communicate if you are writing small samples.

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

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

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

[RTI Issue ID CORE-6411]

7.9 Memory Leak if Foo:initialize() Called Twice

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

[RTI Issue ID CORE-7678]

7.10 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]

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

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

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

Affected platforms: AIX, Solaris.

[RTI Issue ID CORE-8040]

7.12 Spy and Ping do not Support Security Plugins' Distributed Logging

Spy and Ping do not support enabling the distribution of security-related log messages through the builtin DDS:Security:LogTopic topic.

[RTI Issue ID CORE-8300]

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

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

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

7.14 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]

7.15 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-5.3.1
```

[RTI Issue ID INSTALL-323]

7.16 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]

7.17 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]

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

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

Example:

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

[RTI Issue ID CORE-3018]

7.19 Communication between Kernel and RTP Mode Participants with Shared Memory Transport not Working on 64-bit VxWorks 6 Platforms

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

[RTI Issue ID CORE-8171]

7.20 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 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]

7.21 Known Issues in RTI Monitoring Library

7.21.1 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).

[RTI Issue ID MONITOR-222]

7.21.2 Participant's CPU and Memory Statistics are Per Application

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

[RTI Issue ID CORE-7972]

7.21.3 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]

7.21.4 ResourceLimit channel_seq_max_length must not be Changed

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

[RTI Issue ID MONITOR-220]

Chapter 8 Experimental Features

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

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

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

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

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

Disclaimers:

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

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