RTI Connext Core Libraries Release Notes

Version 7.3.0

Your systems.
Working as one.

Contents

Chapter 1

Copyrights and Notices

Notices

Deprecations and Removals

Any deprecations or removals noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

Deprecated means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release, RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

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

Chapter 2

Introduction

RTI® *Connext®* 7.3.0 is a long-term support release that is built upon and combines all of the features in releases 7.0.0, 7.1.0, and 7.2.0. This document highlights changes since *Connext* 6.1.2.

For an overview of new features in 7.3.0, see RTI Connext Core Libraries What's New .

2.1 Additional Documentation

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

- **Use the RTI Customer Portal** (https://support.rti.com) to download RTI software and contact RTI Support. The RTI Customer Portal requires a username and password. You will receive this in the email confirming your purchase. If you do not have this email, please contact **license@rti.com**. Resetting your login password can be done directly at the RTI Customer Portal.
- **The RTI Community Forum** (https://community.rti.com) provides a wealth of knowledge to help you use *Connext*, including:
  - Documentation, at https://community.rti.com/documentation
  - Best Practices,
  - Example code for specific features, as well as more complete use-case examples,
  - Solutions to common questions,
  - A glossary,
  - Downloads of experimental software,
  - And more.
- Whitepapers and other articles are available from http://www.rti.com/resources.
- Performance benchmark results for *Connext* are published online at http://www.rti.com/products/dds/benchmarks.html. Updated results for new releases are typically published within two months after general availability of that release.

Chapter 3

System Requirements

Connext 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* 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* application for any architecture. You will also need a target installation, which provides the libraries required to build a *Connext* application for that particular target architecture.

Supported platforms, for all products in the *Connext* suite, are listed in *Supported Platforms*.

Future releases may not support all of the platforms supported in this release, or may support different versions of platforms supported in this release.

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

3.1 Supported Platforms

A *platform* refers to the combination of your target machine's OS version, CPU, and toolchain (compiler or Visual Studio). Each platform has an RTI architecture name, which is a shorthand way to identify the platform. The "target" is the machine where you will deploy your completed application. (As opposed to a "host", which is where you will be developing the application.)

For example, if you have a 64-bit Windows machine with Visual Studio® 2017, the architecture name is `x64Win64VS2017`. For a 64-bit Linux machine with gcc version 7.3.0, the architecture name is `x64Linux4gcc7.3.0`.

The first table lists the supported operating systems and their architecture names.

- Section 3.1.1 *RTI Architecture Names*

Once you know your architecture name, use the following tables to see which products/features are supported. You will also need to know your architecture name when downloading/installing various *Connext* libraries.

These table have columns that show you the supported products/features for each architecure. In these tables, Y means Supported.

Note: You may need to scroll down to the end of each table and then scroll to the right in order to see all the content.

- Section 3.1.2 *RTI Infrastructure Services*
- Section 3.1.3 *RTI Tools*
- Section 3.1.4 *RTI Security Extensions and Security Plugins SDK*
- Section 3.1.5 *RTI Connext Add-ons*
- Section 3.1.6 *Other Connext Professional Features*

3.1.1 RTI Architecture Names

Table 3.1: Architecture Names for Connext Professional

OS	Version	CPU	RTI Architecture [2]
Android	Android 12	ARM64	arm64Android12clang12.0.8 ndkr23b [5]
Linux	Ubuntu 18.04 LTS	Arm v7	armv7Linux4gcc7.5.0 [6]
	Ubuntu 18.04 LTS, 22.04 LTS	Arm v8	armv8Linux4gcc7.3.0
	Red Hat Enterprise Linux 8, 9; Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS	x64	x64Linux4gcc7.3.0 [10]
			x64Linux4gcc7.3.0FACE_GP [8]
	Ubuntu 22.04 LTS	x64	x64Linux5Unreal5.2clang15 [9]
macOS	macOS 11, 12, 13	ARM64	arm64Darwin20clang12.0

continues on next page

Table 3.1 – continued from previous page

OS	Version	CPU	RTI Architecture [2]
		x64	x64Darwin20clang12.0
QNX	QNX Neutrino 7.1	Arm v8	armv8QNX7.1qcc_gpp8.3.0
		x64	x64QNX7.1qcc_cxx8.3.0
VxWorks	VxWorks 23.09	x64	x64Vx23.09llvm16.0
			x64Vx23.09llvm16.0_rtp
Windows	Windows 11	ARM64	arm64Win64VS2022
	Windows 10, 11; Windows Server 2016, 2022	x64	x64Win64VS2017
Custom-supported target platforms, only available on demand:			
AIX	AIX 7.2	POWER9	64p9AIX7.2xlclang16.1 [1]
Linux	TI Linux 8.2.0.3	Arm v8	armv8Linux-armgcc9.2.1 [1]
	Red Hat Enterprise Linux 7.x [23]; CentOS 7.0	x64	x64Linux3gcc4.8.2 [1]
		x86	i86Linux3gcc4.8.2 [1]
	RedHawk Linux 8.4.1	x64	x64RedHawk8.4gcc8.5.0 [1]
		x86	i86RedHawk8.4gcc8.5.0 [1]
QNX	QNX for Safety 2.2	Arm v8	armv8QOS2.2qcc_cxx8.3.0 [1]
	QNX Neutrino 7.0.4	Arm v8	armv8QNX7.0.0qcc_cxx5.4.0 [1]
		Arm v7	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]
	QNX Neutrino 7.1	Arm v8	armv8QNX7.1qcc_cxx8.3.0 [1]
	QNX Neutrino 7.0.4	x64	x64QNX7.0.0qcc_gpp5.4.0 [1]
VxWorks	VxWorks 7.0 (SR0630)	x64	x64Vx7SR0630llvm8.0.0.2 [1]
			x64Vx7SR0630llvm8.0.0.2_rtp [1]
	VxWorks 22.03	ppc	ppc32Vx22.03gcc8.3.0_rtp [1]
Windows	Windows 10; Windows Server 2012 R2, 2016	x64	x64Win64VS2015 [1]
	Windows 10; Windows Server 2016	x86	i86Win32VS2015 [1]
			i86Win32VS2017 [1]

3.1.2 RTI Infrastructure Services

This table shows which RTI Infrastructure Services are supported on each architecture.

Table 3.2: RTI Infrastructure Services

OS	RTI Architecture [2]	Persistence [4]	Routing	Recording	Web Integration
Android	arm64Android12clang12.0.8 ndkr23b [5]				
Linux	armv7Linux4gcc7.5.0 [6]	Y	Y	Y	
	armv8Linux4gcc7.3.0	Y	Y	Y	Y
	x64Linux4gcc7.3.0 [10]	Y	Y	Y	Y
	x64Linux4gcc7.3.0FACE_GP [8]				
	x64Linux5Unreal5.2clang15 [9]	Y	Y	Y	Y
macOS	arm64Darwin20clang12.0	Y	Y	Y	Y
	x64Darwin20clang12.0	Y	Y	Y	Y
QNX	armv8QNX7.1qcc_gpp8.3.0		Y	Y	
	x64QNX7.1qcc_cxx8.3.0		Y	Y	
VxWorks	x64Vx23.09llvm16.0				
	x64Vx23.09llvm16.0_rtp				
Windows	arm64Win64VS2022	Y	Y	Y	
	x64Win64VS2017	Y	Y	Y	Y
Custom-supported target platforms, only available on demand:					
AIX	64p9AIX7.2xlclang16.1 [1]				
Linux	armv8Linux-armgcc9.2.1 [1]		Y	Y	

continues on next page

Table 3.2 – continued from previous page

OS	RTI Architecture [2]	Persistence [4]	Routing	Recording	Web Integration
	x64Linux3gcc4.8.2 [1]	Y	Y	Y	Y
	i86Linux3gcc4.8.2 [1]	Y	Y	Y	
	x64RedHawk8.4gcc8.5.0 [1]		Y	Y	
	i86RedHawk8.4gcc8.5.0 [1]		Y	Y	
QNX	armv8QOS2.2qcc_cxx8.3.0 [1]		Y	Y	
	armv8QNX7.0.0qcc_cxx5.4.0 [1]		Y	Y	
	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]		Y		
	armv8QNX7.1qcc_cxx8.3.0 [1]		Y	Y	
	x64QNX7.0.0qcc_gpp5.4.0 [1]		Y	Y	
VxWorks	x64Vx7SR0630llvm8.0.0.2 [1]				
	x64Vx7SR0630llvm8.0.0.2_rtp [1]				
	ppc32Vx22.03gcc8.3.0_rtp [1]				
Windows	x64Win64VS2015 [1]	Y	Y	Y	Y
	i86Win32VS2015 [1]	Y	Y	Y	
	i86Win32VS2017 [1]	Y	Y	Y	

3.1.3 RTI Tools

This table shows which RTI Tools are supported on each architecture.

Table 3.3: RTI Tools

OS	RTI Architecture [2]	Shapes Demo	Launcher	Monitor	Admin Console	System Designer
Android	arm64Android12clang12.0.8 ndkr23b [5]					
Linux	armv7Linux4gcc7.5.0 [6]					
	armv8Linux4gcc7.3.0					
	x64Linux4gcc7.3.0 [10]	Y	Y	Y	Y	Y [12]
	x64Linux4gcc7.3.0FACE_GP [8]					
	x64Linux5Unreal5.2clang15 [9]	Y	Y	Y	Y	Y [12]
macOS	arm64Darwin20clang12.0	Y	Y	Y	Y	Y [13]
	x64Darwin20clang12.0	Y	Y	Y	Y	Y [14]
QNX	armv8QNX7.1qcc_gpp8.3.0					
	x64QNX7.1qcc_cxx8.3.0					
VxWorks	x64Vx23.09llvm16.0					
	x64Vx23.09llvm16.0_rtp					
Windows	arm64Win64VS2022					
	x64Win64VS2017	Y	Y	Y	Y	Y [7]
Custom-supported target platforms, only available on demand:						
AIX	64p9AIX7.2xlclang16.1 [1]					
Linux	armv8Linux-armgcc9.2.1 [1]					
	x64Linux3gcc4.8.2 [1]	Y	Y	Y	Y	Y [12]
	i86Linux3gcc4.8.2 [1]					
	x64RedHawk8.4gcc8.5.0 [1]			Y		
	i86RedHawk8.4gcc8.5.0 [1]					
QNX	armv8QOS2.2qcc_cxx8.3.0 [1]					
	armv8QNX7.0.0qcc_cxx5.4.0 [1]					
	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]					
	armv8QNX7.1qcc_cxx8.3.0 [1]					
	x64QNX7.0.0qcc_gpp5.4.0 [1]					
VxWorks	x64Vx7SR0630llvm8.0.0.2 [1]					

continues on next page

Table 3.3 – continued from previous page

OS	RTI Architecture [2]	Shapes Demo	Launcher	Monitor	Admin Console	System Designer
	x64Vx7SR0630llvm8.0.0.2_rtp [1]					
	ppc32Vx22.03gcc8.3.0_rtp [1]					
Windows	x64Win64VS2015 [1]	Y	Y	Y	Y	Y [7]
	i86Win32VS2015 [1]					
	i86Win32VS2017 [1]					

3.1.4 RTI Security Extensions and Security Plugins SDK

This table shows which architectures support the *RTI Security Extensions*, and the separate add-on *Security Plugins SDK*.

Table 3.4: Security Extensions and Security Plugins SDK

		Security Extensions			Add-on
OS	RTI Architecture [2]	Security Plugins (for OpenSSL) [15]	Security Plugins (for wolfSSL) [16]	TLS Support [15]	Security Plugins SDK [15, 16]
Android	arm64Android12clang12.0.8 ndkr23b [5]	Y		Y	
Linux	armv7Linux4gcc7.5.0 [6]	Y		Y	
	armv8Linux4gcc7.3.0	Y		Y	
	x64Linux4gcc7.3.0 [10]	Y	Y	Y	Y
	x64Linux4gcc7.3.0FACE_GP [8]				
	x64Linux5Unreal5.2clang15 [9]	Y		Y	
macOS	arm64Darwin20clang12.0	Y		Y	Y
	x64Darwin20clang12.0	Y		Y	Y
QNX	armv8QNX7.1qcc_gpp8.3.0	Y	Y	Y	Y
	x64QNX7.1qcc_cxx8.3.0	Y		Y	
VxWorks	x64Vx23.09llvm16.0	Y [18]			
	x64Vx23.09llvm16.0_rtp	Y [18]			
Windows	arm64Win64VS2022	Y		Y	
	x64Win64VS2017	Y		Y	Y
Custom-supported target platforms, only available on demand:					
AIX	64p9AIX7.2xlclang16.1 [1]				
Linux	armv8Linux-armgcc9.2.1 [1]	Y		Y	
	x64Linux3gcc4.8.2 [1]	Y	Y	Y	Y
	i86Linux3gcc4.8.2 [1]	Y		Y	
	x64RedHawk8.4gcc8.5.0 [1]	Y [17]		Y [17]	
	i86RedHawk8.4gcc8.5.0 [1]	Y [17]		Y [17]	
QNX	armv8QOS2.2qcc_cxx8.3.0 [1]	Y	Y	Y	Y
	armv8QNX7.0.0qcc_cxx5.4.0 [1]	Y	Y	Y	
	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]	Y		Y	
	armv8QNX7.1qcc_cxx8.3.0 [1]	Y		Y	
	x64QNX7.0.0qcc_gpp5.4.0 [1]	Y		Y	
VxWorks	x64Vx7SR0630llvm8.0.0.2 [1]	Y [19]			
	x64Vx7SR0630llvm8.0.0.2_rtp [1]	Y [19]			
	ppc32Vx22.03gcc8.3.0_rtp [1]				
Windows	x64Win64VS2015 [1]	Y		Y	Y
	i86Win32VS2015 [1]	Y		Y	
	i86Win32VS2017 [1]	Y		Y	

3.1.5 RTI Connext Add-ons

This table shows various add-on products and which architectures support them.

Table 3.5: Add-ons

OS	RTI Architecture [2]	Cloud Discovery Service	Real-Time WAN Transport	Ada [20]	Ltd. Bandwidth [21]	Queuing Service
Android	arm64Android12clang12.0.8 ndkr23b [5]		Y			
Linux	armv7Linux4gcc7.5.0 [6]		Y			
	armv8Linux4gcc7.3.0		Y		Y	
	x64Linux4gcc7.3.0 [10]	Y	Y		Y	Y
	x64Linux4gcc7.3.0FACE_GP [8]					
	x64Linux5Unreal5.2clang15 [9]	Y	Y		Y	Y
macOS	arm64Darwin20clang12.0	Y	Y			Y
	x64Darwin20clang12.0	Y	Y			Y
QNX	armv8QNX7.1qcc_gpp8.3.0		Y			
	x64QNX7.1qcc_cxx8.3.0		Y			
VxWorks	x64Vx23.09llvm16.0		Y			
	x64Vx23.09llvm16.0_rtp		Y			
Windows	arm64Win64VS2022	Y	Y		Y	
	x64Win64VS2017	Y	Y		Y	Y
Custom-supported target platforms, only available on demand:						
AIX	64p9AIX7.2xlclang16.1 [1]					
Linux	armv8Linux-armgcc9.2.1 [1]		Y			
	x64Linux3gcc4.8.2 [1]	Y	Y	Y	Y	Y
	i86Linux3gcc4.8.2 [1]	Y	Y		Y	
	x64RedHawk8.4gcc8.5.0 [1]		Y			
	i86RedHawk8.4gcc8.5.0 [1]		Y			
QNX	armv8QOS2.2qcc_cxx8.3.0 [1]		Y			
	armv8QNX7.0.0qcc_cxx5.4.0 [1]		Y		Y	
	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]		Y			
	armv8QNX7.1qcc_cxx8.3.0 [1]		Y			
	x64QNX7.0.0qcc_gpp5.4.0 [1]		Y			
VxWorks	x64Vx7SR0630llvm8.0.0.2 [1]		Y			
	x64Vx7SR0630llvm8.0.0.2_rtp [1]		Y			
	ppc32Vx22.03gcc8.3.0_rtp [1]		Y			
Windows	x64Win64VS2015 [1]	Y	Y		Y	Y
	i86Win32VS2015 [1]	Y	Y		Y	
	i86Win32VS2017 [1]	Y	Y		Y	

3.1.6 Other Connext Professional Features

This table shows other features in *Connext Professional* and which architectures support them.

Table 3.6: Other Features

OS	RTI Architecture [2]	Distributed Logger	Monitoring	Monitoring 2.0	LBED [3]	Observability Collector Service
Android	arm64Android12clang12.0.8 ndkr23b [5]	Y	Y	Y		
Linux	armv7Linux4gcc7.5.0 [6]	Y	Y	Y		
	armv8Linux4gcc7.3.0	Y	Y	Y	Y	
	x64Linux4gcc7.3.0 [10]	Y	Y	Y	Y	Y

continues on next page

Table 3.6 – continued from previous page

OS	RTI Architecture [2]	Distributed Logger	Monitoring	Monitoring 2.0	LBED [3]	Observability Collector Service
	x64Linux4gcc7.3.0FACE_GP [8]					
	x64Linux5Unreal5.2clang15 [9]	Y	Y	Y	Y	
macOS	arm64Darwin20clang12.0	Y	Y	Y		
	x64Darwin20clang12.0	Y	Y	Y		
QNX	armv8QNX7.1qcc_gpp8.3.0	Y	Y	Y		
	x64QNX7.1qcc_cxx8.3.0	Y	Y	Y		
VxWorks	x64Vx23.09llvm16.0	Y	Y [22]	Y		
	x64Vx23.09llvm16.0_rtp	Y	Y [22]	Y		
Windows	arm64Win64VS2022	Y	Y	Y	Y	
	x64Win64VS2017	Y	Y	Y	Y	
Custom-supported target platforms, only available on demand:						
AIX	64p9AIX7.2xlclang16.1 [1]	Y	Y [22]	Y		
Linux	armv8Linux-armgcc9.2.1 [1]	Y	Y	Y		
	x64Linux3gcc4.8.2 [1]	Y	Y	Y	Y	Y
	i86Linux3gcc4.8.2 [1]	Y	Y	Y	Y	
	x64RedHawk8.4gcc8.5.0 [1]	Y	Y	Y		
	i86RedHawk8.4gcc8.5.0 [1]	Y	Y	Y		
QNX	armv8QOS2.2qcc_cxx8.3.0 [1]	Y	Y	Y		
	armv8QNX7.0.0qcc_cxx5.4.0 [1]	Y	Y	Y	Y	
	armv7QNX7.0.0qcc_cxx5.4.0 [1, 6, 11]	Y	Y	Y		
	armv8QNX7.1qcc_cxx8.3.0 [1]	Y	Y	Y		
	x64QNX7.0.0qcc_gpp5.4.0 [1]	Y	Y	Y		
VxWorks	x64Vx7SR0630llvm8.0.0.2 [1]	Y	Y [22]	Y		
	x64Vx7SR0630llvm8.0.0.2_rtp [1]	Y	Y [22]	Y		
	ppc32Vx22.03gcc8.3.0_rtp [1]	Y	Y [22]	Y		
Windows	x64Win64VS2015 [1]	Y	Y	Y	Y	
	i86Win32VS2015 [1]	Y	Y	Y	Y	
	i86Win32VS2017 [1]	Y	Y	Y	Y	

3.1.7 Footnotes

These are the footnotes used in the preceding tables.

Table 3.7: Footnotes for Supported Platforms Tables

1	Custom Target Library (CTL), only available on demand. Contact your RTI sales representative or sales@rti.com for more information.
2	Supports DDS 1.4 and RTPS 2.5.
3	LBED = Limited Bandwidth Endpoint Discovery. Supports dynamic linking only.
4	Tested with filesystem only in PERSISTENT mode.
5	Advanced example generation in code generator not supported.
6	These libraries require a hardware FPU in the processor and are compatible with systems that have hard-float libc. See the Platform Notes for compiler flag details.
7	Tested on x64 Windows 10 with Chrome version 112 and Firefox version 108.
8	Request-reply API not supported, DDS Ping and Spy not supported. FACE architectures only available for Connext TSS.
9	Target libraries for Unreal Engine 5.2.
10	This should also work on Wind River Linux 9.
11	Tested with QNX 7.0.0 kernel.
12	Tested on x64 Linux, with Chrome version 112 and Firefox version 108.
13	Tested on ARM64 macOS, with Chrome version 112, and Firefox version 108.
14	Tested on x64 Mac OS 10 with Chrome version 112, Firefox version 108, and Safari version 16.2.
15	Tested with OpenSSL 3.0.12 unless stated otherwise.
16	Tested with wolfSSL 5.5.1.
17	Tested with OS stock version of OpenSSL (OpenSSL 1.1.1k FIPS).

continues on next page

Table 3.7 – continued from previous page

18	Tested with OpenSSL from VxWorks 23 (OpenSSL 3.1.1).
19	Tested with OpenSSL from VxWorks 7 (OpenSSL 1.1.1).
20	Built with AdaCore GNAT Pro 18.2, compatible with version 18.2-23.3.
21	Ltd. Bandwidth = Limited Bandwidth Plugins
22	Memory and CPU usage not available in monitoring data.
23	7.x refers to Red Hat Enterprise Linux 7.0, 7.3, 7.5, and 7.6.

3.2 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 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 Java applications.

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

When Using Visual Studio 2017 — Redistributable Package Requirement

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

You can download the Visual C++ Redistributable for Visual Studio 2017 from this Microsoft website: https://visualstudio.microsoft.com/vs/older-downloads/. Then look in this section: "Redistributables and Build Tools" for Microsoft Visual C++ Redistributable for Visual Studio 2017".

When Using Visual Studio 2019 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2019 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2019 from this Microsoft website: https://visualstudio.microsoft.com/vs/older-downloads/. Then look in this section: "Other Tools and Frameworks" for Microsoft Visual C++ Redistributable for Visual Studio 2019".

When Using Visual Studio 2022 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2022 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2022 from this Microsoft website: https://www.visualstudio.com/downloads/. Then look in this section: "Other Tools, Frameworks, and Redistributables" for Microsoft Visual C++ Redistributable for Visual Studio 2022".

3.3 Disk and Memory Usage

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

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

Chapter 4

Compatibility

Below is basic compatibility information for this release.

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

4.1 Wire Protocol Compatibility

Connext 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 currently supported version is OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5, although some features are not supported. Unsupported features currently are FilteredCountFlag in GAP Submessage, HeartbeatFrag Submessage, and ALIVE_FILTERED instance state. RTI plans to maintain interoperability between middleware versions based on RTPS 2.1. For more details, see Table 4.1 *RTPS Versions*.

Table 4.1 *RTPS Versions* shows RTPS versions supported for each *Connext* release. In general, RTPS 2.1 and higher versions are interoperable, unless noted otherwise. RTPS 2.0 and RTPS 1.2 are incompatible with current (4.2e and later) versions of *Connext*.

Although RTPS 2.1 and higher versions are generally interoperable, there may be specific wire protocol interoperability issues between *Connext* releases. These issues are documented in the "Wire Protocol" section for your release, in the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). Wire protocol issues between 5.3.1 and previous releases are documented in the *Core Libraries Release Notes* for release 5.3.1.

Table 4.1: RTPS Versions

Connext Release	RTPS Standard Version[Page 15, 1]	RTPS Protocol Version[2]
Connext 7.1.0 and above	2.5 (partial support)	2.5
Connext 6 and 7.0.0	2.3 (partial support)	2.3
Connext 5.2 and 5.3	2.2	2.1
Connext 4.5f - 5.1	2.1	2.1
Data Distribution Service 4.2e - 4.5e	2.1	2.1
Data Distribution Service 4.2c	2.0	2.0
Data Distribution Service 4.2b and lower	1.2	1.2

4.2 Code and Configuration Compatibility

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

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

4.3 Extensible Types Compatibility

This release of *Connext* includes partial support for the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3 (DDS-XTypes) 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 *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). See also the *RTI Connext Core Libraries Extensible Types Guide* for a full list of the supported and unsupported extensible types features.

[1] Version number of the RTPS standards document, OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5

[2] RTPS wire protocol version number that *Connext* announces in messages it puts on the wire

Chapter 5

What's New in 7.3.0 LTS

This section describes what's new in the Core Libraries compared to release 7.2.0. For information on new features in releases 7.0.0, 7.1.0, and 7.2.0, which are all also part of 7.3.0 LTS, see Previous Releases, in the *RTI Connext What's New*.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on https://community.rti.com/documentation or in your installation. Or start with the RTI Connext What's New for a launchpad to all products.

Note: For backward compatibility information between 7.3.0 LTS and previous releases, see the Migration Guide on the RTI Community Portal (https://community.rti.com/documentation).

5.1 Sending and Receiving Data

5.1.1 Type evolution now allowed with Zero Copy transfer over shared memory when using FlatData types

Previously, type coercion was not permitted for any topics using Zero Copy regardless of the language binding, which means that you could not evolve the types. If you had set `reader_qos.type_consistency.kind` to AUTO_TYPE_COERCION for a Zero Copy *DataReader*, *Connext* translated this value to DISALLOW_TYPE_COERCION. Setting `reader_qos.type_consistency.kind` to ALLOW_TYPE_COERCION resulted in an error when creating a Zero Copy *DataReader*.

In this release, this restriction no longer applies to topics using Zero Copy when the underlying type is annotated as a FlatData type, allowing type evolution while using Zero Copy.

See Other Considerations, in Zero Copy Transfer Over Shared Memory, in the *RTI Connext Core Libraries User's Manual*, for more information.

5.1.2 Simpler DataReader and DataWriter constructors (Python, Modern C++ APIs)

Many applications don't need an explicit *Subscriber* or *Publisher* if they don't require features like ordered access or partitions. Starting with this release, in the Python and Modern C++ APIs, the `subscriber` argument for the `DataReader` constructors and the `publisher` argument to the `DataWriter` constructors are optional. When not specified, the implicit subscriber and the implicit publisher are used, respectively.

For example, this new way to create a *DataReader* is available:

In Python:

```
reader = dds.DataReader(topic, Foo)
```

In modern C++:

```
DataReader<Foo> reader(topic);
```

These are equivalent to the following:

In Python:

```
reader = dds.DataReader(topic.participant.implicit_publisher, topic, Foo)
```

In modern C++:

```
DataReader<Foo> reader(rti::sub::implicit_subscriber(topic.participant()),␣
↪topic);
```

Note that other language APIs that use a factory pattern are unchanged, since the *DomainParticipant* provides `create_datareader()` and `create_datawriter()` methods that don't require a *Subscriber* or *Publisher*.

5.1.3 Topic constructors now receive std::string_view arguments

When an application using the Modern C++ API is compiled with support for C++17 or higher, the Topic constructors now receive a `std::string_view` argument instead of a `std::string`. Many applications define their topic names as string constants in the application or in IDL. The new IDL4-C++ mapping translates IDL string constants to `std::string_view` in C++. By declaring a `std::string_view` argument, the Topic constructors now accept any type of string (`std::string`, `std::string_view`, `char*`, etc.).

5.1.4 Directly get duration in nanoseconds, instead of calculating it, using new to_nanosec function in Duration class

A new function, `to_nanosec`, is now available in the `Duration` class to get the number of nanoseconds equivalent to the time expressed in a `Duration` object. This new functionality enables you to get a duration value directly in nanoseconds. You no longer need to calculate nanoseconds from the output of other functions (like the output of `to_microsec`).

5.1.5 JSON now a fully supported data format, enabling easier integration of Connext with other technologies

Connext already provided `to_string` APIs that allowed converting a data sample into a JSON string (and other formats). This release adds `from_string` APIs that allow creating a data sample from a JSON string representation.

For example, given an IDL type `Point`, defined as follows:

```
struct Point {
    int32 x;
    int32 y;
};
```

Connext applications can now create and publish a `Point` instance from a JSON string such as `{"x": 1, "y": 2}`.

The following APIs have been added.

- In the Modern C++ API: `rti::topic::from_string`

```
Point p = rti::topic::from_string<Point>("{\"x\": 10, \"y\": 20}");
```

- In the Python API: `TypeSupport.from_string` and, for convenience, `TypeSupport.from_json` as well as `TypeSupport.to_json`:

```
import rti.types as idl

point_support = idl.get_type_support(Point)
point = point_support.from_json(r'{"x": 10, "y": 20}')
```

- In the C# API, `<Type>Support.FromString`:

```
Point p = PointSupport.Instance.FromString("{\"x\": 10, \"y\": 20}")
```

- In the Java API, `<Type>TypeSupport.data_from_string`:

```
Point p = PointTypeSupport.get_instance().data_from_string("{\"x\": 10, \"y\
→": 20}")
```

It's also possible to convert from `DynamicData` samples to JSON strings; the class `DynamicData` includes a new `from_string` method.

After obtaining the `Point` data sample, you can publish it using a *DataWriter* for `Point` (or `DynamicData`) as usual. A *DataReader* can receive the `Point` sample and convert it to JSON using the `to_string` APIs.

(Note that in the C and Traditional C++ APIs, only `DynamicData` provides this functionality.)

5.2 Performance

5.2.1 Skip deserialization in DynamicData for efficient data handling, using new property

By default, a DynamicData object stores its contents in an implementation-specific representation that allows you to get and set fields by name or ID. For this to work, *Connext* must deserialize the sample; however, there are cases where you do not need to mutate or access the data. One such case is when you are storing the data for later. Another is when you are bridging samples to another domain and do not need to inspect the data. For enhanced performance in these types of cases, you can now skip deserialization.

The property `skip_deserialization` can be set to true to skip the automatic deserialization of DynamicData objects. To determine if a sample is in this internal format, use the `DDS_DynamicData_is_cdr` functions. Additionally, some function names have been changed to be consistent with this new functionality. `DDS_DynamicData_set_and_lock_buffer` has been changed to `DDS_DynamicData_set_cdr_buffer`. `DDS_DynamicData_get_storage_buffer` has been renamed `DDS_DynamicData_get_cdr_buffer`. Both of these functions can be used to work with the data that has not been deserialized yet. For more information about them, see the language-specific API Reference documentation. Also find an example here: https://github.com/rticommunity/rticonnextdds-examples/tree/master/examples/connext_dds/dynamic_data_skip_serialization.

5.2.2 Compression level applied to built-in Instance State Consistency DataWriter reduced, speeding up response time and reducing CPU usage

Instance state consistency is achieved through the use of a built-in *DataWriter* that sends information regarding instances to a built-in *DataReader*. The samples sent by this *DataWriter* can be very large, and so compression is applied to this channel. The compression level was previously DDS_COMPRESSION_LEVEL_BEST_COMPRESSION (slowest, but best, compression). This compression level has been updated to DDS_COMPRESSION_LEVEL_BEST_SPEED. See Data Compression in the *RTI Connext Core Libraries User's Manual* for more information on these compression levels. (Note that the builtin *DataWriter/DataReader* for instance state consistency does not use the same **compression_settings** defaults as described in Data Compression. It uses ZLIB, BEST_SPEED, and 1024 as the defaults.)

5.2.3 Performance improvement for durable writer history and Persistence Service with high late-joiner activity

You may have experienced performance degradation while using durable writer history and *Persistence Service*, particularly in scenarios with high late-joiner activity. This problem was due to the durable *DataWriter* not caching repair samples in the durable writer history, leading to excessive database queries from the different late-joiners. *Connext* now caches repair samples if the cache has sufficient capacity (`writer_qos.resource_limits.max_samples` is less than or equal to `writer_qos.durability.storage_settings.writer_instance_cache_allocation.max_count`).

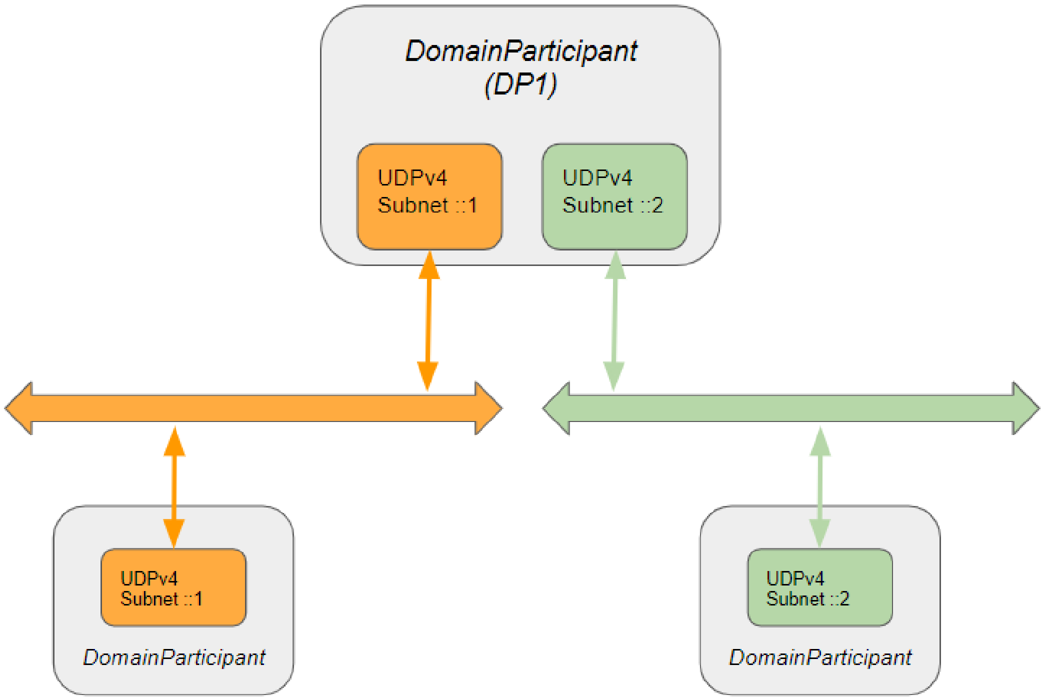
5.2.4 Performance improvement in large systems using SHMEM transport

This release introduces changes to the SHMEM transport that may lead to application performance improvements in systems that create many *DomainParticipants* on the same host that communicate with each other over SHMEM. Before, all *DataWriter* write operations in a *DomainParticipant* were serialized at the transport level, regardless of whether the application employed distinct *Publishers* for different *DataWriters*. With this improvement, a *DomainParticipant* can now perform concurrent writes to other *DomainParticipants* in the system.

5.2.5 More efficient use of network and CPU resources, through support for multiple instances of a UDPv4/UDPv6 transport in a single DomainParticipant

By default, it is not possible to have multiple instances of the UDPv4 or UDPv6 transport in the same *DomainParticipant*, because all instances will try to bind to the same UDP port(s) for receiving data. This release adds the ability to instantiate multiple instances of the UDPv4 and UDPv6 transport within a single *DomainParticipant*, if desired, using two new properties, `dds.transport.UDPv4.builtin.port_offset` and `dds.transport.UDPv6.builtin.port_offset`. This enhancement enables more efficient use of network and CPU resources, which is particularly beneficial when *Connext* applications are operating across various subnets.

With the new properties, you can specify an offset that will be added to the RTPS port(s) to determine the UDP port(s) to bind to. In this way, you can have multiple instances of the UDPv4/UDPv6 transport in the same *DomainParticipant,* if each instance uses different offsets. For more information, see Instantiating Multiple Instances of UDPv4/UDPv6 Transports in the *RTI Connext Core Libraries User's Manual*.



5.3 Debugging and Logging

5.3.1 Set exactly the metrics you want to collect for Observability Framework, using new setting in MONITORING QoS policy

Previously, every observable resource (*DomainParticipant*, *Publisher*, *DataReader*, etc.) was, by default, subscribed to all the available metrics for that resource when it was registered. However, you may not be interested in collecting all of those metrics.

Starting in release 7.3.0, you can now configure the initial set of metrics a resource will be subscribed to upon registration, using the MonitoringQosPolicy. More precisely, you can use the <metrics> tag under <participant_factory_qos>/<monitoring> in XML or the field **telemetry_data.metrics** in the MONITORING QosPolicy documentation.

Note: Starting in this release, by default no metrics will be enabled for any resource unless you specify them in the **metrics** QoS setting. However, if you choose to enable monitoring by using the built-in snippet `Feature.Monitoring2.Enable`, all available metrics will be enabled for all resources.

5.3.2 Better understand port collision warnings through improved logging

If enabled, some benign warning level log messages are printed during Automatic Participant ID Selection (see Choosing Participant IDs in the *RTI Connext Core libraries User's Manual* for more information) because of an expected port collision. Before 7.3.0, these log messages were shown as follows:

```
WARNING [0x0101B4A0,0x30B6995B,0x3CA153D8:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|LC:DISC]NDDS_Transport_UDPv4_Socket_bind_with_ip:0X1CF2 in use
WARNING [0x0101B4A0,0x30B6995B,0x3CA153D8:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|LC:DISC]NDDS_Transport_UDPv4_SocketFactory_create_receive_
↪socket:invalid port 7410
WARNING [0x0101B4A0,0x30B6995B,0x3CA153D8:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|LC:DISC]NDDS_Transport_UDP_create_recvresource_rrEA:!create socket
```

From 7.3.0, these log messages have been improved to clarify that they are benign, adding the `AUTO ID COMPUTE` activity context and a last log message clarifying that we are trying the next port:

```
WARNING [0x01018B7A,0x24B6E2C3,0xC6FB093A:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|AUTO ID COMPUTE|LC:Discovery]NDDS_Transport_UDPv4_Socket_bind_
↪with_ip:BIND FAILURE | Port 7410 in use
WARNING [0x01018B7A,0x24B6E2C3,0xC6FB093A:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|AUTO ID COMPUTE|LC:Discovery]NDDS_Transport_UDPv4_SocketFactory_
↪create_receive_socket:BIND FAILURE | Invalid port 7410
WARNING [0x01018B7A,0x24B6E2C3,0xC6FB093A:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|AUTO ID COMPUTE|LC:Discovery]NDDS_Transport_UDP_create_
↪recvresource_rrEA:CREATION FAILURE | Socket
WARNING [0x01018B7A,0x24B6E2C3,0xC6FB093A:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|AUTO ID COMPUTE|LC:Discovery]DDS_DomainParticipantPresentation_
↪reserve_participant_index_entryports:Trying next port...
```

5.3.3 New log message warning if sample's serialized size exceeds MTU and risks not being sent

A new warning level log message has been added that lets you know when you have created a type whose serialized size is greater than the smallest `message_size_max` across all transports. This message is not logged if asynchronous publishing is used or if the type contains unbounded members. If you receive this message and you do not update the `message_size_max`, the samples will not be able to be sent. See Large Data Fragmentation in the *RTI Connext Core Libraries User's Manual* for more information on `message_size_max`.

5.3.4 Overly long activity context section of log message now truncated instead of generating additional log messages

Previously, if the activity context section of a log message was too long to fit in the log message, the following log message would be printed:

```
There was a problem while logging the following LOCAL message through the␣
↪Connext built-in logging system. It will be logged only to STDOUT:␣
↪RTIOsapiActivityContext_getString:!context maxEntryCount has been reached.
```

This log message has been removed. The activity context section of the log message will now be truncated, and the last 14 characters will be replaced with `(msgTruncated)`. An example of a truncated log message is below:

```
LOCAL [0xDFCD91E1,0x68683864,0xB9B0569F:0x000001C1{Domain=0}|CREATE␣
↪DP|ENABLE|:0x00000188{Entity=Pu,Domain=0}|CREATE DW WITH TOPIC␣
↪DCPSParticipantVolatileMessageSecure|:0xFF0202C3{Entity=DW,
↪Topic=DCPSParticipantVolatileMessageSecure,Type=ParticipantGenericMessage,
↪Domain=0}|ENABLE|:0x000001C1{Domain=0}|ASSERT REMOTE DW|:0xFF0202C4
↪{Entity=DR,Topic=DCPSParticipantVolatileMessageSecure,
↪Type=ParticipantGenericMessage,Domain=0}|LINK 0xDFCD91E1,0x68683864,
↪0xB9B0569F:0xFF0202C3{Type=ParticipantGenericMessag(msgTruncated)|LC:DISC,
↪SEC]RTI_Security_Cryptography_registerEndpoint:{"DDS:Security:LogTopicV2":{
↪"f":"10","s":"3","t":{"s":"1698688649","n":"612150999"},"h":"RTI-10827.local
↪","i":"0.0.0.0","a":"RTI Secure DDS Application","p":"38892","k":"security",
↪"x":[{"DDS":[{"domain_id":"0"},{"guid":"dfcd91e1.68683864.b9b0569f.1c1"},{
↪"plugin_class":"Cryptography"},{"plugin_method":"RTI_Security_Cryptography_
↪registerEndpoint"}]}],"m":"successfully registered endpoint"}}
```

5.3.5 Set finer-grained time values for logging properties that take a duration

The following properties now accept values smaller than one second:

- **dds.participant.logging.time_based_logging.event.timeout**
- **dds.participant.logging.time_based_logging.send.timeout**
- **dds.participant.logging.time_based_logging.process_received_message.timeout**
- **dds.participant.logging.time_based_logging.authentication.timeout**

The properties still accept a single number, which represents the threshold in seconds, but they also accept a value in the form of #s#ms#us#ns so that seconds, milliseconds, microseconds, and nanoseconds can be specified as part of the threshold. The format allows any of the values to be provided, or not, in any order. Some examples of accepted values are:

- 2
- 1s500ms
- 250000000ns

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

5.3.6 Log messages for use by Logger Device contain further useful information: timestamp, source (facility), and message ID

This feature was introduced in 7.1.0, but not documented at that time.

The `LogMessage` data type now contains three new fields:

- `facility`: The Syslog facility associated with the log message. In the Syslog Protocol, the facility is a numerical code that represents the machine process that created a Syslog event. *Connext* uses the facility to represent the source of a given log message. Valid values are: `23 (middleware)`, `10 (security_event)`, `22 (service)`, and `1 (user)`.
- `message_id`: A numeric code that identifies a specific log message. Two log messages that have the same `message_id` will have a similar structure. For example, the `message_id` of both *"ERROR: Failed to get DataWriterQos"* and *"ERROR: Failed to get TopicName"* is associated with the get failure.
- `timestamp`: A `Duration` structure referring to the time when the message was logged.

These fields are available in your application through the use of a Logger Device. See Customizing the Handling of Generated Log Messages. The new fields are documented in the API Reference HTML documentation (for example, here in the Modern C++ API Reference).

5.3.7 New logging APIs provide ability to emit custom log messages in Connext applications

The new logging APIs provide the ability to emit custom log messages in *Connext* applications. This feature will provide more control over the logging process, capture specific application events, and provide deeper insights into the behavior of the application. These new APIs are flexible and allow developers to include relevant information about application-specific events, errors, warnings, or any other important data that requires logging. The new APIs support the use of NDDS_Config_SyslogLevel severity levels and provide an API function in the NDDS_Config_Logger for each level. A new log category NDDS_CONFIG_LOG_CATEGORY_USER has been added to NDDS_Config_LogCategory to accommodate this new category of log messages.

5.3.8 New log warnings when SPDP and SPDP2 participants cannot communicate

DomainParticipants using different discovery protocols (SPDP vs SPDP2) cannot communicate. Now, in this release, when *DomainParticipants* using different discovery protocols detect each other, a log message is generated at the warning level. By default, this logging of mismatch warnings is enabled. You can disable or enable this log message using the boolean property **dds.participant.discovery_config.log_discovery_mismatch**. The messages generated by this condition are:

```
SPDP2 participant received message from SPDP participant. These two␣
↪participants will not communicate.

SPDP participant received message from SPDP2 participant. These two␣
↪participants will not communicate.
```

5.4 DDS Ping and DDS Spy

5.4.1 New option in DDS Ping and DDS Spy to configure Builtin Discovery Plugins

There is a new option, `-discoveryPlugins`, in *DDS Ping* and *DDS Spy* to support configuring the Builtin Discovery Plugins. This new option accepts two values:

- `SDP`: Simple Participant Discovery Protocol and Simple Endpoint Discovery Protocol.
- `SDP2`: Simple Participant Discovery Protocol 2.0 (SPDP2) and Simple Endpoint Discovery Protocol.

The `-discoveryPlugins` option enables you to configure the discovery mechanism (to SDP or SDP2) more easily without the need to load an XML file with that configuration (and then specify that configuration using the `-qosProfile` option).

See the RTI DDS Spy User's Manual and the RTI DDS Ping User's Manual.

5.4.2 New option in DDS Ping and DDS Spy to configure participant partitions

There is a new option, `-participantPartition`, in *DDS Ping* and *DDS Spy* to support configuring *DomainParticipant* partitions. Also, the existing, `partition` option in *DDS Spy* has been renamed to `-subscriberPartition`.

5.5 Miscellaneous

5.5.1 Python API supports latest version, Python 3.12

This release adds support for the Connext Python API on Python 3.12. The API now runs on versions 3.6 to 3.12.

5.5.2 Determine a dynamic type's minimum serialized sample size using new cdr_serialized_sample_min_size

The Python API now exposes a function, `cdr_serialized_sample_min_size`, for finding the minimum size in bytes that a Dynamic Type can be. This function can be used to help determine the minimum resource limits required. For example, along with the maximum serialized size, you can now determine how much disk space or RAM your system will need in order to perform a task. The maximum size was already available.

5.5.3 Validate a license using a shared library call

This release adds the option for a *Connext* application to validate an activation string (license) provided through a shared library call that you can implement. This feature removes the need for a license file on the system where *Connext* runs, and supports use cases where applications retrieve activation strings from a license server. See License Management in the *RTI Connext Installation Guide* for details.

5.5.4 Micro Compatibility Builtin Profile updated to not send serialized types

The `Generic.ConnextMicroCompatibility` builtin QoS profile has been updated to set the `type_code_max_serialized_length` and `type_object_max_serialized_length` resource limits to 0, which disables sending serialized types as part of endpoint discovery. *Connext Micro* does not use serialized type information for endpoint discovery, so sending the type from a *Connext Professional* application to a *Connext Micro* application is unnecessary.

5.5.5 EXCLUSIVE_AREA QoS Policy no longer supported; documentation removed

To advance the deprecation of the EXCLUSIVE_AREA QoS Policy, RTI has removed the EXCLUSIVE_AREA QoS policy documentation. RTI no longer supports the use of this QoS policy. This change specifically targets the EXCLUSIVE_AREA QoS policy, eliminating the ability to set `use_shared_exclusive_area`. In future releases, RTI plans to remove *Connext* components that support or implement the EXCLUSIVE_AREA QoS Policy.

5.6 Third-Party Software Changes

The following third-party software is now used by *Connext*:

Table 5.1: New Third-Party Software

Third-Party Software	Version
y2038	20100403

The following third-party software used by *Connext* has been upgraded:

Table 5.2: Third-Party Software Upgrades

Third-Party Software	Old Version	New Version
InstallBuilder	21.6.0	23.10.1

For information on third-party software used by *Connext* products, see the "3rdPartySoftware" documents in your installation: <NDDSHOME>/doc/manuals/connext_dds_professional/release_notes_3rdparty.

Chapter 6

What's Fixed in 7.3.0 LTS

This section describes bugs fixed in *Connext* 7.3.0 LTS. These are fixes since 7.2.0. For information on what was fixed in releases 7.0.0, 7.1.0, and 7.2.0, which are all also part of 7.3.0 LTS, see *Previous Releases*.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

6.1 Discovery

6.1.1 [Critical] SPDP2 participants with RTPS peers and participant_liveliness_assert_period less than participant_announcement_period may have crashed upon deletion *

Applications with *DomainParticipants* using SPDP2, with RTPS `initial_peers` (that is, peers prefixed with `rtps@`) and a `participant_liveliness_assert_period` less than the `participant_announcement_period`, may have crashed upon deleting the participant.

[RTI Issue ID CORE-14089]

6.1.2 [Major] Rediscovery failed if participant with SPDP2 lost liveliness before receiving remote participant's configuration message *

For participants with SPDP2, participant discovery is only considered "complete" after a local participant has received both a bootstrap message (`DATA(Pb)`) and a configuration message (`DATA(Pc)`) from a remote participant. If the local participant does not receive a configuration message from the remote participant within the remote participant's `participant_liveliness_lease_duration` after receiving the bootstrap message, then the local participant will consider the remote participant to be unalive and can remove it. In previous releases, if the remote participant regained liveliness and sent additional bootstrap messages to the local participant, the local participant failed to rediscover the remote participant. This bug only affected a liveliness loss that occurred before participant discovery completed; participants that lost liveliness after participant discovery completed could be correctly rediscovered. Now, a local participant will successfully rediscover a remote participant that loses liveliness before participant discovery completes.

[RTI Issue ID CORE-13386]

6.1.3 [Major] Participants with SPDP2 failed to discover new participant that was using the same unicast locator as a previously discovered (and not removed) participant *

A *DomainParticipant* with SPDP2 failed to discover a new participant that had the same unicast locators as a previously discovered participant if that participant had not been removed. For example, Participant A and Participant B discover each other. Participant B then shuts down ungracefully (for example, the process receives a SIGINT signal without a signal handler attached to shut down the participant) and does not send a dispose message to Participant A. Participant A still thinks Participant B is alive until Participant B's `participant_liveliness_lease_duration` passes. A new process is then launched which starts Participant C on the same host as Participant B, giving it the same locators as Participant B. Participant C sends a bootstrap message to Participant A; however, Participant A does not respond with additional bootstrap messages because it stopped sending bootstrap messages to those locators. Participant C then cannot discover Participant A because it will never receive a bootstrap message from it. Once Participant A removed Participant B, Participant A could send bootstrap messages to Participant C.

This problem has been fixed. Participant A will now be able to send bootstrap messages to Participant C's locators even when Participant B is still considered "alive", so discovery can complete.

[RTI Issue ID CORE-14048]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.2 Serialization and Deserialization

6.2.1 [Critical] Endpoint creation failed for types with large maximum serialized size

Endpoint (*DataWriter* or *DataReader*) creation may have failed for all language bindings except Java with errors like these:

```
RTIXCdrInterpreter_generateTypePluginProgram:failure generating skip program␣
↪for type recording::final_zero_copy::PackageMessages: too many primitive␣
↪values
RTIXCdrInterpreterPrograms_generate:failure generating skip program for type␣
↪recording::final_zero_copy::PackageMessages
RTIXCdrInterpreterPrograms_generateTopLevelPrograms:failure generating␣
↪programs for type recording::final_zero_copy::PackageMessages
RTIXCdrInterpreterPrograms_initializeWithParams:failure generating programs␣
↪for type recording::final_zero_copy::PackageMessages
DDS_TypeCodeFactory_assert_programs_w_parameters:ERROR: Failed to initialize␣
↪resultPrograms
DDS_TypeCodeFactory_assert_programs_in_global_list:!assert_programs
PRESPsService_enableLocalEndpointWithCursor:failed to attach endpoint to␣
↪typePlugin
PRESPsService_enableLocalEndpoint:!enable local endpoint
```

The issue may have occurred when the following three conditions were true:

- The code for the type was generated with optimization level 2 (default). The optimization level was configured with the command-line option `-optimization`.
- Inline expansion of nested types was applied to the type.
- The maximum serialized size of the type was greater than 256 MB.

[RTI Issue ID CORE-14174]

6.3 Usability

6.3.1 [Major] Incorrect, too-restrictive maximum string size enforced on certain XML fields

There was a bug in the way the accumulated length for `allow_interfaces_list`, `deny_interfaces_list`, `allow_multicast_interfaces_list`, and `deny_multicast_interfaces_list` was computed. As a result, in rare cases where a huge number of interfaces was configured through XML, the maximum limit (32768 bytes) may have incorrectly been applied to the combination of the contents of the `allow_interfaces_list`, `deny_interfaces_list`, `allow_multicast_interfaces_list`, and `deny_multicast_interfaces_list` fields, as opposed to being enforced on a per-field basis. The maximum limit is now applied per field as intended.

[RTI Issue ID CORE-14225]

6.3.2 [Major] rtipkginstaller error in Windows when user name had space *

Windows users who have a space in their user name will now be able to use **rtipkginstaller** without having to worry about their installation path.

[RTI Issue ID INSTALL-965]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.4 Transports

6.4.1 [Critical] High CPU and several warnings in some cases when using Multi-Channel or TransportUnicast QoS

On Linux systems using a UDP transport, it was possible to run into a rare situation in which a socket was marked for removal but then reused before the removal was complete.

In this situation, the partially shutdown socket would return immediately from any **recvfrom()** call. If a message was waiting in the receive buffer, it would be read correctly, but if there was no message, the receive thread printed a warning similar to `WARNING NDDS_Transport_UDP_receive_rEA:got` `unknown message on port 56005 from host 10.0.0.1 port 49825` and then immediately called **recvfrom()** again, causing this message to be printed continuously until the receive socket was eventually deleted.

This situation could have occurred when using the MultiChannelQosPolicy and changing the locator filter, or when using the TransportUnicastQosPolicy and deleting, then immediately recreating, an endpoint with the same QoS values.

Any sockets being shut down are now completely destroyed and cannot be reused while in an inconsistent state.

[RTI Issue ID CORE-13883]

6.4.2 [Critical] Participant may have received RTPS traffic over SHMEM transport not intended for participant

A *DomainParticipant* using the shared memory (SHMEM) transport may have received RTPS traffic over SHMEM intended for a remote *DomainParticipant* running in a different host. This may have led to performance issues in large systems.

The affected traffic included *DomainParticipant* announcement traffic and NACK traffic sent from reliable *DataReaders* to reliable *DataWriters*. In addition, if you disabled or limited the sending of transport information with the participant announcements (for example, you set `participant_qos.resource_limits.transport_info_list_max_length` to 0), the affected traffic was all traffic.

[RTI Issue ID CORE-13828]

6.4.3 [Critical] Undefined behavior of shared memory transport if shared mutex or semaphores removed externally

The shared memory transport uses shared mutexes and semaphores to control access to the shared memory segment. In some operating systems, like Linux, those semaphores could be removed externally, **in some cases, without your knowing.** This behavior occurred on Linux systems when someone running the application disconnected all sessions while a *Connext* application was still running in the background (as a service), and `RemoveIPC=yes` was configured in `logind.conf` (which is the default value in recent Linux versions).

When the shared resources were removed, the behavior of the shared memory transport was undefined. Undefined behaviors included errors handled by *Connext*, CPU usage of 100%, or application crashes.

[RTI Issue ID CORE-13852]

6.4.4 [Minor] DLL leak when using UDP/TCP transports

A Windows DLL was leaked when using UDP or TCP transports.

[RTI Issue ID CORE-14045]

6.5 Reliability Protocol and Wire Representation

6.5.1 [Critical] Writer-side filtered samples not marked as acknowledged when application acknowledgement was used

When application acknowledgements were used in conjunction with ContentFilteredTopics, samples that were writer-side filtered were not immediately marked as acknowledged. This meant, for example, that calling `DDS_DataWriter_wait_for_acknowledgements()` would always time out (until a sample that was not filtered out was sent). This no longer happens: samples that are writer-side filtered that are being sent to *DataReaders* using application acknowledgments are now automatically marked as acknowledged. If installed, the `on_application_acknowledgement()` callback on the *DataWriter* is called for these samples with an empty `DDS_AcknowledgmentInfo::response_data`.

[RTI Issue ID CORE-6132]

6.6 Debuggability

6.6.1 [Major] Thread names longer than 15 characters on QNX platforms caused errors in API calls

When you set a thread name longer than 15 characters on a QNX platform, certain uses of the *Connext* API, such as creating a waitset by calling `DDS_WaitSet_new()` using the C API, failed with an error related to the name of the worker thread. This regression was introduced in release 6.1.2. You can once again set thread names up to the maximum allowed by the QNX platform.

[RTI Issue ID CORE-13827]

6.6.2 [Major] Wrong information in shared memory 'send' error log message

When the send operation of the shared memory transport failed due to the shared memory queue being full, the log message displayed information referring to the local shared memory transport instead of the information related to the shared memory transport that caused the failure, making the problem hard to debug. Now, the information in the message refers to the right transport.

[RTI Issue ID CORE-13832]

6.6.3 [Minor] take_discovery_snapshot APIs incorrectly always printed keyed_type as false *

Discovery snapshots' `keyed_type` was always printed as false even if the type of a specific row was keyed, which provided misleading information for debugging. Now the information provided by `take_discovery_snapshot` APIs is correct.

[RTI Issue ID CORE-13818]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.7 Content Filters and Query Conditions

6.7.1 [Critical] Error message printed for each filtered sample when using writer-side filtering, FlatData, and Zero Copy over shared memory *

When using FlatData, Zero Copy over shared memory, and writer-side filtering, every sample that was filtered incorrectly caused an error similar to the following to be logged:

```
ERROR [0x0101501F,0x044D6680,0xBF657AAB:0x80000002{Entity=DW,
Topic=Example CameraImage,Type=CameraImage,Domain=0}|WRITE]
REDAThresholdBufferPool_returnBuffer:!attempting to return a buffer
to a pool that it was not allocated from.
```

There was no impact on functionally, the samples were still filtered correctly.

[RTI Issue ID CORE-14144]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.8 TopicQueries

6.8.1 [Critical] Communication could stop when using bounded max_samples and TopicQueries

Communication may have stopped when using TopicQueries and setting the *DataReader's* **max_samples** QoS setting to less than 256 x (the number of *DataWriters* responding to the TopicQueries).

This situation could only happen if all of the following were also true:

- The *DataReader* was configured with KEEP_LAST history kind.
- The *DataReader* created multiple TopicQueries, and the responses to those TopicQueries contained some of the same samples.
- There were dropped samples.

If all of those conditions were met and, before the losses were repaired, the *DataReader's* queue filled up to the **max_samples** resource limit, then the *DataReader* would never accept any more samples into its queue.

[RTI Issue ID CORE-13784]

6.8.2 [Major] max_samples resource limit not honored in some cases when using an unkeyed topic and TopicQueries

If a *DataReader* created multiple TopicQueries and had a finite **max_samples** resource limit, that limit was not correctly enforced in TopicQuery queues. After some time, the *DataReader* may have accepted more samples than **max_samples** or failed to allocate more samples, leading to errors and lost samples. This only happened if the topic was unkeyed. Now, when a *DataReader* issues multiple TopicQueries, the **max_samples** resource limit is properly enforced for each of the TopicQuery queues correctly.

[RTI Issue ID CORE-14363]

6.9 Logging

6.9.1 [Major] Modern C++ Distributed Logger Options header incorrectly included generated header file *

The Modern C++ Distributed Logger implementation included a header file in `DistLoggerOptions.hpp` that should not be needed. The header file `distlogSupport.h` is no longer required.

[RTI Issue ID DISTLOG-237]

6.9.2 [Major] Misleading log message when sending specific number of bytes through socket *

In 7.2.0, a misleading warning was printed when *Connext* sent some specific number of bytes through the socket. The log message would look like, or similar to, this one:

```
NDDS_Transport_UDP_send:SENDING FAILURE | Inconsistent message size. Written␣
↪bytes (<inconsistent_value>) and bytes to send (<expected_value>)
```

The `<inconsistent_value>` could be 0 or a big number. This warning is now only printed when *Connext* has actually failed to send a message through the socket.

[RTI Issue ID CORE-14007]

6.9.3 [Major] Missing logging on the standard output for Windows GUI applications

Windows graphical user interface (GUI) applications using *Connext* libraries were not able to print log messages to a console through the standard output. The console was opened, but the messages did not appear. Log messages are now displayed properly in the console.

[RTI Issue ID CORE-14061]

6.9.4 [Minor] Incorrect error message when setting inconsistent ReaderDataLifeCycleQosPolicy values

The error messages that were printed when inconsistent values were used in the ReaderDataLifeCycleQosPolicy referenced the wrong fields. For example, when the value for `autopurge_nowriter_samples_delay` was set to a value that was out of bounds, the error message incorrectly referenced `autopurge_disposed_samples_delay`. These messages now reference the correct fields.

[RTI Issue ID CORE-14148]

6.9.5 [Minor] Log messages truncated below maximum size of 1024 bytes *

Some log messages were truncated below the maximum size of 1024 bytes if `NDDS_Config_LogPrintFormat` had been configured to print the timestamp, log level, thread ID, category, or activity context. Log messages are now logged with the full 1024 bytes.

[RTI Issue ID CORE-14105]

6.9.6 [Trivial] Error message that was printed when failing to allocate the writer buffer pool was wrong *

The error message that was printed when failing to allocate the writer buffer pool was wrong. It printed the incorrect resource limit value and the size of the buffers in the pool was always printed as 0.

Previous message:

```
PRESTypePluginDefaultEndpointData_createWriterPool:ALLOCATION FAILURE |
9 initial samples with size 0 in writer buffer pool. Consider setting
dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size
if your type has a large or unbounded max serialized size or reduce
initial_samples.
```

New message:

```
PRESTypePluginDefaultEndpointData_createWriterPool:ALLOCATION FAIL-
URE | 32 initial samples with size 2147482620 in writer buffer pool.
Consider setting dds.data_writer.history.memory_manager.fast_pool.
pool_buffer_max_size if your type has a large or unbounded max seri-
alized size or reduce initial_samples.
```

[RTI Issue ID CORE-14218]

6.9.7 [Trivial] Missing space between Activity Context and message text if Logging Category was printed *

If a log message belongs to the Security or Discovery category and your application logging was configured to print the Activity Context and the Logging Category, then there was a space missing between the end of the Activity Context and the beginning of the message. For example:

```
WARNING [0x01017774,0xFF40EEF6,0xEC566CA8:0x000001C1{Domain=2}
↪|ENABLE|LC:Discovery]NDDS_Transport_UDPv4_Socket_bind_with_ip:0X1EE6 in use
```

Now, a space is always printed:

```
WARNING [0x01017774,0xFF40EEF6,0xEC566CA8:0x000001C1{Domain=2}
↪|ENABLE|LC:Discovery] NDDS_Transport_UDPv4_Socket_bind_with_ip:0X1EE6 in use
```

[RTI Issue ID CORE-14262]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.10 Dynamic Data

6.10.1 [Major] DynamicData equals operation returned incorrect results for sequences of different lengths

When two DynamicData objects were created with a TypeCode representing a sequence type, and were compared using the `DynamicData::equals` operation or the == operator in C++, the results were incorrect if the sequences had different lengths. If instead the DynamicData objects were created with a TypeCode of structures that contained sequences, then the `equals` operation compared the sequences correctly and reported correct results when the sequences had different lengths.

[RTI Issue ID CORE-14288]

6.11 APIs (C or Traditional C++)

6.11.1 [Critical] Traditional C++ get_participants() API returned invalid pointer if Monitoring Library 2.0 was enabled

If *RTI Monitoring Library 2.0* was enabled in a Traditional C++ *Connext* application through the MonitoringQosPolicy, the `DDSDomainParticipantFactory::get_participants()` API returned an extra invalid *DomainParticipant* pointer, in addition to the *DomainParticipants* created by your application. Dereferencing that pointer had an undefined behavior, leading to a segmentation fault in the worst case.

This issue happened because the Traditional C++ *DomainParticipantFactory* instance is used to enable *Monitoring Library 2.0* and to create the dedicated *DomainParticipant* that the library uses. That is a C *DomainParticipant,* which doesn't have an associated C++ *DomainParticipant.* The `get_participants()` API added the nonexistent C++ Monitoring *DomainParticipant* anyway, which had an arbitrary value.

The `get_participants()` API now only returns valid *DomainParticipants* for C++.

[RTI Issue ID CORE-13925]

6.11.2 [Major] Potential error when waiting for samples in C API

When using one of the functions to wait for samples on either the Requester or the Replier, a potential error may have occurred in which the wait ended with an error instead of returning `DDS_RETCODE_TIMEOUT` when it timed out.

[RTI Issue ID REQREPLY-126]

6.12 APIs (Modern C++ API)

6.12.1 [Major] Potential crash when mixing the C and Modern C++ APIs in the same executable

Applications that used the C and modern C++ APIs in the same executable could crash when a *DomainParticipant* initially created using the C API was used to create a Modern C++ *Topic*.

This error is now detected and an exception is thrown.

Note that this is an uncommon use case. It has been documented in the Knowledge Base.

[RTI Issue ID CORE-14075]

6.12.2 [Major] DataReader created with builtin topic not automatically destroyed *

A regression in *Connext* 7.2.0 caused a user-created *DataReader* for the built-in topic `VirtualSubscriptionBuiltinTopicData` to not be deleted by its destructor. Calling `close()` explicitly still worked. The *DataReader* destructor now works as expected in this case.

[RTI Issue ID CORE-14005]

6.12.3 [Major] Possible link error when building a Windows DLL *

Trying to build a Windows DLL that uses the Modern C++ API may have failed with the following link error:

```
error LNK2019: unresolved external symbol DDS_LENGTH_AUTO
```

[RTI Issue ID CORE-14431]

6.12.4 [Minor] Conversion of invalid Time to integer units caused unexpected behavior

Given a Time object with value `Time::invalid()`, the conversion to integer units (such as `to_microsecs()`) caused unexpected behavior.

- Before Connext 7.2: the operations returned a large integer value.
- In Connext 7.2: the operations threw an unexpected exception.

These operations now return 0 (zero) when the value is `Time::invalid()`.

[RTI Issue ID CORE-14283]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.13 APIs (Java)

6.13.1 [Major] DynamicData API now supports setting and getting wchar fields

The DynamicData Java API did not provide a way to use `wchar` data fields. It now allows working with wide character (`wchar`) fields, including sequences and arrays. Specifically, the implementation of the following APIs was enhanced:

- `set_char` and `get_char`: now accept `char` values representing a `wchar`.
- `set_char_array` and `get_char_array`: now accept `char[]` with values representing a `wchar`.
- `set_char_seq` and `get_char_seq`: now accept `WcharSeq`.

Wide characters in the context of the Java API are restricted to Unicode BMP characters—that is, characters representable as 16-bit integers. These map directly to Java's `char` primitive type, a 16-bit unsigned value, and so a new API was not needed.

[RTI Issue ID CORE-5933]

6.13.2 [Major] Possible data serialization error for keyed DataReaders using XCDR2 format

Java applications subscribing to some data types with keys may have printed an error such as the following:

```
Exception in thread "Thread-7" com.rti.dds.cdr.IllegalCdrStateException: not␣
↪enough available space in CDR buffer
```

This may have caused the *DataReader* to not receive data.

This problem only affected certain non-mutable types when the XCDR2 format was used (the default is XCDR).

[RTI Issue ID CORE-13888]

6.13.3 [Minor] "data_to_string" of DynamicDataTypeSupport failed with exception

When using the `DynamicDataTypeSupport`'s `data_to_string` operation, an exception was raised with the following error:

```
java.lang.IllegalStateException: (de)serialization of dynamic types should␣
↪take place in native code
```

While this made it impossible to transform `DynamicData` samples into strings using `DynamicDataType-Support`, it was still possible to transform `DynamicData` samples into strings using the instance method `to_string`.

[RTI Issue ID CORE-14325]

6.14 APIs (Python)

6.14.1 [Critical] Potential deadlock in applications that call certain APIs and use Entity Listeners

Applications that installed an *Entity Listener* (for example, with `DataReader.set_listener`) and called certain APIs may have deadlocked. The affected APIs were the following three properties: `Entity.status_changes` , `Condition.trigger_value`, and `DataWriter.matched_subscriptions_locators`.

[RTI Issue ID PY-114]

6.14.2 [Minor] Possible memory leak in DynamicData.loan_value

The DynamicData method `loan_value` may have leaked memory when the argument to `loan_value` was a nested field name as shown in this example:

```
dynamic_data.loan_value("foo.bar")
```

Using index numbers or non-nested names, such as "foo," as the argument didn't cause the leak.

[RTI Issue ID PY-133]

6.14.3 [Minor] Converting a SampleInfo object to string failed when source_timestamp was invalid

Given a SampleInfo object `info`, calling `str(info)` threw an exception if `info.source_timestamp.sec < 0` or `info.source_timestamp == Time.invalid`.

Now `str(info)` will simply omit the `source_timestamp` when it's invalid.

[RTI Issue ID PY-154]

6.14.4 [Minor] Some functions didn't allow keyword arguments

The following functions in the Python API didn't support keyword arguments: `DataTag.set`, `Logger.verbosity_by_category`, `Logger.output_file` and `TopicQuery.find`. They now allow them.

[RTI Issue ID PY-143]

6.15 APIs (Multiple Languages)

6.15.1 [Major] Using a Listener and a Waitset in the same application may have resulted in the Waitset waking up unexpectedly

In applications where both a Waitset and a Listener are used on the same *DataReader*, if the application did not provide a Listener upon *Entity* creation, but later used the `DDS_DataReader_set_listener` API, the Waitset may have repeatedly woken unexpectedly.

[RTI Issue ID CORE-11125]

6.15.2 [Major] Sentinel constant for "invalid" Time contained unexpected value *

Due to a regression in 7.2.0, the sentinel Time value that indicates an "invalid" timestamp was incorrect and didn't match the value of an "invalid" `SampleInfo::source_timestamp` (which can be returned when an instance state is NOT_ALIVE_NO_WRITERS). This made a comparison such as the following to never be true:

```
if (info.source_timestamp() == Time::invalid()) {} // never true due to this␣
↪bug
```

`Time::invalid()` now contains the expected sentinel value.

[RTI Issue ID CORE-14334]

6.15.3 [Minor] IDL printing of Enum TypeCodes was not standards-compliant

The `DDS_TypeCode_to_string` API can be used to print a `DDS_TypeCode` in IDL representation. When used to print enums, TypeCodes were printed in a non-standard format, as follows:

```
enum MyEnum {
  ZERO = 0,
  ONE = 1
};
```

The following format is mandated in the OMG 'Interface Definition Language' specification, version 4.2:

```
enum MyEnum {
  @value(0) ZERO,
  @value(1) ONE
};
```

Now, enums are printed using the @value annotation.

[RTI Issue ID CORE-13956]

6.15.4 [Minor] Extensibility of unions defined within modules incorrectly printed as IDL *

When a union that was defined within a module was printed, the extensibility of that union was printed at the scope of the module, instead of the union.

[RTI Issue ID CORE-13945]

6.15.5 [Minor] Incorrect output when printing a union with an enum discriminator as IDL

The `DDS_TypeCode_to_string` API allows TypeCodes to be printed as IDL. When printing a union that had an enum as its discriminator, the case labels may have been incorrectly printed, if the enum case labels were not used in the same order in which they were defined in the enum. For example:

```
enum MyEnum {
  RED,
  GREEN,
  YELLOW
};

union MyUnion switch (MyEnum) {
case YELLOW:
  char x;
case GREEN:
case RED:
  uint32 data;
};
```

The union would have been printed with incorrect case labels, since the case labels appeared in an order different than the order in which they were defined.

[RTI Issue ID CORE-13941]

6.15.6 [Trivial] First enum label not printed

By default, when printing TypeCodes (using the `DDS_TypeCode_to_string` APIs), the ordinal values associated with an enum are only printed if they are explicitly provided in the type definition. There was a bug where the first ordinal value was never printed (even if it was explicitly supplied in the type definition).

[RTI Issue ID CORE-14400]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.16 XML Configuration

6.16.1 [Critical] Potential segmentation fault when using XML application creation if the names of <domain_participant_library> and <domain_library> were the same

A bug in the XML parser meant that an application may have suffered a segmentation fault if it used XML application creation, and the XML tags `<domain_participant_library>` and `<domain_library>` had the same value for their name attribute. The segmentation fault would occur when creating a second participant from that XML file. This could have happened when using XML Application Creation directly or indirectly (for example, via *RTI Connector*).

This problem has been resolved. If an XML file that would have caused the crash is used, an error message is now produced and the *DomainParticipant* is not created. Using the same name for multiple XML objects is currently not supported.

[RTI Issue ID CORE-13692]

6.16.2 [Major] Micro Compatibility Builtin Profiles updated UDPv4 message_size_max so that samples larger than 8192 were not silently dropped by Micro applications

The Generic.ConnextMicroCompatibility, Generic.ConnextMicroCompatibility.2.4.9, and Generic.ConnextMicroCompatibility.2.4.3 builtin QoS profiles have been updated to use a UDPv4 `message_size_max` of 8192, because that is what *Connext Micro* 2.x versions use. By default, *Connext Professional* uses a larger UDPv4 `message_size_max` of 65507. If messages larger than 8192 were sent by *Connext Professional*, these messages would be dropped by the *Connext Micro* application. The new QoS profile values ensure that this will not happen.

[RTI Issue ID CORE-12749]

6.16.3 [Minor] XML parser did not parse scientific notation

The XML parser failed with an error message when parsing numeric literals expressed in scientific notation.

[RTI Issue ID CORE-10767]

6.17 Instances

6.17.1 [Critical] Two log messages used memory after it was freed *

When the instance state consistency feature was enabled and the logging level was set to local, two log messages could have been generated that used memory after the memory had been freed.

[RTI Issue ID CORE-14475]

6.17.2 [Major] Instance state consistency may not have worked for DataReaders using multiple data representations *

Instance state consistency updates may not have been applied for *DataReaders* using multiple data representations. Note that the state of the instance would only have been incorrect if the *DataReader* was using a Durability of VOLATILE and the instance of interest had not changed state during the disconnection.

[RTI Issue ID CORE-14076]

6.17.3 [Major] Indeterminate instance state in systems with multiple DataWriters *

Instance state consistency is used to ensure that, upon recovering from a disconnection, *DataReaders* have the latest instance state data for each instance. In scenarios where a *DataReader* lost liveliness with multiple *DataWriters*, there may have been some indeterminism in the final state, depending on which *DataWriter* the *DataReader* recovered liveliness with first. This issue has been resolved for *DataReaders* that have a DestionationOrderKind of BY_SOURCE_TIMESTAMP. Upon recovering liveliness, the eventual state of each instance will be the latest state across all *DataWriters* with which liveliness has been recovered.

[RTI Issue ID CORE-13740]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.18 Crashes

6.18.1 [Critical] Crash when deserializing PID_TYPE_OBJECT_LB with class ID of RTI_OSAPI_COMPRESSION_CLASS_ID_NONE

A *DomainParticipant* would crash when deserializing an endpoint discovery message containing PID_TYPE_OBJECT_LB that was followed by a compression class ID of RTI_OSAPI_COMPRESSION_CLASS_ID_NONE (0x00000000). This should not have occurred in normal operation but was possible to encounter if a packet had been tampered with or corrupted. Now the *DomainParticipant* does not crash but will log an error message and not deserialize the compressed type object.

[RTI Issue ID CORE-14079]

6.18.2 [Critical] Potential crash while calling DynamicData APIs when running out of system memory

Running out of memory during certain DynamicData initialization API calls, such as `DDS_DynamicDataTypeSupport_new`, may have resulted in a crash. Now, running out of memory during DynamicData initialization APIs will provoke those APIs to gracefully fail.

[RTI Issue ID CORE-14232]

6.18.3 [Critical] Potential crash when calling DDS_TypeCodeFactory_create_value_tc_ex with a NULL ex parameter

Performing a call to `DDS_TypeCodeFactory_create_value_tc_ex` with a `NULL` `ex` parameter (which is a valid input value) may have resulted in a crash. Specifically, the crash would have been triggered if passing a not `NULL` `concrete_base` and an exception occurred. `DDS_TypeCodeFactory_create_value_tc_ex` no longer crashes if passing a `NULL` `ex` parameter.

[RTI Issue ID CORE-14224]

6.18.4 [Critical] Crash when calling DDS_DataWriter_set_qos with a NULL qos parameter

Performing a call to `DDS_DataWriter_set_qos` with a `NULL` `qos` parameter resulted in a crash. Now, an illegal call to `DDS_DataWriter_set_qos` will gracefully fail and return `DDS_RETCODE_BAD_PARAMETER`.

[RTI Issue ID CORE-14223]

6.18.5 [Critical] Crash when performing an illegal call to DDS_DataWriter_get_qos

Performing an illegal call to `DDS_DataWriter_get_qos` (see Restricted Operations in Listener Callbacks in the *RTI Core Libraries User's Manual*) resulted in a crash. Now, an illegal call to `DDS_DataWriter_get_qos` will gracefully fail and return `DDS_RETCODE_ILLEGAL_OPERATION`.

[RTI Issue ID CORE-14222]

6.18.6 [Critical] Crash if SPDP2 participant received unexpected field in participant discovery message *

This issue was fixed in 7.2.0, but not documented at that time.

A participant using Simple Participant Discovery 2.0 would crash if it received a bootstrap or configuration message with a PID that it was unable to deserialize. This occurred with a subset of PIDs. The participant would crash if it received a bootstrap message with any of the following fields present: `PID_PROPERTY_LIST`, `PID_USER_DATA`, `PID_ENTITY_NAME`, `PID_ROLE_NAME`. The participant would crash if it received a configuration message with any of the following fields present: `PID_DOMAIN_TAG`, `PID_IDENTITY_TOKEN`, `PID_PERMISSIONS_TOKEN`, `PID_TRANSPORT_INFO`.

Now if these fields are unexpectedly present in a participant discovery message, the participant will not attempt to deserialize them and will simply skip the field.

[RTI Issue ID CORE-13695]

6.18.7 [Critical] Crash during DomainParticipant initialization if failure to get local address mapping when using UDPV4_WAN transport

Using UDPV4_WAN when creating a *DomainParticipant* may have resulted in a crash if *Connext* failed to get the local address mapping.

[RTI Issue ID CORE-14272]

6.18.8 [Critical] Crash when converting a DynamicData object to a CDR buffer

If a DynamicData object was bound to another DynamicData object or was populated as a result of a call to `DynamicData::get_complex_member`, and the top-level DynamicData object was an unbounded type, any attempt to convert the nested DynamicData object to a CDR buffer resulted in a crash.

For example, using the following types:

```
struct MyA {
    string str;
};

struct MyB {
    MyA my_a;
};
```

Creating a DynamicData object with type `MyB`, then getting a DynamicData object for member `my_a`, and finally converting that DynamicData object to a serialized buffer in CDR format, resulted in a crash.

[RTI Issue ID CORE-14167]

6.18.9 [Critical] Potential crash if allocation of RTI Monitoring Library's publish thread failed

A crash could occur if the allocation of the *RTI Monitoring Library's* publish thread failed. Now, the creation of the thread will gracefully fail.

[RTI Issue ID MONITOR-644]

6.18.10 [Critical] Segmentation fault upon destruction of DDSGuardCondition or DDSWaitset

When using the Traditional C++ or Modern C++ API, a segmentation fault occurred if the *DomainParticipant-Factory* was finalized in the same scope as stack-allocated DDSGuardCondition or DDSWaitset instances.

[RTI Issue ID CORE-8967]

6.18.11 [Critical] Crash if participant received endpoint discovery sample and was not able to allocate memory to process it

If a *DomainParticipant* received an endpoint discovery sample and was unable to allocate the memory to properly process it (for example, if the system was out of memory), the *DomainParticipant* may have crashed.

[RTI Issue ID CORE-14342]

6.18.12 [Critical] Possible exception after using a Condition object if it was not explicitly disposed

An exception may have occurred after using a `ReadCondition` or `GuardCondition` if you did not explicitly dispose it. This may have also affected the methods `TakeReplies(SampleIdentity relatedRequestId)` and `ReadReplies(SampleIdentity relatedRequestId)` from the C# Request-Reply API.

[RTI Issue ID CORE-14154]

6.18.13 [Critical] Potential crash or errors when using SHMEM transport in QNX *

If you used the shared-memory (SHMEM) transport in *Connext* 7.2.0, you may have seen unexpected errors or crashes in your applications during startup. The errors were more likely to occur when all the applications in the system were started at the same time.

[RTI Issue ID CORE-14038]

6.18.14 [Critical] Crash if participant failed to allocate memory for endpoint discovery type plugins

If a *DomainParticipant* failed to allocate memory for the Simple Endpoint Discovery plugins (for example, because the system was out of memory), the *DomainParticipant* crashed while starting up.

[RTI Issue ID CORE-14343]

6.18.15 [Critical] Modern C++ Distributed Logger may hang or crash upon instance finalization *

The Modern C++ Distributed Logger may have produced a crash after calling `DistLogger::finalize()` when a *DomainParticipant* had been set by the user.

[RTI Issue ID DISTLOG-238]

6.18.16 [Critical] Invalid multicast locator could cause precondition error or segmentation violation

When particular combinations of Data(p) messages were received by a *DomainParticipant*, a precondition error (using the debug version of our libraries) or a segmentation violation (using the release version of our libraries) occurred. Such a combination of messages is beyond the scope of regular system operations and could only arise through the manipulation of Data(p) RTPS messages. Such manipulation might have stemmed from internal testing scenarios or unauthorized access by a malicious entity in systems where DDS security measures were not fully implemented or enforced.

There were various combinations of Data(p) messages that triggerred this behavior, but the essential scenario required the discovery by a *DomainParticipant* of at least two other *DomainParticipants*. One of these discovered *DomainParticipants* initially used a multicast locator, but later switched to unicast locators due to a corrupted Data(p) RTPS message. The other discovered *DomainParticipant* used unicast locators and was discovered before the first *DomainParticipant* switched to unicast locators.

[RTI Issue ID CORE-14349]

6.18.17 [Critical] Crash during DomainParticipant enable operation when running out of system memory

Running out of memory during *DomainParticipant* `enable()` may have resulted in a crash. Now, running out of memory during *DomainParticipant* `enable()` provokes the enable operation to gracefully fail.

[RTI Issue ID CORE-14220]

6.18.18 [Critical] Segmentation fault when a reader was deleted while a remote writer cleanup event was scheduled *

To prevent unbounded memory growth when the Instance State Consistency feature is enabled, *DataReaders* periodically purge information associated with *DataWriters* that are no longer communicating with the *DataReader*. If the purge event ran for a deleted *DataReader*, a segmentation violation occurred.

[RTI Issue ID CORE-14438]

6.18.19 [Critical] Race condition between the creation of a Replier and the call to its Listener

A race condition may have caused a ReplierListener to be called in a state where the Replier was not fully created, potentially causing a crash or an exception. This issue has been resolved in the Modern C++ and Java APIs. A fix for other APIs is expected soon.

[RTI Issue ID REQREPLY-132]

6.18.20 [Critical] Undefined behavior when Requesters or Repliers for same service name were concurrently created and deleted

A race condition in the creation and destruction of the Requester type may have caused a failure or crash when several Requesters for the same service name were created or deleted in different threads. The Replier type was also affected.

Protecting the creation and destruction of these objects with a mutex resolved the problem. This issue has been resolved in the Modern C++ and Java APIs. A fix for other APIs is expected soon.

[RTI Issue ID REQREPLY-127]

6.18.21 [Critical] Hang led to crash if Monitoring Library 2.0 was enabled then right away disabled *

When *Monitoring Library 2.0* is enabled, it creates a set of threads used for collecting and publishing telemetry data.

If the library was disabled right after enablement, there was a chance that one of these threads was still setting up all the components it needs to operate. Setting up these components required taking a semaphore that was already taken by the disablement operation. At the same time, the disablement operation was waiting for that thread to finish. This situation led to a hang.

The hang was not indefinite. The disablement of the library waits for a fixed period of time for the thread to finish. After that period, the disablement continues regardless of whether the thread was stopped. Once the disable operation released the semaphore, the thread that was waiting for it continued with its execution, accessing already freed memory and producing a crash.

Before the crash, these errors occurred:

```
ERROR RTIOsapiJoinableThread_shutdown:Join timeout (20000 millisec) for␣
↪thread expired
ERROR RTI_MonitoringEventSnapshotThread_finalize:FAILED TO FINALIZE |␣
↪Monitoring Event Snapshot Thread
```

The hang is now fixed.

[RTI Issue ID MONITOR-664]

6.18.22 [Critical] Possible crash when creation of TCP Transport failed

A failure in the creation of the TCP Transport could lead to a crash while trying to finalize the partially created transport. This error is now managed without crashing.

[RTI Issue ID COREPLG-731]

6.18.23 [Critical] Possible crash upon destruction of TCP transport if it was created programmatically and it logged messages

When creating the TCP Transport plugin object programmatically, a crash could happen when the transport was destroyed, if the transport produced log messages during its execution. The crash did not happen when the transport was created through QoS configuration. Programmatic creation of the transport now works properly.

[RTI Issue ID COREPLG-718]

** This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.19 Hangs

6.19.1 [Critical] Undefined behavior when using SHMEM transport in Linux, macOS, QNX, INTEGRITY, and LynxOS

There was an issue in the SHMEM transport implementation that may have led to undefined behavior in your *Connext* application, including data corruption, errors, and hangs.

When the undefined behavior was a hang, you may have seen the following stack trace in some of the threads in your process:

```
#1 0x00007f42d1348d39 in RTIOsapiSharedMemorySemMutex_take_os () from␣
↪libnddscore.so
#2 0x00007f42d134938f in RTIOsapiSharedMemorySemMutex_take () from␣
↪libnddscore.so
#3 0x00007f42d14a468a in NDDS_Transport_Shmem_send () from libnddscore.so
```

The problem could occur on the following platforms: Linux, macOS, QNX, INTEGRITY, and LynxOS.

[RTI Issue ID CORE-12923]

6.19.2 [Major] Possible deadlock related to failures in DNS Tracker

In some cases there could have been a deadlock after an error occurred starting the DNS tracker or removing a hostname from an internal list. The error conditions were not handled appropriately, and a lock which had been previously taken was never released.

[RTI Issue ID CORE-13813]

6.19.3 [Critical] Segmentation fault or hang when using SHMEM transport on VxWorks 6 or higher platforms

A local *DomainParticipant* in a *Connext* application on a VxWorks 6 or higher platform, using the shared memory transport, may have generated a segmentation fault or hang when a remote *DomainParticipant* communicating with the local *DomainParticipant* was deleted or lost its liveliness. If you configured individual *DataWriters* and *DataReaders* on the remote *DomainParticipant* to use their own receive port using the TransportUnicastQosPolicy and TransportMulticastQosPolicy, the problem could have also occurred when you deleted these *DataReaders* or *DataWriters*.

In some cases, before the segmentation fault or hang, you may have seen the following error message:

```
RTIOsapiSharedMemorySegment_detach:OS sdUnmap() failure, error
0XBE000A: S_sdLib_NOT_MAPPED
```

[RTI Issue ID CORE-14041]

6.20 Memory Leaks/Growth

6.20.1 [Critical] Memory leak in best-effort writers when switching from more than one unicast locator to a multicast locator

Each time a best-effort *DataWriter* transitioned from using more than one unicast locator to using a multicast locator, a small amount of memory leaked. Such a transition is beyond the scope of regular system operations and can only arise through the manipulation of Data(p) RTPS messages. Such manipulation might stem from internal testing scenarios or unauthorized access by a malicious entity in systems where DDS security measures have not been fully implemented or enforced.

[RTI Issue ID CORE-14427]

6.20.2 [Critical] Concurrency problem in Asynchronous WaitSet's global instance initialization led to memory and TSS key leaks in multi-threading scenarios

The first time an *Asynchronous WaitSet* is created, *Connext* initializes a global singleton instance shared among all the *Asynchronous WaitSets* in your application. That global instance contains some fields used in the *Asynchronous WaitSet* logic, like a thread-specific storage (TSS) key.

Due to a concurrency problem, several threads could initialize that global singleton instance at the same time, without realizing that the instance was already initialized. Some existing pointers were overwritten with new ones, resulting in a leak of the TSS key and a memory leak similar to the following:

```
==44591== 32 bytes in 1 blocks are definitely lost in loss record 1 of 1
==44591==    at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/
↪vgpreload_memcheck-amd64-linux.so)
==44591==    by 0x49789AD: RTIOsapiHeap_reallocateMemoryInternal (heap.c:830)
==44591==    by 0x4A8DF06: REDAWorkerPerThread_newWithTss (Worker.c:1599)
==44591==    by 0x99D63F: DDS_AsyncWaitSetGlobals_initializeConcurrency␣
↪(AsyncWaitSetGlobals.c:247)
==44591==    by 0x99DB9E: DDS_AsyncWaitSetGlobals_get_instance␣
↪(AsyncWaitSetGlobals.c:351)
...
```

This issue could happen if your application had several threads creating *Asynchronous WaitSets* (or calling `DDS_DomainParticipantFactory_unregister_thread`) simultaneously and for the first time.

If this process was repeated enough times in a loop after calling `DDS_DomainParticipantFactory_finalize_instance` (that deletes the global singleton instance), theoretically, this concurrency problem could lead to an unbounded memory growth or to reaching the maximum limit of TSS keys available in your operating system (the consequences of reaching that limit are explained in CORE-14157). In practice, the timing for this race condition to happen was highly unlikely.

The concurrency issue is now fixed and the threads do nothing if the global singleton instance is already initialized.

[RTI Issue ID CORE-14321]

6.20.3 [Critical] Memory leak when creating a QueryCondition with Parameters

When creating a new QueryCondition with Parameters using the `create_querycondition_w_params` API, some memory was leaked.

[RTI Issue ID CORE-14301]

6.20.4 [Critical] Memory leak when using NetworkCaptureParams

When using the Java API, some native memory could be leaked when starting Network Capture with non-default parameters or when setting new default parameters. This leak was restricted to using parameters, and did not affect Network Capture execution itself.

[RTI Issue ID CORE-14275]

6.20.5 [Critical] Memory Leak in Java API when printing QoS objects

When using the Java API, printing QoS objects could have caused some native memory to be leaked. This issue affected any QoS field of variable-length, such as Entity Name; therefore, it affected the following:

- `DomainParticipantQos`
- `PublisherQos`
- `SubscriberQos`
- `DataWriterQos`
- `DataReaderQos`

[RTI Issue ID CORE-14252]

6.20.6 [Critical] Asynchronous WaitSet global instance's thread-specific storage key leaked

The first time an *Asynchronous WaitSet* is created, *Connext* initializes a global singleton instance shared among all the *Asynchronous WaitSets* in your application. That global instance contains some fields used in the *Asynchronous WaitSet* logic, like a thread-specific storage (TSS) key.

The *Asynchronous WaitSet* global instance is deleted when calling `DDS_DomainParticipantFactory_finalize_instance`. However, the TSS key was never deleted, resulting in a leak of the key.

If, for some reason, your application created *AsyncWaitSets* and finalized the *DomainParticipantFactory* instance enough times in a loop, you could eventually reach the limit of available TSS keys imposed by your operating system. Once that limit was reached, next attempts of *Connext* to create a TSS key resulted in failure:

```
ERROR DDS_AsyncWaitSetGlobals_initializeConcurrency:ERROR: Failed to create␣
↪thread-specific storage for WSCT
ERROR DDS_AsyncWaitSetGlobals_get_instance:!init concurrency
ERROR DDS_AsyncWaitSet_newI:!init DDS_AsyncWaitSet
```

Not only *AsyncWaitSets* use TSS keys in *Connext*. If your system ran out of keys because of this leak, the issue could affect any other *Connext* feature relying on thread-specific storage.

The TSS key is now always deleted when calling `DDS_DomainParticipantFactory_finalize_instance`.

Note: Even if your application doesn't use *Asynchronous WaitSets*, you could have been affected by this issue if you called `DDS_DomainParticipantFactory_unregister_thread`. This function also initializes the global singleton if it is not already initialized.

[RTI Issue ID CORE-14157]

6.20.7 [Major] Memory leak when using XML-Based Application Creation and DynamicData

When using *XML-Based Application Creation* and DynamicData, a small amount of memory was leaked for every DynamicData type registered in a *DomainParticipant*, upon *DomainParticipant* deletion.

[RTI Issue ID CORE-14163]

6.20.8 [Minor] Memory leak when finalizing DomainParticipantFactory for first time

When finalizing the `DomainParticipantFactory` for the first time, and only for the first time, some native memory was never being freed. The leak could be observed when using `HeapMonitoring.take_heap_snapshot`. For example:

```
block_id, timestamp, block_size, alloc_method_name, type_name, pool_alloc,␣
↪pool_buffer_size, pool_buffer_count, topic_name, function_name, activity_
↪context
359, 1702981391, 48, RTIOsapiHeap_allocateStructure, struct␣
↪REDAObjectPerWorker, MALLOC, 0, 0, unknown, unknown, unknown
```

[RTI Issue ID CORE-14180]

6.20.9 [Minor] DomainParticipantFactory was leaked when factory finalization failed

When an error occurred during the finalization of the *DomainParticipantFactory* (for example, because some *DomainParticipants* were not deleted), the *DomainParticipantFactory* itself would have been leaked.

[RTI Issue ID CORE-14302]

6.20.10 [Minor] Potential Memory leak upon ContentFilteredTopic creation failure

When creating a ContentFilteredTopic with a custom filter, a native memory leak could occur if the `compile` function of the filter failed with a Java exception.

[RTI Issue ID CORE-14333]

6.20.11 [Critical] ODBC DataWriters may have leaked instances when they were replaced if writer-side filtering was used

DataWriters configured to use an ODBC database may have leaked instances when they were replaced. The instances were leaked if writer-side filtering was used, and a sample for that instance was filtered on the writer-side.

[RTI Issue ID CORE-14434]

6.21 Data Corruption

6.21.1 [Critical] Undefined behavior using XCDR2 with keyed topic types with key union members

XCDR2 with keyed topic types with key union members was not supported. For example:

```
union MyUnion switch(long) {
   case 0:
       long m_long;
   case 1:
       short m_short;
};

struct StructWithUnionKey {
   @key MyUnion m_union;
   long m_long;
};
```

The behavior was undefined if any of your topic types had a union key member going from a potential segmentation fault to an erroneous key hash in which two different instances could be considered equal.

[RTI Issue ID CORE-14186]

6.21.2 [Critical] Stack overflow if value of "rti.monitor.config.publish_thread_options" property had 512 or more characters

If *RTI Monitoring Library* was enabled for a *DomainParticipant* and the `rti.monitor.config.publish_thread_options` property was specified with a string value of 512 or more characters, a stack overflow happened in the application. Now, if the property value has 512 characters or more, an error will be printed.

[RTI Issue ID MONITOR-643]

6.21.3 [Critical] Failure to send serialized key with dispose when using dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size property

You may have seen a serialization error when disposing instances. This error occurred under specific conditions: when the `writer_qos.protocol.serialize_key_with_dispose` setting was enabled (set to TRUE) and the `dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size` was configured to a size that required the serialization buffer to be allocated from the heap instead of from pre-allocated memory.

This error only occurred in cases where the serialized key was not aligned on a 4-byte boundary.

[RTI Issue ID CORE-14370]

6.21.4 [Critical] Error uncompressing samples when using batching and setting serialize_key_with_dispose to TRUE

A *DataReader* failed to uncompress batch samples when the *DataWriter* set the QoS **writer_qos.protocol.serialize_key_with_dispose** to TRUE, and the batch sample contained one or more dispose messages.

When this problem occurred, the *DataReader* printed an error like this:

```
ERROR RTIOsapi_Zlib_uncompress:The input data was corrupted
ERROR RTICdrStream_uncompress:!uncompress sample
ERROR PRESReaderQueue_decodeAndUncompress:FAILED TO TRANSFORM | Stream␣
↪decompression failed.
ERROR PRESCstReaderCollator_newData:FAILED TO TRANSFORM | Failed to decode␣
↪and/or uncompress sample.
```

[RTI Issue ID CORE-14344]

6.21.5 [Critical] SampleInfo's flag and related_original_publication_virtual_guid may have had invalid information for unkeyed Topics

If unkeyed *Topic* samples had **valid_data** in the SampleInfo set to FALSE and **instance_state** set to NOT_ALIVE_NO_WRITERS_INSTANCE_STATE, the *DataReader* provided incorrect information in the **flag** and **related_original_publication_virtual_guid** fields of the SampleInfo for those samples.

[RTI Issue ID CORE-14260]

6.21.6 [Critical] DataReader on a Topic using an appendable type may have received samples with incorrect value

A *DataReader* subscribing to a *Topic* on an appendable type may have received incorrect samples from a matching *DataWriter*.

The problem only occurred when the *DataWriter* published a type with fewer members than the *DataReader* type. For example, consider a *DataWriter* on FooBase and a *DataReader* on FooDerived:

```
@appendable struct FooBase {
    sequence<uint8,1024>base_value;
};

@appendable struct FooDerived {
    sequence<uint8,1024> base_value;
    @default(12) uint8 derived_value;
};
```

When the *DataWriter* published a sample with type `FooBase`, in some cases the *DataReader* received a sample in which the field derived_value was set to 0 instead of 12.

This issue was caused by a bug in which *Connext* was not setting the padding bits in the encapsulation header for a serialized sample as required by the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. As a result, some of the padding bytes were interpreted as data.

Note: This fix may lead to a compatibility issue causing a *Connext Professional DataWriter* to not match with a *Connext Micro* or *Connext Cert DataReader*. For details, see Extensible Types Compliance Mask in the *RTI Connext Core Libraries Extensible Types Guide*.

[RTI Issue ID CORE-9042]

6.22 OMG Specification Compliance

6.22.1 [Critical] Extensible types did not include padding size in length value of each element of RTPS parameter list

Previously, *Connext* did not follow the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3 with regards to the Parameterized CDR Encoding. For each element of the RTPS Parameter list, the specification states that any padding bytes that may follow a serialized member won't be taken into account in the parameter length of the member.

Connext now follows this specification; however, to be compatible with previous releases and not break backward compatibility, *Connext* does not follow the specification by default. To enable this fix and follow the specification, you have to unset the bit `NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_BIT(0x00000004)`. For details, see Extensible Types Compliance Mask in the *RTI Connext Core Libraries Extensible Types Guide*.

Note that *Connext*'s non-compliance with the specification did not break functional correctness, it only affected interoperability with other vendors who were compliant with the specification.

[RTI Issue ID CORE-8640]

6.22.2 [Critical] Problems exchanging data with other vendors for types containing unbounded members

Connext did not follow the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.1 with regards to treating a 0 bound as UNBOUNDED when received as part of the TypeObject v1 for an endpoint. (TypeObject v1 was replaced with TypeObject v2 in version 1.2, but *Connext* does not support TypeObject v2 yet.) Therefore, you may have encountered issues when exchanging data with other vendors for types containing unbounded collections. For example:

```
struct MyType {
    string m1;
    sequence<long> m2;
};
```

The use of this unbounded type led to specific issues:

Deserialization Errors with DynamicData Readers

1. When using `MyType`'s `TypeCode` from the endpoints discovered from other vendors (type_code field in `PublicationBuiltinTopicData` and `SubscriptionBuiltinTopicData`) to create a DynamicData *DataReader*, you may have seen errors in deserializing data coming from *DataWriters*.

Incompatibility in DataReader and DataWriter Matching with Type Coercion

2. If `reader_qos.type_consistency.kind` was set to `ALLOW_TYPE_COERCION` in a *DataReader* for `MyType` from a different vendor, a *Connext DataWriter* for the same type failed to match with the *DataReader*.

3. Conversely, when `reader_qos.type_consistency.kind` was set to `DISALLOW_TYPE_COERCION`, a *Connext DataReader/DataWriter* for `MyType` wouldn't match with a *DataReader/DataWriter* from another vendor.

This update resolves the first two issues. The third issue, involving matching problems when disallowing type coercion, is targeted for resolution in future releases. See *Other Known Issues*.

[RTI Issue ID CORE-14185]

6.22.3 [Critical] Non-primitive sequences and arrays serialized incorrectly with XCDR2_DATA_REPRESENTATION when using dds.type_plugin.dheader_in_non_primitive_collections

The property `dds.type_plugin.dheader_in_non_primitive_collections` was added in 7.0.0 to allow for compliance with the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. When this property was enabled, an additional DHEADER was serialized after the member header. However, the fix was incomplete, and there were still cases where setting `dds.type_plugin.dheader_in_non_primitive_collections` to TRUE did not lead to specification compliance.

This problem has been fixed. In addition, the property `dds.type_plugin.dheader_in_non_primitive_collections` has been removed and replaced with the ability to set an Extensible Types compliance mask. To allow compliance with the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, for XCDR 2, you will need to set the bit `NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT` instead of setting the property. For details, see Extensible Types Compliance Mask in the *RTI Connext Core Libraries Extensible Types Guide*.

[RTI Issue ID CORE-13906]

6.22.4 [Critical] DataReader on a Topic using an appendable type may have received samples with incorrect value

See RTI Issue ID CORE-9042 in *Data Corruption*.

6.22.5 [Major] FlatData did not support XCDR2-compliant serialization

The serialization/deserialization of sequences and arrays with non-primitive members for FlatData did not follow the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. This led to compatibility issues with other DDS implementations.

This problem has been fixed, although the new behavior is not enabled by default, in order to keep backward compatibility with previous *Connext* releases. If interoperability with other vendors is a concern, you can enable the new standard-compliant serialization by setting the `rti::config::compliance::XTypesMask::dheader_in_non_primitive_collections()` compliance mask. For information, see the API Reference HTML documentation. (For example, see the Modern C++ API Reference here.)

[RTI Issue ID CORE-13944]

6.23 Entities

6.23.1 [Critical] FlatData language binding allowed you to specify XCDR data representation

Since the debut of the *RTI FlatData™ language binding* in release 6.0.0, you have been able to specify XCDR as the data representation in Quality of Service (QoS) settings for entities that use FlatData, although the only valid data representation for FlatData types is XCDR2. When XCDR was selected in these scenarios, the system would usually issue a warning and convert the data representation to XCDR2 to maintain functionality. While this process was intended to safeguard against configuration errors, it could lead to confusion about the actual data representation being used and could cause a segmentation fault. Note that the FlatData language binding is only valid for the Traditional C++ and Modern C++ APIs.

To ensure clarity and system integrity, the QoS validation logic has been refined. From this release forward, when specifying the data representation QoS for entities using the FlatData language binding, strict enforcement will be applied to require XCDR2. Any attempt to configure an entity with XCDR as the data representation for FlatData types will be blocked, with the entity creation call failing.

Starting in this release, if you use the Traditional C++ and Modern C++ APIs, you should examine your entity QoS configurations to ensure compliance with the new validation rules. Make adjustments to use XCDR2 where the FlatData language binding is in use, to align with the updated and more stringent requirements.

[RTI Issue ID CORE-14059]

6.24 Interoperability

6.24.1 [Minor] JRE version check prevented desktop Java tools from opening

A bug prevented desktop Java tools from opening if they detected certain versions of the Java Runtime Environment (JRE).

[RTI Issue ID INSTALL-898]

6.25 Other

6.25.1 [Critical] Potential bus error when calling print and to_string APIs in TypeCode *

In 7.1.0, your application may have generated a bus error after calling the **print** and **to_string** APIs, such as `DDS_TypeCode_print_IDL`, in TypeCode.

This issue only happened when TypeCode propagation was enabled by setting `participant_qos.resource_limits.type_code_max_serialized_length` to a value greater than zero (the default is zero) and when using the affected APIs in the TypeCodes that were part of `PublicationBuiltinTopicData` and `SubscriptionBuiltinTopicData`.

[RTI Issue ID CORE-13968]

6.25.2 [Critical] Restarted keyed DataReaders using durable reader state and destination order by source timestamp may have received old samples

A keyed *DataReader* may have received old samples after being restarted. This occurred when using durable reader state and destination order by source timestamp. In this scenario, the restarted *DataReader* received samples that should have been filtered out because their source timestamps were older than the last source timestamp received by the *DataReader* before it was restarted.

[RTI Issue ID CORE-14103]

6.25.3 [Critical] Reliable DataReader may have stopped receiving samples from DataWriter using durable writer history and DDS fragmentation

A reliable *DataReader* may have stopped receiving samples from a *DataWriter* using durable writer history and DDS fragmentation (asynchronous publishing with samples that exceed the minimum `mesage_size_max` across all installed transports). This issue occurred when a sample fragment was lost, which is more likely to occur in lossy networks.

[RTI Issue ID CORE-14099]

6.25.4 [Critical] Support for systems running beyond 2038 when using a database and logging

In release 7.2.0, *Connext* added support for systems running beyond the year 2038. See Support for systems running beyond 2038. However, 2038 support for *Connext* applications that use a database, like *Persistence Service*, or that use durable writer history, was not added at that time. Now, release 7.3.0 adds that 2038 support, too. Logging has also been updated to properly display the date up to year 2106.

[RTI Issue ID CORE-13866]

6.25.5 [Major] Durable writer history failed to restore data in buildable sources

There was a bug in Buildable Sources code that caused durable writer history to fail while restoring data.

[RTI Issue ID CORE-14255]

6.25.6 [Minor] Discovery plugins libraries did not close if creation of plugin failed

In previous releases, if your *Connext* application used any discovery plugin (like *Limited Bandwidth Endpoint Discovery* or *Limited Bandwidth Participant Discovery*) and the creation of the discovery plugin object failed (for example, due to an incorrect properties configuration), the loaded plugin's dynamic library was not properly closed, resulting in a memory leak.

Now if there is a failure, opened library handlers are closed.

[RTI Issue ID CORE-13410]

6.25.7 [Trivial] Strings with default size (255) may have printed as unbounded when printing TypeCodes as IDL

When printing a TypeCode in IDL format, strings and wstrings may have incorrectly been printed as unbounded if they had the default size (255).

For example, the following member:

```
string<255> my_string;
```

would have been printed as:

```
string my_string;
```

[RTI Issue ID CORE-14095]

6.25.8 [Trivial] Alias's annotations may not have printed out correctly

Annotations can be given to an alias in IDL or XML. When printing out this alias as IDL (using the `DDS_TypeCode_print` function), the annotations would not have been printed, unless the alias was being printed within a struct.

[RTI Issue ID CORE-14084]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

Chapter 7

Previous Releases

For “What’s New” in the Core Libraries in previous releases, see RTI Connext Core Libraries What’s New .

7.1 What’s New in 7.2.0

For what’s new in the Core Libraries in 7.2.0, see What’s New in 7.2.0 in the Previous Releases section of the *RTI Connext What’s New*.

7.2 What’s Fixed in 7.2.0

This section describes bugs fixed in *Connext* 7.2.0. These are fixes since 7.1.0.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

7.2.1 Discovery

[Critical] SPDP2 participants may not have completed discovery if IP mobility event occurred during discovery *

When using Simple Participant Discovery Protocol 2.0, discovery may not have completed between two *DomainParticipants* if one *DomainParticipant* changed locators due to an IP mobility event before its configuration message was received by the remote participant. You would have had to wait for *DomainParticipant* liveliness to expire at the `participant_liveliness_lease_duration` for discovery to be restarted. Now, the locator change is correctly propagated to the remote participant and participant discovery will complete.

[RTI Issue ID CORE-13384]

[Critical] Crash if initial_peers sequence contained a NULL string

Previously, if you configured the initial peers sequence through code, you could potentially add a NULL element. *Connext* did not check for the NULL element; therefore, when the *DomainParticipant* was created in this case, *Connext* crashed. Now a NULL element will be reported, resulting in an 'inconsistent qos' failure when creating the *DomainParticipant*.

[RTI Issue ID CORE-13802]

[Critical] Unbounded memory growth when creating/deleting DomainParticipants *

In *Connext* 7.1.0, a *DomainParticipant* was not freeing some of the memory associated with a remote *DomainParticipant* that was deleted. This may have led to unbounded memory growth if your applications continuosly create/delete *DomainParticipants*. When this growth occurred, you may have seen the following error message:

```
ERROR [DELETE DP|LC:DISC]COMMENDAnonWriterService_as-
sertRemoteReader:DELETION FAILURE | skiplist node already removed
```

This problem has been fixed.

[RTI Issue ID CORE-13964]

[Major] Failure to deserialize participant discovery information incorrectly allowed discovery to complete

It was possible for participant discovery to "succeed" even if the deserialization of the participant discovery information failed. In those cases, this error was printed:

```
PRESCstReaderCollator_storeSampleData:deserialize sample error in topic
'DISCParticipant' with type 'DISCParticipantParameter'
```

This incorrect 'success' could have led to unexpected behavior or crashes. This problem has been fixed. Now participant discovery won't complete if deserialization issues are detected.

[RTI Issue ID CORE-12952]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.2 Serialization and Deserialization

[Critical] Unbounded memory growth when deserializing SPDP discovery sample

Potential unbounded memory growth occurred when some parameters appeared multiple times within a Simple Participant Discovery Protocol (SPDP) discovery sample. This problem has been fixed. See also *Some parameters cannot be received multiple times within same SPDP sample*.

[RTI Issue ID CORE-13594]

[Critical] Potential unexpected behavior or crash when deserializing SPDP discovery sample

Potential unexpected behavior or a crash could occur when deserializing some inconsistent or malformed parameters within a Simple Participant Discovery Protocol (SPDP) discovery sample. This problem has been fixed.

[RTI Issue ID CORE-13811]

[Trivial] Wrong error message when deserializing PropertyQos property value and exceeding property_string_max_length resource limit

If property_string_max_length was exceeded when deserializing the PropertyQos property value, the resulting error message was wrong (the value of the maximum size in particular). This problem has been fixed. Now the error message shows the correct information.

[RTI Issue ID CORE-13678]

7.2.3 Debuggability

[Major] DataWriter instance statistics were not updated in all cases

The instance statistics within the `DDS_DataWriterCacheStatus` were not correct if `dds.data_writer.history.source_timestamp_based_autopurge_instances_delay` on that *DataWriter* was also being used. This issue has been resolved.

[RTI Issue ID CORE-13278]

[Trivial] Instance State Consistency QoS was commented out when printed out as XML from code *

When the `instance_state_consistency_kind` in the RELIABILITY QoS policy was printed as XML from code (for example, while calling `DDS_DataWriterQos_to_string_w_params()` in the C API), it was commented out. It is printed out now without the XML `<!--` and `-->` strings.

[RTI Issue ID CORE-13909]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.4 Transports

[Critical] Ungracefully terminated QNX processes using SHMEM transport prevented startup of new processes due to unclosed POSIX semaphores

If a QNX application using the shared-memory transport was ungracefully shut down, crashed, or otherwise had an abnormal termination while holding a POSIX semaphore used by the transport (for example, while sending data through the shared-memory transport), *Connext* applications launched after that point on the same domain may have waited forever for that semaphore to be released.

This problem has been resolved for QNX 7.1 and higher only. However, the fix makes communication with applications from a previous *Connext* version impossible when using the shared-memory transport. If you try to use shared memory with old applications, you will see the following error message(s):

```
incompatible shared memory protocol detected.
Current version 5.0 not compatible with x.y.

OR

incompatible shared memory protocol detected.
Current version x.y not compatible with 5.0.
```

There is no way to be backwards-compatible. You will have to use other transports such as UDPv4.

[RTI Issue ID CORE-9434]

[Critical] Stalled communication when using shared-memory transport

On systems with a weak memory architecture, such as Arm®, the shared-memory transport may have been corrupted due to a data race in the concurrent queue where the messages are written into the shared-memory segment. This data race may have occurred until `received_message_count_max` messages were sent through the transport. The corrupted transport resulted in parsing errors, which filled up the shared-memory segment. For example, you may have seen messages such as the following:

```
MIGInterpreter_parse:available space 24 < 28
MIGInterpreter_parse:!RTPS
MIGInterpreter_parse:INVALID from 0X1014D5A,0X7E8C7D92
NDDS_Transport_Shmem_send:failed to add data. shmem queue for port 0x1d3e is␣
↪full (received_message_count_max=2880, receive_buffer_size=100971520). Try␣
↪to increase queue resource limits.
```

This problem has been resolved. Now the data race that led to this situation cannot occur.

[RTI Issue ID CORE-13846]

[Major] Connext started before Windows completed duplicate address detection on network interfaces

In some cases, such as the use of *Connext* in a Windows service, *Connext* would be started before Windows completed duplicate address detection on its network interfaces. This would result in the inability to use those interfaces in *Connext*.

Connext will now delay the usage of Windows network interfaces until duplicate address detection completes successfully (i.e., the DadState is IpDadStatePreferred).

[RTI Issue ID CORE-13425]

[Minor] QNX applications using shared-memory transport may have led to thread priority inversion issues

Running QNX applications using the *Connext* shared-memory transport may have led to thread priority inversion issues.

This problem has been resolved for QNX 7.1 and higher only. However, the fix makes communication with applications from a previous *Connext* version impossible when using the shared-memory transport. If you try to use shared memory with old applications, you will see the following error message(s):

```
incompatible shared memory protocol detected.
Current version 5.0 not compatible with x.y.

OR

incompatible shared memory protocol detected.
Current version x.y not compatible with 5.0.
```

There is no way to be backwards-compatible. You will have to use other transports such as UDPv4.

[RTI Issue ID CORE-13745]

[Minor] Overflow in default TransportMulticastMappingQosPolicy procedure

This release fixes an integer overflow in a function that maps a multicast IP address to *DataReaders*. You may now see a different IP address being assigned to a *DataReader* when the TRANSPORT_MULTICAST_MAPPING QoS policy is set and the default `DDS_TransportMulticastMappingFunction_t` is used.

[RTI Issue ID CORE-13653]

7.2.5 Reliability Protocol and Wire Representation

[Critical] Samples lost by reliable reader acknowledging samples it did not receive after remote writer update

A reliable *DataReader* may have lost samples by incorrectly acknowledging samples it did not receive. This could occur after a remote *DataWriter* update, such as if the writer had an IP mobility event or updated its QoS policy. When the reader processed this event, it began sending a periodic ACK/NACK at the `nack_period` to the writer until it received another message from the writer. This ACK/NACK acknowledged samples up to the last sequence number that it received from the writer, even if samples before that sequence number had not been received. When the writer received this ACK/NACK, it may have considered those samples to be fully acknowledged.

The reader could request the lost samples again, but if the reader was using **VOLATILE** durability, the remote writer would GAP for the samples and they would not be resent. If the reader was using **TRANSIENT_LOCAL** durability, the writer would resend the samples if they were still available, but if the writer had updated the send window beyond the samples being requested, the samples would not be resent and would be lost.

This issue has been resolved. If a reader receives a remote writer update from a writer that is still alive, it will not begin sending additional ACK/NACKs at the `nack_period` to the writer. This prevents the reader from incorrectly acknowledging samples it did not receive. If a reader receives a remote writer update from a writer that is not alive, it will send additional ACK/NACKs at the `nack_period` to the writer, but the bitmap will accurately represent the missing samples rather than acknowledging the last received sample. Samples are no longer lost because they are not incorrectly acknowledged.

[RTI Issue ID CORE-13611]

[Critical] Sample loss when using asynchronous publisher due to missing GAP

Samples may have been lost when using an asynchronous publisher in the following scenario:

1. A reader sent a NACK to the writer requesting missing samples where the first `m` (where `m >= 0`) samples should have been sent to the reader and at least the last `n` (where `n >= 2`) samples were not for the reader (for example, were filtered with a content filter).
2. Some (but not all) of the `n` samples were no longer present in the writer queue (for example, were removed due to exceeding the `writer_qos.history.depth`).
3. The next sample after the NACK bitmap sent by the reader was also not for the reader.

In this scenario, the writer may have failed to send a GAP to the reader to inform the reader about the samples that were not for the reader. The reader may then have continued to NACK for these samples and failed to progress, leading to sample loss.

[RTI Issue ID CORE-13844]

[Major] Inconsistent RTPS protocol versions broadcasted by Connext

Previously, *Connext* broadcasted different RTPS protocol versions in different messages. The versions are fixed and unified in this release.

[RTI Issue ID CORE-13676]

7.2.6 Content Filters and Query Conditions

[Critical] Instance handling on a DataReader and filtering operations in ContentFilteredTopics, QueryCondition, TopicQueries, and Multi-Channel DataWriters may have failed

Starting in 6.0.0, you may have experienced invalid results in filtering operations when using ContentFilteredTopics, QueryCondition, TopicQueries, or Multi-Channel *DataWriters*. This issue may have resulted in *DataReaders* not receiving samples they should have. The following error message occurred: `DDS_SqlFilter_evaluateOnSerialized:deserialization error: sample`. This issue may also have caused failures on a *DataReader* when setting `writer_qos.protocol.disable_inline_keyhash` to TRUE on a matching *DataWriter*. This could have led to incorrect instance handling, where two different instances were considered the same.

This problem was specific to *Topic* types containing optional members, and occurred when the *DataReaders* and *DataWriters* on the *Topic* used XCDRv1 encapsulation. The problem affected all languages.

This problem has been resolved.

[RTI Issue ID CORE-13829]

7.2.7 Dynamic Data

[Major] Problems with int8/uint8 support

Previous releases of *Connext* had problems supporting int8/uint8. There were issues serializing/deserializing the type and getting/setting the values with DynamicData.

Support for int8/int8 has been improved. Generated code will now send and receive the data correctly in all languages. The only pending issue (not yet fixed in this release) is int8/uint8 collection in Python (RTI Issue ID CODEGENII-1912). This release also adds a method to DynamicData to set and get the data with the correct type and sign. This release provides a Java method to access unsigned integers.

This fix does not change the Type Kind on the wire. Features and products, such as *Admin Console*, that rely on the Type Kind for the data will not be able to detect the type correctly.

[RTI Issue ID CORE-8865]

7.2.8 Performance and Scalability

[Major] Performance issues when using FlatData with payload encryption or compression

You may have seen performance issues when using the FlatData language binding along with compression or payload encryption. In this case, the number of copies of each sample was not reduced to two, as is expected when using FlatData. (See "34.1.4 FlatData Language Binding" in the *Core Libraries User's Manual*.) This issue removed the performance improvement that FlatData provides, but *only* when compression or payload encryption was enabled. This problem did not occur when using FlatData without compression or payload encryption. This problem has been fixed.

[RTI Issue ID CORE-11262]

[Major] Transport utilization metrics overflowed in applications with high throughput *

Transport utilization periodic metrics (like `dds_participant_udpv4_usage_in_net_bytes_count` or `dds_participant_udpv6_usage_out_net_bytes_count`) could overflow in high-throughput applications (for example, applications that wrote and/or received large data with high frequency). If the polling period of *Monitoring Library 2.0* (previously called *Observability Library*) was big enough, the variation of the metrics in the period of time did not fit into a 32-bit integer.

If a metric overflowed, an error message like the following was produced:

```
ERROR RTI_Monitoring_getTransportUtilizationStatistics:TYPE CONVERSION␣
↪FAILURE | count (4421753352) exceeds max. representable UINT32 for metric␣
↪with metricGroupIndex 22
```

The metric was not propagated to *Observability Dashboards* in this case.

To mitigate this issue, transport utilization count metrics have been promoted to a 64-bit integer. Reducing the polling period also makes the overflow less likely.

[RTI Issue ID MONITOR-597]

[Minor] Performance degradation when using FlatData with ContentFilteredTopics

In previous releases, a *DataWriter* using FlatData and doing writer-side filtering for *DataReaders* using ContentFilteredTopics may have done more data copies than necessary, leading to suboptimal performance. This problem has been fixed.

[RTI Issue ID CORE-13250]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.9 APIs (C or Traditional C++)

[Critical] Some DDS_TypeCode operations may have crashed when invalid arguments were used

Some operations related to `DDS_TypeCode` did not properly check for `NULL` arguments, which could have caused a crash. Checks are now in place to avoid this issue.

[RTI Issue ID CORE-13681]

[Critical] Several C API DDS_GUID functions did not account for NULL parameters correctly

Multiple `DDS_GUID` functions from the C API such as `DDS_GUID_copy` did not account for NULL as their input parameters. Both the documentation and the implementation for these functions should now reflect the correct behavior.

[RTI Issue ID CORE-13483]

7.2.10 APIs (Modern C++ API)

[Critical] Unexpected rti.connextdds.PreconditionNotMetError when setting optional string members in QoS policies

Attempting to assign a non-set value to an optional string member in a QoS policy in modern C++ resulted in the generation of an `rti.connextdds.PreconditionNotMetError`.

The QoS policy members affected by this issue were:

```
EntityName::name
EntityName::role_name
Monitoring::application_name
MonitoringPeriodicDistributionSettings::datawriter_qos_profile_name
MonitoringEventDistributionSettings::datawriter_qos_profile_name
MonitoringLoggingDistributionSettings::datawriter_qos_profile_name
MonitoringDedicatedParticipantSettings::participant_qos_profile_name
MonitoringDistributionSettings::publisher_qos_profile_name
```

This problem has been resolved.

[RTI Issue ID CORE-13801]

[Critical] Move constructors for some of the built-in topic-types were incorrectly implemented

This issue was fixed in release 6.1.0, but not documented at that time.

The implementation of the move constructor and move assignment for the built-in topic types, such as PublicationBuiltinTopicType, may have caused undefined behavior. This problem has been resolved.

[RTI Issue ID CORE-13791]

[Critical] Manually closing some built-in readers could lead to a crash

Calling `close()` on the built-in *DataReaders* with topic names `service_request_topic_name` or `virtual_subscription_topic_name` could have led to a crash. (Note that if they were not manually closed, which is not necessary, the issue did not happen.) This issue has been fixed.

[RTI Issue ID CORE-13757]

[Critical] Incorrect implementation of DynamicDataMemberInfo constructor and assignment may have led to undefined behavior

It was not safe to copy a `DynamicDataMemberInfo` object. Using its copy constructor or copy-assignment operator may have led to undefined behavior if the `DynamicData` object that created it had been destroyed before. This problem has been resolved by making `DynamicDataMemberInfo` a true value type. It now owns the memory instead of relying on the related DynamicData object to be alive.

[RTI Issue ID CORE-13753]

[Major] int8_t, uint64_t, int64_t not supported as primitive types in Dynamic Type API

The types int8_t, uint64_t, int64_t were not accepted as a valid type for the templates of `dds::core::xtypes::PrimitiveType`. Therefore, the following code did not compile with C++11:

```
my_struct_type.add_
↪member((dds::core::xtypes::Member(dds::core::xtypes::PrimitiveType<int8_t>
↪()));
my_struct_type.add_
↪member((dds::core::xtypes::Member(dds::core::xtypes::PrimitiveType<int64_t>
↪()));
my_struct_type.add_
↪member((dds::core::xtypes::Member(dds::core::xtypes::PrimitiveType<uin64_t>
↪()));
```

The issue with int64_t and uint64_t was fixed in release 7.0.0. The error with int8_t is fixed in this release, 7.2.0. Now, the above code will compile and work.

[RTI Issue ID CORE-13689]

[Major] Policy getter for rti::core::policy::Monitoring previously missing *

The policy getter for the `rti::core::policy::Monitoring` QoS was previously missing in Modern C++. The missing getter has now been added.

[RTI Issue ID MONITOR-552]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.11 APIs (Java)

[Critical] Possible memory leak in DynamicData copy constructor

In the Java API only, under certain conditions, copying a DynamicData object using the constructor that receives another DynamicData object may have leaked native heap memory. This problem has been fixed.

[RTI Issue ID CORE-13609]

[Major] Some ReliabilityQos methods did not consider the instance state consistency QoS *

The `copy_from` and `equals` methods, as well as the implementation of the hash code for objects of that class, were not complete; they were missing the `instance_state_consistency_kind` QoS. This problem has been remedied.

[RTI Issue ID CORE-13785]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.12 APIs (Python)

[Major] Access to collection elements in some DynamicData accessors was not zero-based

Given a DynamicData instance `sample` with a sequence or array field (`my_seq`), when accessed via a nested field expression, the indexes were 1-based, not 0-based as in the rest of the API accessors. For example, the following was incorrect because the first element was 1:

```
value = sample["my_seq[0].x"]
```

This problem has been resolved. Now, indexes are zero-based and the expression above is valid.

[RTI Issue ID PY-98]

7.2.13 APIs (Multiple Languages)

[Major] Looking up a DataReader using the wrong class in Modern C++ or Python did not raise clear exception *

In Modern C++, when using the `find_datareader_by_topic_name` or `find_datawriter_by_topic_name` functions and the wrong *DataReader* type, the function may have returned an invalid entity. Now, it will throw a `dds::core::InvalidArgumentError`. For example:

```
auto dr =
    rti::sub::find_datareader_by_topic_name<DataReader<Foo>>(
    dds::sub::builtin_subscriber(participant),
    dds::topic::publication_topic_name());
```

In Python, the following code now throws a `dds.InvalidArgumentError`:

```
dr = dds.DataReader.find_by_topic(
            participant.builtin_subscriber, dds.PublicationBuiltinTopicData.
↪topic_name)
```

since the right *DataReader* class for the built-in `PublicationBuiltinTopicData` reader is `dds.PublicationBuiltinTopicData.DataReader`, not `dds.DataReader`.

[RTI Issue ID CORE-13800]

[Minor] Alias type not obtainable using a QosProvider

Alias types were not obtainable using a QosProvider. This problem affected all language bindings that support a QosProvider. This problem has been fixed.

[RTI Issue ID CORE-13830]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.14 XML Configuration

[Major] Creating Topic-specific entities from a <qos_profile> using QoS profile inheritance and/or composition returned incorrect values

Topic-specific entities include *DataWriter*, *DataReader* and *Topic*. Their corresponding tags `<datareader_qos>`, `<datawriter_qos>` and `<topic_qos>` contain the `topic_filter` attribute that allows you to indicate which Topic name the XML values should be used for. The internal mechanism of the Core Libraries XML parser had a bug where incorrect values could be returned from a `<qos_profile>` when the following conditions were true:

1. The `<qos_profile>` used QoS Profile inheritance and/or composition, where the parent QoS Profiles contained any of the above Topic-specific entities.

2. The `<qos_profile>` did not contain the QoS tag for the *Topic*-specific entity being created by pointing to it: for example, in the C API, if you called `DDS_DomainParticipant_create_datawriter_with_profile()` on a `<qos_profile>` that did not contain a `<datawriter_qos>` tag.

This issue has been resolved.

[RTI Issue ID CORE-13438]

[Major] Using languageBinding attribute on union types in XML caused parsing error

When a union type that used the languageBinding attribute was created in XML, a parsing error would result. This issue has been fixed.

[RTI Issue ID CORE-13905]

[Minor] configuration_variables tag was not effective

The `<configuration_variables>` tag was visible and accepted by the *Connext* .xsd files, but it had no effect: the configured values were not used by the Core Libraries to set the value of XML-defined environment variables. This has been corrected. Now, the `<configuration_variables>` tag can be used to define default values for XML-defined environment variables, which will take effect if those environment variables are not set on the terminal.

[RTI Issue ID CORE-11871]

[Minor] Incorrect parsing of data_representation attribute in XML type definitions

The type attribute `data_representation` was not parsed correctly. This could result in a type requiring a different representation (XCDR1, XCDR2, or both) than defined by the XML for the type.

[RTI Issue ID CORE-13769]

7.2.15 Instances

[Major] Instance purging based on source timestamp did not work *

In 7.1.0, the source timestamp-based purge delay did not purge instances based on their source timestamp. Instead, it purged instances based on their sequence number. This problem has been resolved. Now, the **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay** property works as intended.

[RTI Issue ID CORE-13911]

[Minor] Instances transitioned due to instance state consistency did not respect propagate_dispose_of_unregistered_instances *

By default, *Connext* does not support transitions between NOT_ALIVE instance states; however, this can be configured on the *DataReader* by setting **propagate_dispose_of_unregistered_instances** and/or **propagate_unregister_of_disposed_instances** in the DATA_READER_PROTOCOL QoS policy. Instances that were transitioned due to instance state consistency (i.e., instances that transitioned upon recovering liveliness with a previously matched *DataWriter*) were not abiding by this configuration and may have transitioned from NOT_ALIVE_NO_WRITERS to NOT_ALIVE_DISPOSED even though **propagate_dispose_of_unregister_instances** was false. This issue has been resolved.

[RTI Issue ID CORE-13477]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.16 Crashes

[Critical] Race condition when using multiple threads to enable same DomainParticipant

Suppose you created a disabled *DomainParticipant*. If you used multiple threads to enable this *DomainParticipant*, then a race condition may have led to a segmentation fault in release libraries or a precondition error in debug libraries. The precondition error looked similar to this:

```
REDAWeakReference_getReferent:!precondition: !((reference) != ((void *)0) &&␣
↪(reference)->_manager != ((void *)0) && (reference)->_
↪referentEpochAtCreation != (0)) || tableWithStartedCursor==((void *)0)

RTINetioReceiver_removeEntryport:!goto WR NetioReceiver_Entryport
```

This problem has been fixed. Calling `enable()` on a *DomainParticipant* is now thread-safe with respect to other calls to `enable()` on the same *DomainParticipant*.

[RTI Issue ID CORE-13535]

[Critical] Possible crash gathering periodic metrics for a resource that was being added or deleted at the same time *

Due to concurrency issues in the thread that gathers the periodic metrics of the observable resources, an application might have crashed because the thread accessed invalid memory. The crash could occur in any of the following scenarios:

- When a resource was deleted (for example, you deleted a *DataWriter*) at the same time that *Monitoring Library 2.0* (formerly called *Observability Library*) was gathering the periodic metrics of that resource. The thread may have accessed already freed memory.
- When a resource was added (for example, you created a *DataWriter*), the thread could start gathering the periodic metrics of that resource before the resource was completely initialized. The thread may have accessed uninitialized memory.

Depending on the configured `polling_period` for periodic metrics and the frequency your application creates and deletes observable resources, the chances of the conditions explained above happening at the same time were unlikely.

These concurrency issues are now fixed. *Monitoring Library 2.0* will not gather periodic metrics for resources that are being deleted or added.

[RTI Issue ID MONITOR-533]

[Critical] Potential crash when configuring logging verbosity to NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL or higher

Connext Receive threads may have crashed as a result of a race condition during the Receive thread destruction process.

This problem, which was only possible when the *Connext* logging verbosity was set to `NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL` or higher (i.e., more verbose than STATUS_LOCAL), is now resolved: Receive threads no longer crash during their destruction.

[RTI Issue ID CORE-13649]

[Critical] Malloc called when handling SIGSEGV

Previously, when handling a segmentation violation signal (SIGSEGV), it was possible for malloc to be called while logging backtrace information. In certain scenarios, this could cause another segmentation violation, and this cycle of events would continue indefinitely. Now, malloc will not be called when handling segmentation violation signals.

[RTI Issue ID CORE-13396]

[Critical] Calling delete_contained_entities APIs could cause a crash in the thread that collects periodic metrics *

If your application used any of the `delete_contained_entities` APIs (e.g., `DDS_DomainParticipant_delete_contained_entities`) and *Monitoring Library 2.0* (previously called *Observability Library*) was still enabled, there was a possibility of a crash happening in the thread that collects periodic metrics. The crash happened because the children DDS Entities were removed before deleting their observable resources. Therefore, the periodic metrics thread could try to collect metrics for an observable resource whose DDS Entity no longer exists.

This issue is now fixed. The periodic metrics collector thread will not try to collect metrics for observable resources that are being deleted.

[RTI Issue ID MONITOR-549]

[Critical] Application could crash when disabling and re-enabling Monitoring Library 2.0 due to internal error *

If there was an error in an internal function of *Monitoring Library 2.0* (formerly known as *Observability Library*), depending on the memory state an application using the Library could crash in the following scenario:

1. *Monitoring Library 2.0* was enabled.
2. You created some DDS Entities (*DomainParticipant*, *Publisher*, *DataReader* …) in your application.
3. You disabled and re-enabled the Library. Due to an internal error, an exception was printed in the `RTI_Monitoring_collectDdsResources` function.
4. You deleted any DDS Entity before disabling the Library.

Because of the error in `RTI_Monitoring_collectDdsResources`, the observable resources associated with the DDS Entities were not updated for the second activation of the Library. The DDS Entities kept the old observable resources object from the previous activation, which were no longer valid.

When deleting the DDS Entities, these old observable resources were used without checking their validity. The behavior was undefined at that point and, depending on the memory state, the application could crash.

This issue is now fixed. *Monitoring Library 2.0* no longer uses observable resources without checking their validity first.

[RTI Issue ID MONITOR-548]

[Critical] Low-memory conditions could lead to crash on several platforms if allocation of high resolution clock failed

If the system was running very low in memory, a failure to allocate the high-resolution clock could then lead to a crash, since a NULL pointer would have been dereferenced while attempting to handle the failure. This issue applied to all platforms except Windows, Solaris, and Integrity, where the issue would not have occurred. This problem has been fixed.

[RTI Issue ID CORE-13899]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.17 Entities

[Critical] Application may have hung when deleting a monitored DDS entity *

If *RTI Monitoring Library 2.0* (previously called *Observability Library*) was enabled, you deleted a DDS entity (for example, by calling `DDS_Publisher_delete_datawriter` or a similar API), and periodic metrics were being collected for the same DDS entity, the application may have hung. The hang occurred because the deletion thread and the periodic thread took the same pair of semaphores in inverted order.

This hang is now fixed. Periodic metrics are not collected for an entity that is being deleted.

[RTI Issue ID MONITOR-580]

[Major] Monitoring Library 2.0 incorrectly collected both enabled and disabled DDS Entities *

In the previous release, *Monitoring Library 2.0* (then called *Observability Library*) incorrectly collected both enabled and disabled DDS Entities if the library was enabled after creating the entities. Now, *Monitoring Library 2.0* will only assert enabled DDS Entities, ensuring that disabled entities are not unnecessarily collected. Disabled DDS Entities are asserted when they are enabled.

[RTI Issue ID MONITOR-594]

[Major] Monitoring Library 2.0 did not assert disabled DDS Entities when the Entities were enabled *

If *Monitoring Library 2.0* (previously called *Observability Library*) was enabled in an application and then DDS Entities were created disabled (by setting the `entity_factory.autoenable_created_entities` QoS setting to `false`), disabled Entities were not asserted by the library when they were enabled. This meant that these DDS Entities were never observed by *Monitoring Library 2.0*.

This issue is fixed. Disabled DDS Entities (and all their contained Entities) are now asserted after enabling.

[RTI Issue ID MONITOR-574]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.18 Interoperability

[Critical] Possible incomplete endpoint discovery when communicating with other DDS vendors

Connext only supports a maximum of four representations in the DATA_REPRESENTATION QoS policy for readers, and one representation for writers. However, other DDS vendors may support more than this. If a *Connext* endpoint was communicating with another vendor's endpoint with more than the supported representations, there may have been interoperability issues:

- Without Security: Builtin Topic Publication/Subscription listeners failed to call the associated callbacks for received discovery samples from other vendors announcing more than one data representation for writers, or more than four data representations for readers.
- With Security: If enabled, the SECURITY PLUGINS *(RTI Security Plugins)* failed to interoperate with other vendors announcing more than one data representation for writers, or more than four data representations for readers.

This problem no longer occurs. In the case of a *DataReader* with more than four representations, *Connext* now uses only the first four. In the case of a *DataWriter* with more than one representation, *Connext* now uses only the first.

[RTI Issue ID CORE-13836]

7.2.19 Vulnerabilities

[Critical] Out-of-bounds read while deserializing malformed partition parameters from malicious RTPS message *

An out-of-bounds read may have occurred while deserializing malformed partition parameters from a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Out-of-bounds read while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on confidentiality of *Connext* application.
- CVSS Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L

User Impact with Security

Same as "User Impact without Security," above.

[RTI Issue ID CORE-13669]

[Critical] Out-of-bounds read while deserializing malformed IPv6 locator from malicious RTPS message

An out-of-bounds read may have occurred while deserializing a malformed IPv6 locator from a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Out-of-bounds read while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on confidentiality of *Connext* application.
- CVSS Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L

User Impact with Security

Same as "User Impact without Security," above.

[RTI Issue ID CORE-13764]

[Critical] Remote modification of DomainParticipant names in unsecure system

In a system without security, a vulnerability in the *Connext* application could have potentially allowed remote attackers to modify the *DomainParticipant* name of any *DomainParticipant* in the system. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Any *DomainParticipant* could have its participant's name changed by an attacker.
- Remotely exploitable.
- Potential impact on integrity of *Connext* application.
- CVSS Base Score: 5.3 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

User Impact with Security

No impact when using the SECURITY PLUGINS if enabling `rtps_protection` or if `discovery_protection_kind` is different than NONE: in this case, participant discovery samples will be protected against tampering from an external malicious agent after authentication is completed. Moreover, non-legitimate changes in the participant discovery information before authentication are always prevented by the authentication process, which ensures that the participant discovery information is authentic.

[RTI Issue ID CORE-13817]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.2.20 Other

[Critical] Possible hang in application if something failed while adding a new observable resource *

An application might have hanged if something went wrong while adding a new observable resource (for example, you created a *DataWriter*). Before the hang, you would have seen an exception error in the `RTI_MonitoringResourceRegistry_assertResource` function. However, not all errors in this function led to the hang. This issue is now fixed.

[RTI Issue ID MONITOR-534]

[Critical] Application may have hung when event and event snapshot were published simultaneously for same observable resource *

When *RTI Observability Collector Service* discovers a *Connext* application, *RTI Monitoring Library 2.0* (previously called *Observability Library*) automatically sends a special sample named "event snapshot". This sample contains the current values of event metrics for each observable resource. If an event (for example, liveliness change) was triggered for an observable resource at the same time as an event snapshot was being published for the same resource, the application may have hung. The hang occurred because the thread that published the event and the thread that published the snapshot took the same pair of semaphores in inverted order.

This hang is fixed. Now, both threads take the semaphores in the same order.

[RTI Issue ID MONITOR-584]

[Critical] Unable to start Launcher, Admin Console, Code Generator, and Monitor in Windows when the RTI Workspace contained white spaces *

On Windows systems, *Launcher*, *Admin Console*, *Code Generator,* and *Monitor* failed to start when the RTI Workspace contained white spaces. This issue has been fixed.

[RTI Issue ID TELEMETRY-28]

[Critical] Deadlock issue resolved when disabling Monitoring Library 2.0 during command processing *

In the previous release, a deadlock could occur if *RTI Monitoring Library 2.0* (previously known as *RTI Observability Library*) was disabled while a remote administration command was being processed. The hang was caused because the thread that processed the command and the thread that disabled the Library took the same pair of semaphores in inverted order.

This issue has been addressed in this release. Disabling the Library while a remote administration command is being processed is now thread safe.

[RTI Issue ID MONITOR-609]

[Major] Native Android applications were not shipped

Native Android applications are now included in Android target bundles, along with the APKs.

[RTI Issue ID INSTALL-789]

[Major] References to missing header file in Connext Professional source bundle

The Connext Professional source bundle included references to a header file in the `xmlutils.1.0` module that is not part of the source bundle. As a result, if you were building *Connext* from source, you were unable to complete the build due to the missing header file. RTI has now removed this dependency from the `xmlutils.1.0` module.

[RTI Issue ID CORE-12846]

[Major] Access to an internal field of observable resources was not thread safe *

In *Monitoring Library 2.0* (previously known as *Observability Library*), if a remote administration command was issued for an observable resource (such as changing the Forwarding verbosity level of an application) at the same time that periodic metrics were collected for the same resource, an internal field of the resource was accessed by the two threads unsafely. The value of the internal field could remain in an inconsistent state, which, in the worst case, might have led to a deadlock when deleting the resource.

This issue is fixed. Accesses to the internal field are now thread safe.

[RTI Issue ID MONITOR-575]

[Minor] Running rtisetenv_<arch>.bat caused issues in PATH environment *

In release 7.1.0, running `rtisetenv_<arch>.bat` may have caused issues in the PATH environment on Windows. This problem has been fixed.

[RTI Issue ID INSTALL-880]

[Minor] Error creating a DataWriter using durable writer history if setting property dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size to -1

Creating a *DataWriter* using durable writer history and setting the property `dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size` to -1 may have failed with the following error:

```
!allocate sample buffer pool
```

Even if the *DataWriter* creation did not fail, the value of `dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size` would be incorrectly applied. The value was set to `dds.data_writer.history.odbc_plugin.builtin.instance_cache_max_size` for keyed topics and 1 for unkeyed topics.

This problem has been resolved.

[RTI Issue ID CORE-13732]

[Trivial] Connext did not print array dimensions for aliases that were arrays

When printing the type information for a type that is an alias of an array type, the array dimensions are now output for both the IDL and XML representations.

[RTI Issue ID CORE-13651]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.3 What's New in 7.1.0

For what's new in the Core Libraries in 7.1.0, see What's New in 7.1.0 in the Previous Releases section of the *RTI Connext What's New*.

7.4 What's Fixed in 7.1.0

This section describes bugs fixed in *Connext* 7.1.0. These are fixes since 7.0.0.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

7.4.1 Fixes Related to Discovery

[Critical] Unbounded memory growth when using domain tags or DomainParticipant partitions

Whenever a *DomainParticipant* discovered another *DomainParticipant* that it did not match with, either due to a mismatched domain tag or participant partition, some state was kept that was never removed if the *DomainParticipant* never received an announcement from that same mismatched participant indicating that it had been shut down. This led to unbounded memory growth, which could become an issue in systems where *DomainParticipants* with various different domain tags or partitions were coming and going.

[RTI Issue ID CORE-12973]

[Critical] Most up-to-date participant configuration may not have been received by other participants and may have led to discovery not completing

It was possible that a configuration change in *DomainParticipant* 'A' may not have been received by *DomainParticipant* 'B' if the change occurred while the two participants were discovering each other. Examples of configuration changes are a change in the PROPERTY QoS policy or an IP mobility event in which *DomainParticipant* 'A' changes one of its IP addresses.

Not having the most recent configuration may have led to discovery not happening if the change was due to an IP mobility event.

The problem only occurred when discovery used multiple transports (e.g, SHMEM and UDPv4). This problem has been fixed.

[RTI Issue ID CORE-13359]

[Major] Error deleting remote endpoints with specific GUID prefixes using debug libraries

An error occurred when using debug libraries in the unlikely case that a *DomainParticipant* had a zero value as the hostId, appId, or instanceId. This problem has been fixed.

[RTI Issue ID CORE-13261]

[Major] Participant failed to assert remote participant if usability of shared memory transport changed *

In 7.0.0, a DomainParticipant failed to assert a remote DomainParticipant if the usability of the shared memory transport changed, resulting in the following log message:

```
ERROR [0x010114FE,0x12488672,0x8EE3B6BC:0x000100C7{Entity=DR,MessageKind=DATA}
↪|RECEIVE FROM 0x00000000,0x00000000,0x00000000:0x000100C2|:0x000001C1
↪{Domain=0}|ASSERT REMOTE DP|LC:DISC]PRESParticipant_
↪assertConfiguredRemoteParticipant:ASSERT FAILURE | compare immutable remote␣
↪participant 0x01017851,0x3B428DDD,0x514330AA config RW
ERROR [0x010114FE,0x12488672,0x8EE3B6BC:0x000100C7{Entity=DR,MessageKind=DATA}
↪|RECEIVE FROM 0x00000000,0x00000000,
↪0x00000000:0x000100C2|LC:DISC]DISCParticipantDiscoveryPlugin_
↪assertRemoteParticipantConfig:!assert remote participant: 0x01017851,
↪0x3B428DDD,0x514330AA,0x000001C1
ERROR [0x010114FE,0x12488672,0x8EE3B6BC:0x000100C7{Entity=DR,MessageKind=DATA}
↪|RECEIVE FROM 0x00000000,0x00000000,
↪0x00000000:0x000100C2|LC:DISC]DISCParticipantDiscoveryPlugin_
↪assertRemoteParticipantFull:ASSERT FAILURE | remote participant 0x01017851,
↪0x3B428DDD,0x514330AA config information
ERROR [0x010114FE,0x12488672,0x8EE3B6BC:0x000100C7{Entity=DR,MessageKind=DATA}
↪|RECEIVE FROM 0x00000000,0x00000000,
↪0x00000000:0x000100C2|LC:DISC]PRESParticipantAnnouncementChannelReaderListenerSpdp_
↪onDataAvailable:!assert remote participant
```

You may have run into this issue if a shared memory segment was deleted during runtime and a DomainParticipant updated its configuration information. A change in the shared memory usability will no longer cause this failure.

[RTI Issue ID CORE-13360]

[Major] Unexpected warning during discovery when multicast disabled

Connext logged a warning during the discovery process when multicast was disabled. The message warned about unreachable multicast locators. The message was unexpected and has been removed.

[RTI Issue ID CORE-13403]

[Minor] Potential memory leak when creation of any of the built-in discovery plugins failed

The first time a *DomainParticipant* is created in an application, some memory is allocated globally for each of the built-in discovery plugins (SPDP and SEDP) enabled for that *DomainParticipant*. This global memory is released when finalizing the *DomainParticipantFactory* instance.

However, if there was a failure in the creation of any of the builtin discovery plugins during the *DomainParticipant* creation, the *DomainParticipantFactory* was not notified properly that this global memory was allocated. Therefore, finalizing the *DomainParticipantFactory* instance did not release the memory, causing a leak.

This problem is fixed. Finalizing the *DomainParticipantFactory* instance always releases the memory if it was previously allocated, regardless of whether or not a failure occurred.

[RTI Issue ID CORE-12882]

[Minor] Unexpected, invalid locator propagated within builtin topics

A *DataReader* could unexpectedly propagate an invalid locator to a *DataWriter* for certain builtin topics. The issue did not affect functionality, since the locator was discarded on the *DataWriter* side. The bug that sent the invalid locator has been fixed.

[RTI Issue ID CORE-13416]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.2 Fixes Related to Serialization and Deserialization

[Critical] Unexpected union value when receiving a discriminator that does not select any union member on DataReader's type

When the property **dds.sample_assignability.accept_unknown_union_discriminator** was set to 1, previous *Connext* releases were not always compliant with the latest OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3 when a *DataWriter* publishes a union sample with a discriminator value that selects a union member, and a *DataReader* subscribes to a union type that does not have a union member for the discriminator published by the *DataWriter*.

For example:

```
/* Publisher */
union MyUnion switch(int32) {
    case 0:
        int32 m1;
    case 1:
        int16 m2;
    case 2:
        double m3;
};

/* Subscriber */
union MyUnion switch(int32) {
    case 0:
        int32 m1;
    case 1:
        int16 m2;
};
```

In this example, if the *DataWriter* published a sample with a discriminator value set to 2 selecting m3, the *DataReader* received a sample where the discriminator is set to 0 and m1 is set to 0, the default value of the union. According to the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, the *DataReader* should preserve the discriminator value received from the *DataWriter* even if this discriminator value does not select any member in the *DataReader's* union.

This problem only occurred when one of these conditions was true:

- The unions are mutable regardless of the data encapsulation (XCDR1 or XCDR2).
- The unions are appendable, and the encapsulation is XCDR2.

Note if the union discriminator did not select any member on the *DataWriter's* type, such as 3 in the above example, the *DataReader* received the expected discriminator 3.

This release accepts a new value for the **dds.sample_assignability.accept_unknown_union_discriminator** property:

- 0 (existing value and default value): Received samples containing a union discriminator value that selects a union member on the *DataWriter* but not on the *DataReader* are dropped.
- 1 (existing value) : Received samples containing a union discriminator value that selects a union member on the *DataWriter* but not on the *DataReader* are set to the default union value.
- 2 (new value): Received samples containing a union discriminator value that selects a union member on the *DataWriter* but not on the *DataReader* preserve the discriminator value.

Received samples containing a union discriminator value that does not select a union member on the *DataWriter* always preserve the discriminator value on the *DataReader* with **dds.sample_assignability.accept_unknown_union_discriminator** set to 1 or 2, unless the union discriminator value is an enumerator which is not valid on the *DataReader's* type. In this case, the union is set to its default value.

To be compliant with the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, set the value to 2.

[RTI Issue ID CORE-13058]

[Critical] Serialization of samples failed or produced a segmentation fault for types with max serialized size larger than 2GB

A *DataWriter* may have failed to send a sample due to serialization errors when the sample's type had a max serialized size with a value larger than 2GB.

For example:

```
@nested
    struct MyNestedStruct2 {
        sequence<octet, 1500000000> m1;
    };

    @nested
    struct MyNestedStruct {
        sequence<octet, 1000000000> m1;
        MyNestedStruct2 m2;
    };

    struct MyStruct {
        MyNestedStruct m1;
    };
```

In this example, the serialize operation failed with an error like this:

```
[0x0101C50B,0x0D4E0B41,0xBBFA04AC:0x80000003{E=DW,T=Example MyStruct,
↪C=MyStruct,D=56}|WRITE] PRESWriterHistoryDriver_serializeSample:serialize␣
↪sample error in topic 'Example MyStruct' with type 'MyStruct' and␣
↪encapsulationId 1
```

For 32-bit platforms, the application may have produced a segmentation fault instead of failing to serialize.

This problem has been fixed.

[RTI Issue ID CORE-12687]

[Critical] Potential sample corruption when deserializing a malformed RTPS message

A sample could be corrupted/incomplete with no error logged in the case of a deserialization failure in the transport info parameter of the RTPS message. This problem has been fixed.

[RTI Issue ID CORE-13366]

[Critical] Unbounded memory growth when deserializing a malformed RTPS message

Potential unbounded memory growth occurred while parsing a malicious RTPS message. This problem has been fixed.

[RTI Issue ID CORE-13397]

7.4.3 Fixes Related to Debuggability

[Critical] Hang/crash when invoking a DataReader/DataWriter discovery snapshot within a callback function *

A hang or even a crash occurred when trying to get a discovery snapshot from a *DataReader* or *DataWriter* within a callback. RTI strongly recommends avoiding calling discovery snapshot APIs in callback functions in release 7.0.0. This issue has been fixed in 7.1.0.

[RTI Issue ID CORE-12959]

[Major] Unexpected fatal error when number of instances reached the limit *

In 7.0.0, an unexpected fatal error could be logged when the following occurred:

- A *DataWriter* is configured to use durable writer history.
- The number of instances reached the **max_instances** limit set in the *DataWriter's* RESOURCE_LIMITS QoS.
- *Connext* could not find an instance to delete (such as an unregistered one), to replace with the new instance. So the new instance could not be added.

This log message is expected, but it is not a fatal error, so its verbosity has been updated to WARNING, as follows:

```
WriterHistoryOdbcPlugin_createResources:FIND FAILURE | Instance for␣
↪replacement
WriterHistoryOdbcPlugin_addInstance:OUT OF RESOURCES | Exceeded the number of␣
↪instances. Current registered instances (128), maximum number of instances␣
↪(128)(writer_qos.resource_limits.max_instances)
```

[RTI Issue ID CORE-13496]

[Trivial] Memory leak if network capture initialization failed

Failure to initialize network capture for a *DomainParticipant* may have caused a memory leak of 746 kB. The leak only happened (upon *DomainParticipant* creation) if the initialization failed when creating the status mutex for a manager:

```
!create status mutex for the network capture manager
```

This issue is now fixed. A failure creating the status mutex for a manager does not leak memory anymore.

[RTI Issue ID CORE-13018]

[Trivial] Unexpected log messages at warning verbosity

You may have seen the following unexpected log messages at the warning verbosity level:

```
!get xxx remoteWriter
!get xxx remoteReader
!goto WR xxx remote reader
!goto WR xxx remote writer
```

These warnings did not signal any unexpected scenario, and they have been removed.

[RTI Issue ID CORE-13434]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.4 Fixes Related to Transports

[Critical] Possible data loss after a Connext application lost its multicast interfaces or gained its first multicast interface

The IP mobility feature detects when the interfaces of an application change, then propagates these changes. If an IP mobility event causes either the loss of the last interface that supported multicast or the gain of the first interface that supports multicast, the way other applications communicate with the application that experienced the IP mobility event changes.

Previously, that transition did not happen properly and may have led to data losses. This problem has been fixed. Now, communication is not affected by these interface changes.

[RTI Issue ID CORE-12609]

[Major] DomainParticipant with non-default metatraffic_transport_priority QoS did not complete discovery

A *DomainParticipant* that had a non-default **metatraffic_transport_priority** in the DISCOVERY QoS Policy was not able to complete endpoint discovery due to a unicast metatraffic channel that was not created correctly. (The channel is used by the participant to send Data(R) and Data(W).)

This issue was introduced in 6.1.0. This issue has been resolved.

[RTI Issue ID CORE-12739]

[Major] TCP Transport did not run with Windows debug libraries when socket_monitoring_kind was set to IOCP *

An internal error prevented the TCP transport from running on Windows with debug libraries when the **socket_monitoring_kind** was set to the recommended value of NDDS_TRANSPORT_TCPV4_SOCKET_MONITORING_KIND_WINDOWS_IOCP. The error has been corrected.

[RTI Issue ID COREPLG-654]

[Minor] dds.transport.minimum_compatibility_version property did not properly adjust locator format

Connext 5.3.0 introduced a new shared memory locator format. *DomainParticipants* in *Connext* 5.3.0 (and above) use the new locator format by default. To allow interoperability with *Connext* versions before 5.3.0, you must indicate to *DomainParticipants* to use the old locator format.

There are two properties for telling a *DomainParticipant* to use the old locator format: **dds.transport.use_530_shmem_locator_matching** (undocumented and deprecated) and **dds.transport.minimum_compatibility_version**. The latter is a newer property that combines several other properties. Its purpose is to set the transport to be compatible with the specified version in a simplified manner.

The problem with the newer property, **dds.transport.minimum_compatibility_version**, was that it did not adjust the locator format depending on the *Connext* version. The workaround was to use the **dds.transport.use_530_shmem_locator_matching** property instead. This issue has been fixed. You can now use **dds.transport.minimum_compatibility_version** without issue.

[RTI Issue ID CORE-12789]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.5 Fixes Related to Reliability Protocol and Wire Representation

[Critical] Samples not delivered to Required Subscription DataReaders when DataWriter used durable writer history and DataReaders disabled positive ACKs

A sample may not have been delivered to a Required Subscription *DataReader* if the *DataWriter* was using durable writer history and there were matching *DataReaders* configured with **reader_qos.protocol.disable_positive_acks**. This behavior violated the required subscription contract. This problem has been resolved.

[RTI Issue ID CORE-12825]

[Critical] DataReader may not have received samples that were sent as gapped samples to another DataReader over multicast

A *DataReader* may not have received samples that were sent as gapped samples to another *DataReader* over multicast. A GAP tells a *DataReader* that it should not expect to receive the samples that are listed in the GAP message. In some cases, when a *DataWriter* was responding to a *DataReader's* NACK message, the response contained a GAP which identified samples that should not have been gapped for any other *DataReader* aside from the *DataReader* whose NACK was being responded to. This was a problem if the NACK response was sent over multicast and was received by other *DataReaders*, because those *DataReaders* would incorrectly assume those gapped samples were irrelevant and would never receive them.

This issue has been resolved.

[RTI Issue ID CORE-13104]

[Critical] Unexpected precondition error with debug libraries on a reliable DataWriter while sending a GAP

In the 6.1.2 and 7.0.0 releases, you may have seen the following precondition error while using the *Connext* debug libraries.

```
DL Debug: :    Backtrace:
141: DL Debug: :     #4  COMMENDSrWriterService_sendGapToRR /rti/jenkins/
↪workspace/connextdds_ci_fastbuild-debug_develop/commend.1.0/srcC/srw/
↪SrWriterService.c:4096 (discriminator 9) [0x5B101E]
141: DL Debug: :     #5  COMMENDSrWriterService_onSendDataEvent /rti/jenkins/
↪workspace/connextdds_ci_fastbuild-debug_develop/commend.1.0/srcC/srw/
↪SrWriterService.c:6570 [0x5BACF6]
141: DL Debug: :     #6  RTIEventActiveGeneratorThread_loop /rti/jenkins/
↪workspace/connextdds_ci_fastbuild-debug_develop/event.1.0/srcC/
↪activeGenerator/ActiveGenerator.c:307 [0x28E2FC]
141: DL Debug: :     #7  RTIOsapiThreadFactory_onSpawned /rti/jenkins/
↪workspace/connextdds_ci_fastbuild-debug_develop/osapi.1.0/srcC/
↪threadFactory/ThreadFactory.c:208 [0x1F3A42]
141: DL Debug: :     #8  RTIOsapiThreadFactory_onSpawned /rti/jenkins/
↪workspace/connextdds_ci_fastbuild-debug_develop/osapi.1.0/srcC/
↪threadFactory/ThreadFactory.c:208 [0x1F3A42]
141: DL Debug: :     #9  RTIOsapiThreadChild_onSpawned /rti/jenkins/workspace/
```

```
↪connextdds_ci_fastbuild-debug_develop/osapi.1.0/srcC/thread/Thread.c:1941␣
↪[0x1EDB64]
141: DL Debug: :      #10 start_thread /build/glibc-CVJwZb/glibc-2.27/nptl/
↪pthread_create.c:463 [0x76DB]
141: DL Debug: :      #11 clone /build/glibc-CVJwZb/glibc-2.27/misc/../sysdeps/
↪unix/sysv/linux/x86_64/clone.S:97 [0x12161F]
141: DL Fatal: : FATAL rCoRTInk####Evt [0x01014F91,0x39810444,
↪0x4EC68AEA:0x000004C2|RECEIVE FROM remote DR (GUID: 0x01015FBD,0x5892DC7E,
↪0x9DB082D4:0x000004C7).
141: ] Mx00:/rti/jenkins/workspace/connextdds_ci_fastbuild-debug_develop/
↪commend.1.0/srcC/srw/SrWriterService.c:4099:RTI0x200003b:!precondition:
↪"((((gapStartSn)->high) > (((&(gapBitmap)->_lead))->high)) ? 1 :␣
↪((((gapStartSn)->high) < (((&(gapBitmap)->_lead))->high)) ? -1 :␣
↪((((gapStartSn)->low) > (((&(gapBitmap)->_lead))->low)) ? 1 :␣
↪((((gapStartSn)->low) < (((&(gapBitmap)->_lead))->low)) ? -1 : 0)))) >= 0"
141: DL Error: : ERROR [0x01014F91,0x39810444,0x4EC68AEA:0x000004C2|RECEIVE␣
↪FROM remote DR (GUID: 0x01015FBD,0x5892DC7E,0x9DB082D4:0x000004C7).
141: ] COMMENDSrWriterService_onSendDataEvent:!send GAP
```

This error was generated by a reliable *DataWriter* sending a GAP to a reliable *DataReader*. After the error was printed, the *DataReader* may have stopped receiving data from the *DataWriter*, leading to a non-recoverable situation. This problem did not occur with release libraries. This problem has been fixed.

[RTI Issue ID CORE-13462]

[Minor] DDS fragmentation may have led to more fragments than expected for a sample *

In 7.0.0, you may have noticed that when using middleware-level fragmentation and a flow controller where **bytes_per_token** is set to a value smaller than the minimum transport **message_size_max** across all installed transports, the number of sample fragments generated for a sample may have been bigger than expected. Although this was not a functional issue, it may have led to performance degradation.

This problem has been fixed.

[RTI Issue ID CORE-13190]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.6 Fixes Related to Content Filters and Query Conditions

[Critical] Unexpected "RTIXCdrSampleInterpreter_initializeSampleWInstruction" error log messages when using QueryConditions, ContentFilteredTopics, TopicQueries, or Multi-Channel

In releases 6.0.x and 6.1.x, a *Connext* application using QueryConditions, ContentFilteredTopics, TopicQueries, or Multi-Channel may have logged an error message like the following when applying filtering to some samples:

```
RTIXCdrSampleInterpreter_initializeSampleWInstruction: <Type>:<Field Name>␣
↪initialize error
```

A potential workaround was to set the property **dds.content_filter.sql.deserialized_sample.min_buffer_size** to -1 in the participant_qos.**property** QoS Policy. However, this may have led to a higher memory utilization.

This problem has been resolved.

[RTI Issue ID CORE-13328]

7.4.7 Fixes Related to Dynamic Data

[Major] DynamicData DataWriters incorrectly serialized optional empty sequences as null

In previous 6.0.0 releases and above, a DynamicData *DataWriter* incorrectly serialized an optional empty sequence as null. When a *DataReader* received the sample, it deserialized the wrong value.

For example, assume the following type:

```
struct AuditLogEntry {
    long long Nanoseconds;
    @optional sequence<long long, 100> Details;
};
```

If the publishing application set Details to an empty sequence with zero elements, the serialized value was incorrectly set to null. When a *DataReader* received the sample, it incorrectly set Details to null instead of the empty sequence with zero elements.

This problem has been fixed.

[RTI Issue ID CORE-12866]

7.4.8 Fixes Related to APIs

[Minor] DynamicData method to get member type missing in Modern C++ and C# APIs

The method to retrieve a member type from a DynamicData object was not provided in the Modern C++ and C# APIs. The following methods have now been added:

- C++: DynamicData::member_type(const std::string& name) and member_type(uint32_t id)
- C#: DynamicData.GetMemberType(string name) and GetMemberType(int id)

[RTI Issue ID CORE-13371]

Fixes Related to Modern C++ API

[Major] banish and subject_name APIs were unresolved in Modern C++ Windows dynamic libraries *

The Modern C++ APIs **banish_ignored_participants**, **discovered_participant_subject_name**, and **discovered_participants_from_subject_name** were unresolved symbols in the nddscpp2 Windows dynamic libraries. If you attempted to use them, you would get LNK2019 unresolved external symbol errors. This problem has been fixed.

[RTI Issue ID CORE-13053]

[Major] Unnecessary small memory allocation in some operations, including read/take

Every call to a *DataReader* read/take operation caused an unnecessary small memory allocation that was immediately released. More generally, initializing a reference type to **dds::core::null** caused the same allocation. For example:

```
DomainParticipant p = dds::core::null;
```

This unnecessary allocation has been removed. Constructing a reference type to **dds::core::null** no longer allocates memory.

[RTI Issue ID CORE-13262]

[Major] close() operation of a ContentFilteredTopic created from XML didn't work

The **close()** operation of a ContentFilteredTopic created from XML didn't actually close it. However, when its *DomainParticipant* was closed or destroyed, the ContentFilteredTopic was correctly closed. This problem has been resolved.

[RTI Issue ID CORE-13367]

Fixes Related to C# API

[Critical] Exception when disposing a DomainParticipant or when entities were not properly disposed

In previous releases of the .NET API, an exception may have occurred when disposing a *DomainParticipant* or whenever unused entities that had not been properly disposed were garbage-collected.

[RTI Issue ID CORE-13231]

[Major] Windows library dependency missing from .NET API NuGet packages *

In release 7.0.0, Windows machines that did not have the Visual Studio redistributable may not have been able to run DDS .NET applications out of the box. This dependency is now managed internally and no longer required by the user.

[RTI Issue ID CORE-13120]

Fixes Related to Java API

[Critical] Java API leaked some objects in certain DomainParticipantFactory operations

The Java API created and pinned a number of objects as a result of calling most methods in the DomainParticipantFactory, including the creation of *DomainParticipants*. While these objects did not consume significant amounts of memory, certain JVMs could have exhausted the maximum number of allowed global references, causing applications to fail. This problem has been resolved.

[RTI Issue ID CORE-12838]

[Major] get_typecode method of a DomainParticipant in Java API failed when the type contained a wstring element

In the Java API, calling the **get_typecode** method on a *DomainParticipant* for a registered type that contained a wstring element failed with the following exception:

```
Exception in thread "main" com.rti.dds.infrastructure.BAD_TYPECODE: Error␣
↪creating type code
```

```
Exception in thread "main" com.rti.dds.infrastructure.BAD_TYPECODE: Error␣
↪creating type code
at com.rti.dds.typecode.TypeCodeFactory.create_tc_from_native(TypeCodeFactory.
↪java:984)
```

```
Exception in thread "main" com.rti.dds.infrastructure.BAD_TYPECODE: Error␣
↪creating type code
at com.rti.dds.typecode.TypeCodeFactory.create_tc_from_native(TypeCodeFactory.
↪java:984)
at com.rti.dds.domain.DomainParticipantImpl.get_
↪typecode(DomainParticipantImpl.java:2027)
```

The exception was caused by a problem in the way the *Connext* Java API interfaced with its internal C implementation. This problem has been resolved.

[RTI Issue ID CORE-13302]

Fixes Related to Python API

[Critical] Possible deadlock between creation of a dds.Topic and a listener callback

A possible deadlock could have occurred, leaving the Python interpreter hanging indefinitely when a **dds.Topic** was created at the same time as a listener callback was in process. This problem has been resolved.

[RTI Issue ID PY-88]

[Major] DynamicData accessor for an enum member in a base type failed (Python API)

Given a DynamicData for a struct type (my_struct) with a base type containing an enum member (my_enum), the following code failed:

```
sample = dds.DynamicData(my_struct)
```

```
sample = dds.DynamicData(my_struct)
print(sample["my_enum"]) # error: member my_enum doesn't exist
```

This problem has been resolved.

[RTI Issue ID PY-30]

[Major] Possible incorrect default values when receiving extensible data

Given the following situation:

- An application uses a **dds.DataReader** for an extensible IDL type “T1” containing a non-optional primitive member “a”.
- The reader receives data for a different-but-compatible type “T2” that doesn’t define “a”.

The reader is expected to return a data sample where “a” is set to its default value (normally 0). However, in some situations the data sample may have contained an unexpected value for “a”. This problem has been resolved.

[RTI Issue ID PY-77]

[Major] Some APIs where missing, incorrectly named, or have been deleted

Removed types, methods, and fields:

- TopicInstance and all related operations in the *DataReader* and *DataWriter* have been removed.
- The static properties **dds.WriterDataLifecycle.auto_dispose_unregistered_instances** and **dds.WriterdataLifecycle.manually_dispose_unregistered_instances** have been removed due to being too similar to the non-static properties.
- The *DataReader* operations **read_next** and **take_next** have been removed.

Renamed types, methods and fields:

- **dds.ReaderDataLifecycle.autopurge_unregistered_instances_delay** was incorrectly named and has been renamed to **autopurge_nowriter_samples_delay**; **autopurge_nowriter_instances_delay** was missing and has been added.
- **dds.Filter.sql_filter_name** has been renamed to **dds.Filter.SQL_FILTER_NAME**; **dds.Filter.stringmatch_filter_name** has been renamed to **dds.Filter.STRINGMATCH_FILTER_NAME**. The same constants have been renamed in **dds.MultiChannel**.
- **dds.DataWriterResourceLimitsInstaceReplacementKind** was misspelled and has been renamed to **dds.DataWriterResourceLimitsInstanceReplacementKind**.
- **dds.TransportMulticast.settings** has been renamed to **dds.TransportMulticast.value**; **dds.TransportMulticastMapping.settings** has been renamed to **dds.TransportMulticastMapping.value**; **dds.TransportSelection.enabled_transports** has been renamed to **dds.TransportSelection.value**; **dds.TransportUnicast.settings** has been renamed to **dds.TransportUnicast.value**.

Newly added missing types, methods, and fields:

- The *DataReader* operation **acknowledge_sample** with **ack_response_data** was missing and has been added.
- **dds.Presentation.drop_incomplete_coherent_set** was missing and has been added.
- **dds.DomainParticipant** - the following methods have been added: **discovered_participant_subject_name, discovered_participants_from_subject_name, banish_ignored_participants**.
- **dds.DomainParticipantQos** - the following QoS policies have been added: **partition, default_unicast**.
- **dds.BuiltinTopicReaderResourceLimits** was missing **max_fragmented_samples_per_remote_writer**, which has now been added.
- The constant dds.DataReaderResourceLimits.AUTO_MAX_TOTAL_INSTANCES was missing and has been added.
- **dds.DataWriterProtocol.initial_virtual_sequence_number** was missing and has been added.
- **dds.DiscoveryConfigBuiltinChannelKindMask** was missing and has been added.
- **dds.DomainParticipantResourceLimits.serialized_type_object_dynamic_allocation_threshold** was missing and has been added.
- The constant dds.PublishMode.PUBLICATION_PRIORITY_UNDEFINED was missing and has been added.
- **dds.SystemResourceLimits.initial_objects_per_thread** was missing and has been added.
- **dds.DataWriterCacheStatus** was missing the following read-only properties, which have been added: **alive_instance_count, alive_instance_count_peak, disposed_instance_count, disposed_instance_count_peak, unregistered_instance_count, unregistered_instance_count_peak**.

- **dds.CompressionSettings** was missing the following constants, which have been added: COMPRESSION_LEVEL_DEFAULT, COMPRESSION_LEVEL_BEST_SPEED, COMPRESSION_LEVEL_BEST_COMPRESSION.
- **dds.Cookie** was missing a no-argument constructor, which has been added.
- **dds.AcknowledgmentInfo.cookie** was missing and has been added.
- The constant dds.FlowControllerProperty.DEFAULT_FLOW_CONTROLLER_NAME was missing and has been added.
- **dds.Property** can now be created from a dictionary.

Other

- In Entity types, listener is now a read-only property; use **set_listener** to change it with a status mask.
- The *DataReader* read/take operations include several changes. See RTI Connext Core Libraries What's New.
- **dds.GroupData's** constructor did not initialize the bytes correctly and has been fixed.
- Setting **dds.EntityName.name** and **role_name** to None explicitly was not supported and caused a crash. This has been fixed.

[RTI Issue ID PY-85]

[Major] Listeners may not have been called in some situations

Entity listener callbacks may not have been called in some situations, causing the application to miss notifications about Entity status changes. This problem was due to a bug in pybind11 version 2.8.1. The build instructions have been updated to require pybind11 2.9.0, which solves this problem.

[RTI Issue ID PY-92]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.9 Fixes Related to XML Configuration

[Critical] Memory leak after an error parsing XML file with <include> tag

If the user's application failed to parse an XML file containing an <include> tag, this caused a memory leak. For example:

```
<types>
<include file=""myFile.xml"">

<struct name=""MyStruct"">
    <member name=""m1"" type=""unknownType"" />
</struct>
```

```
</types>
```

This file cannot be parsed because **m1** refers to an unknown type. When the application finished, running a memory profiling tool such as Valgrind™ showed there was a memory leak. This problem has been resolved.

[RTI Issue ID CORE-12831]

[Minor] Failed to parse XML configuration file containing type member with useVector attribute

Connext libraries failed to parse XML files containing a type member with the attribute useVector, although this is a legal attribute.

For example:

```
<types>
    <struct name= "MyType">
        <member name="m1" sequenceMaxLength="100" useVector="true" type="int32
↪"/>
    </struct>
</types>
```

Parsing this file failed with the following error:

```
RTIXMLParser_validateOnStartTag:Parse error at line xxx: Unexpected attribute
↪'useVector'
```

This problem has been fixed.

[RTI Issue ID CORE-12949]

[Minor] XML composition overwrote system information properties with defaults instead of correct values

The XML composition mechanism (described in QoS Profile Inheritance and Composition) had an issue with the way system properties (described in System Properties) set in an XML Snippet were applied to a <domain_participant_qos> in an XML Profile referencing the Snippet. The properties set in the XML Snippet were not applied to the <domain_participant_qos>, which ended up using the automatic values generated by *Connext*.

Here is an example that illustrates the problem:

```
<qos_library name="SampleQoSLib">
    <qos_profile name="ParentProfile">
        <domain_participant_qos>
            <property>
                <value>
                    <element>
                        <name>dds.sys_info.hostname</name>
                        <value>CustomHostName</value>
                    </element>
                </value>
            </property>
        </domain_participant_qos>
    </qos_profile>

    <qos_profile name="ChildProfile" is_default_qos="true">
        <domain_participant_qos>
            <base_name>
                <element>SampleQosLib::ParentProfile</element>
            </base_name>
            <property>
                <value>
                    <element>
                        <name>dds.sys_info.username</name>
                        <value>CustomUserName</value>
                    </element>
                </value>
            </property>
        </domain_participant_qos>
    </qos_profile>
</qos_library>
```

The <domain_participant_qos> in the ChildProfile ended up with the following values for the system information properties:

- dds.sys_info.hostname - The default value rather than the CustomHostName value as set in the <domain_participant_qos> in ParentProfile, because of the overwriting problem described above.
- dds.sys_info.username - The set value of CustomUserName, which is the correct value.

This issue has been resolved.

[RTI Issue ID CORE-13090]

7.4.10 Fixes Related to Request-Reply and RPC

[Critical] Exceptions sending result of remote operation may have crashed server application

In an RPC server-side application, the user implements the functional interface. The Server uses a thread pool to call those functions with the input sent from the client (Request) and obtain the result. The result is then sent to the client (Reply). The Reply is automatically written using a DDS *DataWriter*. If the **write()** operation failed, the resulting exception would crash the current thread in the thread pool and possibly crash the entire server-side application (a typical **write()** exception is a Timeout). Since the Reply is sent by the server from a separate thread, the user application has no way of catching the exception or sending the Reply again.

This problem has been resolved. If an exception occurs, it is caught and logged. The Reply is never sent. User applications have two ways to react to this event:

- The server application can install a **rti::config::Logger::output_handler** to monitor errors.

- The client application will see a timeout in the function call. The application can then react accordingly (e.g., calling the function again later).

[RTI Issue ID REQREPLY-111]

[Critical] RPC: deadlock when Server::close() was called before Server::run()

In the unlikely scenario that a Server was created and then closed before running (the method **Server::close()** was called before **Sever::run()**), **run()** would never return unless a timeout was specified. This problem has been resolved.

[RTI Issue ID REQREPLY-113]

[Critical] Possible unbounded memory growth when creating many Requesters

This issue was fixed in release 7.0.0, but not documented at that time.

When a Requester is created, a ContentFilteredTopic is internally created on the Requester's *DomainParticipant*. This ContentFilteredTopic is exclusively created for each Requester and was never deleted until the *DomainParticipant* was deleted. This may have caused applications that continuously create and delete Requesters on the same *DomainParticipant* to see unbounded memory growth.

This problem has been resolved in all language APIs. The Requester destructor or deletion function now deletes its ContentFilteredTopic.

[RTI Issue ID REQREPLY-35]

[Critical] Memory leak in Java Request-Reply API

This issue was fixed in release 7.0.0, but not documented at that time.

The Java Request-Reply API leaked a small amount of native heap memory every time a Requester was created. The leak was caused by a few internal WaitSet objects, which did not have a finalizer and were not explicitly deleted either.

[RTI Issue ID REQREPLY-94]

[Critical] Possible data race using Sample and WriteSample classes (Traditional C++ API only)

This issue was fixed in release 7.0.0, but not documented at that time.

The Sample and WriteSample classes are wrapper classes in the Traditional C++ Request-Reply API that used to initialize the underlying user data lazily: the data was initialized the first time it was accessed with the **data()** member function.

This approach made the access to the data unsafe. A data race could occur when two or more threads competed to access the same sample object for the first time. This problem has been resolved. The lazy approach has been reversed, and the data is now initialized in the constructor.

[RTI Issue ID REQREPLY-95]

[Major] RPC interface evolution did not work

Remote Procedure Call (RPC) interfaces were designed to be extensible. A service and a client can communicate even when they have a different number of interfaces. For example:

A base service definition in IDL could be as follows:

```
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed);
    float get_speed();
};
```

If you add new operations to the service interface, such as the following:

```
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed);
    float get_speed();
    float get_position();
};
```

Or remove operations from the service interface, such as the following:

```
@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed);
};
```

They should remain interoperable.

However, in the previous release, the service and the client wouldn't communicate in any case.

This problem has been resolved. A client can now invoke an operation in a service with more or fewer operations. If the operation exists in the service, it will receive a valid response. If the operation doesn't exist in the service, the service will respond accordingly and the client will throw the standard exception **dds::rpc::RemoteUnknownOperationError**.

[RTI Issue ID REQREPLY-105]

7.4.11 Fixes Related to Shipped Examples

[Minor] Hello World TCP example always linked TCP Transport library dynamically

The C **hello_world_tcp** example always linked the *RTI TCP Transport* library dynamically, even if you wanted to use static linking. This issue has been fixed. Now, the **nddstransporttcp** library is linked statically unless you choose Debug DLL or Release DLL from the configuration pull-down menu of the provided projects on Windows. Or, when using a makefile, the *TCP Transport* library is now linked statically, unless you pass the "SHAREDLIB=1" argument to the make command.

Furthermore, the README file for the example has been updated with further instructions on what additional libraries need to be added to the makefile or project file when TLS is enabled.

[RTI Issue ID COREPLG-577]

7.4.12 Fixes Related to Vulnerabilities

[Critical] Arbitrary read access while parsing malicious RTPS message *

Arbitrary read access could occur while parsing a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Arbitrary read access while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on confidentiality of *Connext* application.
- CVSS Base Score: 8.2 HIGH
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:H

User Impact with Security

Same impact as described in “User Impact without Security,” above.

[RTI Issue ID CORE-13160]

[Critical] Out-of-bounds read while parsing malicious RTPS message

An out-of-bounds read could occur while parsing a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Out-of-bounds read while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on confidentiality of *Connext* application.
- CVSS Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L

User Impact with Security

Same impact as described in "User Impact without Security," above.

[RTI Issue IDs CORE-13240 and CORE-13264]

[Critical] Out-of-bounds write while parsing malicious RTPS message

An out-of-bounds write could occur while parsing a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Out-of-bounds write while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on integrity of *Connext* application.
- CVSS Base Score: 8.2 HIGH
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H

User Impact with Security

Same impact as described in "User Impact without Security," above.

[RTI Issue ID CORE-13279 and CORE-13150]

[Critical] Buffer overflow in shared memory if memory was tampered

A buffer overflow occurred when publishing or receiving metadata or data over a tampered shared memory segment. This issue has been fixed.

User Impact without Security

- Exploitable from the same node the *Connext* application is running (needs access to shared memory segment).
- Application crash. Potential impact to the integrity or confidentiality of the *Connext* application.
- CVSS Base Score: 7.8 HIGH
- CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

User Impact with Security

Same impact as described in “User Impact without Security,” above.

[RTI Issue ID CORE-13300]

[Critical] Out-of-bounds read while uncompressing malformed data from malicious RTPS message

An out-of-bounds read occurred while uncompressing malformed data from a malicious RTPS message. This issue has been fixed.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Out-of-bounds read while uncompressing malformed data from a malicious RTPS message.
- Remotely exploitable.
- Potential impact on confidentiality of *Connext* application.
- CVSS Base Score: 4.8 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:L

User Impact with Security

Same impact as described in “User Impact without Security,” above.

[RTI Issue ID CORE-13548]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.13 Fixes Related to Crashes

[Critical] Rare segmentation fault when deleting DomainParticipant or Publisher containing DataWriters using durable writer history

A *Connext* application may have crashed after deleting a *DomainParticipant* or *Publisher* containing *DataWriters* using durable writer history. This issue has been fixed.

[RTI Issue ID CORE-12297]

[Critical] Segmentation fault when creation of DomainParticipant failed due to lack of resources

An application may have produced a segmentation fault using the release libraries if the creation of a *DomainParticipant* failed because the following resource limit was exceeded: **participant_factory_qos.resource_limits.max_objects_per_thread**.

With debug libraries, you may have seen a precondition error such as this:

```
FATAL U000000011d1a15c0_ [CREATE DP|LC:DISC]Mx06:/connextdds/event.1.0/srcC/
↪activeDatabase/ActiveDatabase.c:275:RTI0x2000027:!precondition
```

This problem has been fixed.

[RTI Issue ID CORE-12654]

[Critical] Potential hang upon SIGSEGV signal from a Connext application

For debuggability purposes, *Connext* applications log a backtrace when a SIGSEGV signal is triggered.

In previous releases, this feature may have triggered a hang during the logging of the backtrace. In this release, we address this issue by disabling the logging of the backtrace by default in release libraries (but still keeping it enabled for debug libraries).

This default behavior can be modified by setting the new *DomainParticipant*-level property **dds.participant.enable_backtrace_upon_sigsegv**. See "New property to manually enable or disable logging backtrace upon SIGSEGV signal from a Connext application" in RTI Connext Core Libraries What's New.

[RTI Issue ID CORE-12794]

[Critical] Creating DynamicDataTypePlugin with TypeCode from discovery and using content filtering caused segmentation fault

If the TypeCode that was received from endpoint discovery data (**PublicationBuiltinTopicData.type_code or SubscriptionBuiltinTopicData.type_code**) was used to create a DynamicDataTypeSupport in an application that was also using ContentFilteredTopics and setting **ResourceLimitsQosPolicy.type_code_max_serialized_length** to a non-zero value, the application issued a segmentation fault.

ResourceLimitsQosPolicy.type_code_max_serialized_length is 0 by default, which avoids the segmentation fault.

This issue has been fixed.

[RTI Issue ID CORE-12992]

[Critical] Application crash when calling DDS_DataReader_take_discovery_snapshot on a DataReader with a ContentFilteredTopic *

When taking a discovery snapshot by calling the **DDS_DataReader_take_discovery_snapshot** function on a *DataReader* with a ContentFilteredTopic, the application crashed when trying to obtain non-valid *DomainParticipant* information. This issue has been fixed. Now, *DomainParticipant* information is obtained correctly for *DataReaders* with ContentFilteredTopics.

[RTI Issue ID CORE-13011]

[Critical] Crash with NULL listeners and non-none status masks in C applications that mixed types with and without Zero Copy

In a C application, a crash occurred when both of these were true:

- Types with and without Zero Copy transfer over shared memory were mixed inside the same DomainParticipantFactory instance.
- A *DataReader* or *DataWriter* of the non-Zero Copy types had a NULL listener and a **DDS_StatusMask** different than DDS_STATUS_MASK_NONE.

The crash occurred because *Connext* invoked a NULL listener callback for the statuses enabled in the endpoints' **DDS_StatusMask**.

When there is a Zero Copy type inside an application, some extra pre-processing related to Zero Copy is done before creating the endpoints and setting the listeners. In that extra pre-processing, for non-Zero Copy types, the NULL listener was incorrectly replaced with a non-null listener object with all its callbacks set to NULL. Then, *Connext* was not checking if the callbacks were NULL before calling them (the listener consistency is checked before the incorrect replacement; therefore, at that point, it was assumed the listener object was consistent).

This issue is fixed. The listener is no longer replaced with an invalid listener object, and *Connext* will always check if the callbacks are NULL before calling them.

[RTI Issue ID CORE-13151]

[Critical] Memory was read after it was freed by deleting a Topic with local logging level enabled

If the local logging level was enabled while deleting a topic, *Connext* would use recently freed memory from the deleted *Topic* to print a log message. Using the recently freed memory could cause a crash if local logging was enabled. A log message is now printed immediately before the *Topic* is deleted, so the possibility of using freed memory is eliminated.

[RTI Issue ID CORE-13226]

[Critical] Possible segmentation fault when disabling loopback interface

When a previously enabled loopback interface on a host computer was disabled, a segmentation fault could occur. The handling of loopback interfaces has been redesigned to remove this possibility.

[RTI Issue ID CORE-13228]

[Critical] Segmentation fault could occur if creation of DataReader failed

In some cases, a segmentation fault would occur if the creation of a *DataReader* failed. This problem has been fixed.

[RTI Issue ID CORE-13387]

[Critical] Potential crash when DomainParticipant deleted after creating DataWriter with automatic liveliness kind

There was a small possibility of a crash occurring when the *DomainParticipant* was deleted immediately after creating a *DataWriter* with an AUTOMATIC_LIVELINESS_QOS **kind** in the LIVELINESS QoS policy. This problem has been resolved.

[RTI Issue ID CORE-13524]

[Critical] Possible crash on TCP transport when large number of file descriptors were open

A *Connext* application that used the TCP transport and was built using **_FORTIFY_SOURCE**, which is set by default by some operating systems, could crash if one of the sockets for TCP had a file descriptor higher than FD_SETSIZE (1024). This issue has been fixed. Now, *Connext* overwrites the value of FD_SETSIZE, allowing an application using the TCP transport to open up to 32768 file descriptors, except on Android, where it is not possible to overwrite this value.

[RTI Issue ID COREPLG-644]

[Critical] Application using Monitoring Libraries may have produced segmentation fault during DataReader creation

In 6.0.x releases and above, an application using *Monitoring Library* may have produced a segmentation fault during *DataReader* creation. The issue was very rare and only occurred if a *DataReader* received a sample immediately after being enabled. This issue has been fixed.

[RTI Issue ID MONITOR-429]

[Critical] Possible segmentation fault when using Monitoring Library

When using monitoring libraries, a rare race condition may have led to a segmentation fault. This issue was more likely to occur if the *Connext* application using the monitoring libraries created and deleted entities often. This problem has been resolved.

Note: This problem was reported as fixed in MONITOR-252, in release 6.0.1; however, that fix did not apply to *Publishers* and *Subscribers*. This fix protects applications when frequently creating and deleting *Publisher* or *Subscriber* entities as well.

[RTI Issue ID MONITOR-516]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.4.14 Other Fixes

[Critical] Broken communication when DataWriter with transport priority discovered DataReader with multicast receive address

If a *DataWriter* that had a non-default **DataWriterQos.transport_priority** value set discovered a *DataReader* with a multicast receive address, the *DataWriter* and any other *DataWriters* within the same participant were not able to send any traffic over unicast. This could cause communication failures in a number of different scenarios, including a broken reliability protocol due to the inability to send heartbeats over unicast or the inability to communicate with other *DataReaders* that have not been configured to use a multicast receive address.

This issue was introduced in 6.1.0. This issue has been resolved.

[RTI Issue ID CORE-12772]

[Critical] Potential hang upon SIGSEGV signal from a Connext application

For debuggability purposes, *Connext* applications have the ability to flog a backtrace when a SIGSEGV signal is triggered.

In previous releases, this feature may have triggered a hang during the logging of the backtrace. In this release, we address this issue by disabling the logging of the backtrace in release libraries (but still keeping it enabled for debug libraries).

This default behavior can be modified by setting a new participant-level property, **dds.participant.enable_backtrace_upon_sigsegv**. The accepted values for this new property are: “auto” for the default behavior (backtrace only enabled in debug libraries), “true” for enabling the logging of the backtrace, and “false” for disabling it.

Note: This property takes effect upon the creation of the first *DomainParticipant* within a process. Consequently, if a SIGSEGV signal is received before the creation of the first *DomainParticipant*, the default behavior will be applied (backtrace enabled in debug libraries and disabled in release libraries).

[RTI Issue ID CORE-12794]

[Critical] Samples could be lost using group order access or collaborative DataWriters

There was a possibility of *DataReader* queue corruption, when using group order access or collaborative *DataWriters*, that may have provoked the *DataReader* to stop receiving samples. The possibility was very small and may have occurred randomly since it was caused by an uninitialized flag.

[RTI Issue ID CORE-13153]

[Critical] Release 6.1.2 was not FACE compliant

The |*CONNEXTDDS_ITALIC*| 6.1.2 release was not FACE compliant due to usage of the realpath system call. This problem has been resolved.

[RTI Issue ID CORE-13340]

[Critical] Using dh_param_files leaked memory

Using the property **tls.cipher.dh_param_files** leaked memory when deleting the *DomainParticipant*. A memory checking tool, such as valgrind, would have reported the leak in the OpenSSL function **PEM_read_bio_DHparams**, which is called by the RTI function **RTITLS_tmp_dhparam_callback**. This problem only affected applications using OpenSSL 1.0.2 or applications communicating with applications using OpenSSL 1.0.2. For example, *TLS Support* 5.3 uses OpenSSL 1.0.2, but version 7.0.0 of *TLS Support* could still communicate with version 5.3, so the leak could also happen in version 7.0.0.

This problem has been fixed; memory will no longer be leaked in this scenario. For example, if *TLS Support* 7.1.0 communicates with an application using OpenSSL 1.0.2, the leak will not occur.

[RTI Issue ID COREPLG-641]

[Critical] Segmentation fault when mixing build types in applications linked against libraries in "Find Package" Cmake script

Mixing Release and Debug build types in applications linked against *Connext* libraries in the "Find Package" script (**FindRTIConnextDDS.cmake**) could lead to undesired behaviors like double-freeing pointers, once for the Debug symbol and once for the Release symbol, and in the end causing the application to abort.

The new CONNEXT_LIBS_BUILD_TYPE CMake variable has been added to control the *Connext* libraries build type (Release/Debug). This variable will allow three values: Auto, Release, and Debug.

By default (the Auto value), **FindRTIConnextDDS.cmake** will populate the IMPORTED_LOCATION_DEBUG and IMPORTED_LOCATION_RELEASE properties of all the *Connext* imported target libraries. This means that the *Connext* libraries will be provided in the same build type as the global build (specified by the CMAKE_BUILD_TYPE value).

If you provide Release or Debug values to the CONNEXT_LIBS_BUILD_TYPE variable, the script will force populating only the IMPORTED_LOCATION property of the *Connext* imported target libraries. So, regardless of the CMAKE_BUILD_TYPE value, the *Connext* libraries will have the build type given in the CONNEXT_LIBS_BUILD_TYPE variable.

[RTI Issue ID INSTALL-793]

[Major] Error sending batch when batch size exceeded transport MTU

A *DataWriter* configured to use batching may have failed to send a batch to the destination addresses associated with a transport (e.g, UDPv4) if the batch size exceeded the **message_size_max** (MTU) of the transport.

This problem has been resolved. Now, the batch is automatically flushed when exceeding the minimum **message_size_max** across all installed transports.

[RTI Issue ID CORE-2639]

[Major] No more than 100 asynchronous publisher threads could be created

A change to the thread naming convention inadvertently limited the number of asynchronous publisher threads to 100. The limit is now 65,536. These limits also apply to receive threads, asynchronous waitset threads, and persistence service threads.

[RTI Issue ID CORE-12874]

[Major] Unexpected precondition error while creating a DomainParticipant with debugging libraries using fast database cleanup period

You may have seen the following precondition error while creating a *DomainParticipant* with debugging libraries if **participant_qos.database.cleanup_period** was updated to a small value.

```
FATAL rCo96144####Dtb Mx0D:/rti/jenkins/workspace/connextdds_ci_fastbuild-
↪debug_develop/pres.1.0/srcC/participant/Participant.c:3102:RTI0x200003b:!
↪precondition: "me->_service == ((void *)0)"
```

Release libraries did not have this issue.

This problem has been fixed.

[RTI Issue ID CORE-13204]

[Major] In FindPackage script, low_bandwidth_edisc imported target library was missing

In the "FindPackage" script (**FindRTIConnextDDS.cmake**), the **low_bandwidth_edisc** imported target library was missing, incorrectly named **low_bandwidth_discovery_static**. When you tried to link against **low_bandwidth_discovery_static**, the script actually linked against the LOW_BANDWIDTH_EDISC libraries. And you couldn't link against **low_bandwidth_edisc** because there was no imported target with that name.

In the following example, the second TARGET should have been called **low_bandwidth_edisc**:

```
######################## Low bandwidth plugins ########################
    # Discovery Static
    create_connext_imported_target(
        TARGET "low_bandwidth_discovery_static"
        VAR "LOW_BANDWIDTH_DISCOVERY_STATIC"
        DEPENDENCIES
```

```
            RTIConnextDDS::c_api
    )

    # EDISC
    create_connext_imported_target(
        TARGET "low_bandwidth_discovery_static"
        VAR "LOW_BANDWIDTH_EDISC"
        DEPENDENCIES
            RTIConnextDDS::c_api
    )
```

This problem has been fixed.

[RTI Issue ID INSTALL-719]

[Minor] Potential memory leak when creation of any of the built-in discovery plugins failed

The first time a *DomainParticipant* is created in an application, some memory is allocated globally for each of the builtin discovery plugins (SPDP and SEDP) enabled for that *DomainParticipant*. This global memory is released when finalizing the DomainParticipantFactory instance.

However, if there was a failure in the creation of any of the builtin discovery plugins during the *DomainParticipant* creation, the *DomainParticipantFactory* was not notified properly that this global memory was allocated. Therefore, finalizing the *DomainParticipantFactory* instance did not release the memory, causing a leak.

This problem is fixed. Finalizing the *DomainParticipantFactory* instance always releases the memory if it was previously allocated, regardless of whether or not a failure occurred.

[RTI Issue ID CORE-12882]

[Minor] Problems visualizing participants using Generic.MinimalMemoryFootprint profile with Admin Console

RTI Admin Console could not correctly visualize *DomainParticipants* using the **Generic.MinimalMemory-Footprint** profile. Some of the information, such as process ID and host name, was invalid. This problem has been fixed.

[RTI Issue ID CORE-13509]

[Minor] Failure to load a string-based private key leaked memory

If you set the property **tls.identity.private_key** or **tls.identity.rsa_private_key**, and you either specified a wrong or missing value for the property **tls.identity.private_key_password** or specified a malformed private key, then memory would be leaked upon *DomainParticipant* creation failure. A memory checking tool, such as valgrind, would report the leak in the OpenSSL function **BIO_new_mem_buf**, which is called by the RTI function **RTITLS_context_init**.

This problem has been fixed. Memory will no longer be leaked in this scenario.

[RTI Issue ID COREPLG-643]

[Minor] CONNEXTDDS_ARCH environment variable in FindPackage script was not picked up correctly

Previously, only the CONNEXTDDS_ARCH CMake variable in the "FindPackage" script (**FindRTIConnextDDS.cmake**) could be used to define the *Connext* official architecture to use. Now, the environment variable with the same name can be used, too.

[RTI Issue ID INSTALL-691]

[Trivial] Incorrect "Supported platforms" documentation section for FindRTIConnextDDS.cmake

Now the documentation section in the "FindPackage" script (**FindRTIConnextDDS.cmake**) file listing the "Supported platforms" matches the *Core Libraries Platform Notes*.

[RTI Issue ID INSTALL-548]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

7.5 What's New in 7.0.0

For what's new in the Core Libraries in 7.0.0, see What's New in 7.0.0 in the Previous Releases section of the *RTI Connext What's New*.

7.6 What's Fixed in 7.0.0

This section describes bugs fixed in *Connext* 7.0.0. These are fixes since 6.1.2.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

7.6.1 Fixes Related to Callbacks and Waitsets

[Critical] Unsafe combinations of masks and Listeners may have led to segmentation fault

When entities are created, a *Listener* may be provided by the user to receive calls when specified events occur. The events of interest are set using a **StatusKind** mask. If an event set in the **StatusKind** mask occurs, but no callback function has been assigned by the user, a null pointer dereference will occur. *Connext* checks for many of these errors and prevents the creation of entities when this error is present. However, some of these cases were not checked, allowing unsafe combinations of masks and *Listeners* to be used. This problem has been resolved. The new, stricter checking may cause entity creation errors when no errors were detected before.

[RTI Issue ID CORE-12610]

[Critical] Failure calling DDS_Subscriber::get_datareaders in DDS_SubscriberListener::on_data_on_readers callback implementation

You may have seen the following errors when invoking **DDS_Subscriber::get_datareaders()** within the implementation of the **DDS_SubscriberListener::on_data_on_readers()** callback:

```
ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS] REDACursor_modifyReadWriteArea:!freeze read write area

ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS] PRESPsReaderGroup_getEA:!modify pres psReaderGroup
ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS] PRESPsReaderGroup_lock:!take semaphore
ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS] PRESPsReaderGroup_beginGetPsReaders:!get PRESPsReaderGroup_lock

ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS}
DDS_Subscriber_begin_get_datareadersI:ERROR: Failed to get PRESPsReaderGroup_
↪beginGetPsReaders

ERROR [0x01011B2D,0x8A450DE1,0xBAE5A2A0:0x80000009|SET GROUP LISTENER|GET␣
↪READERS]
DDS_Subscriber_get_datareaders:ERROR: Failed to get DDS_Subscriber_begin_get_
↪datareaders
```

In addition, when using the Traditional C++ API and the legacy .NET API, the application generated a segmentation fault after printing the error. The problem occurred only when:

- You installed a *Listener* on the *Subscriber* using the API **DDS_Subscriber::set_listener()** after the *Subscriber* was enabled.
- Or, you installed a *Listener* on the *DomainParticipant* using the API **DDS_Participant::set_listener()** after the *DomainParticipant* was enabled. This problem has been resolved.

[RTI Issue ID CORE-12316]

[Critical] Using certain callbacks at DomainParticipant or Publisher level may have led to segmentation fault

Handlers were not correctly implemented for the **on_instance_replaced()**, **on_sample_removed()**, **on_application_acknowledgment()**, and **on_service_request_accepted()** callbacks at the *DomainParticipant* and *Publisher* levels. This could have led to segmentation faults when the corresponding events were enabled. This problem has been resolved.

[RTI Issue ID CORE-12647]

[Major] DDS_SubscriberListener::on_data_on_readers on a participant or subscriber not called when Listener installed after the entity is enabled

The callback **DDS_SubscriberListener::on_data_on_readers()** was not invoked when there was data available, if these two conditions were met:

- The *Listener* callback **on_data_on_readers()** was installed after the *Subscriber* or *DomainParticipant* implementing it was enabled.
- The *Listener* callback **on_data_available()** was not installed at any level (*DomainParticipant, Publisher*, or *DataReader*).

This problem has been resolved.

[RTI Issue ID CORE-12338]

[Major] Unable to assign callback function for on_sample_removed event using Modern C++ API

You may have been unable to assign a callback function for the on_sample_removed event using the Modern C++ API. Support for this callback has been added to the Modern C++ API in this release.

[RTI Issue ID CORE-12646]

7.6.2 Fixes Related to Discovery

[Critical] Unexpected memory growth when DataReader could not be matched with DataWriter due to unexpected error condition

Failing to match a *DataReader* with a *DataWriter* because of unexpected error conditions may have led to unexpected memory growth, because *Connext* may not have cleaned up the resources associated with the remote match completely. This problem has been resolved.

[RTI Issue ID CORE-8257]

[Critical] Possible crash upon discovery of applications with unreachable locators

If an application used **DDS_STATUS_MASK_ALL** for a *DomainParticipant* or *Publisher Listener* and an unreachable locator was discovered, the application enabling the *Listener* may have crashed. An unreachable locator occurs most commonly when a Subscribing application uses a transport that the Publishing application does not use. For example, the Publishing application could use UDPv4 and the Subscribing application could use both UDPv4 and UDPv6.

More rarely, a crash may have occurred when a pre-5.2.0 Subscribing application used the shared memory transport and a 5.2.0+ Publishing application was not using the UDPv6 transport. A log message was generated if both participants were running on the same machine and this condition occurred. This condition was caused by a change to the way that transports are identified starting in version 5.2.0.

[RTI Issue ID CORE-11818]

[Critical] Communication problems with applications using shared memory on INTEGRITY systems

If an application on an INTEGRITY platform used the shared-memory transport, the *Connext* libraries sometimes incorrectly assessed that a shared-memory segment was stale and could be reclaimed, when in fact it was not stale. This situation caused problems with communication between *DomainParticipants*, since information could be sent to a shared-memory segment that did not get dequeued by the intended recipient.

You may have seen error messages like these and the application may have hung while deleting the *DomainParticipant*:

```
<Target Output> ERROR RTIOsapiSharedMemoryBinarySemaphore_take:OS␣
↪WaitForSemaphore() failure, error 0XD: ObjectClosed
<Target Output> ERROR NDDS_Transport_Shmem_receive_rEA:!take semaphore
<Target Output> ERROR RTIOsapiSharedMemoryBinarySemaphore_take:OS␣
↪WaitForSemaphore() failure, error 0X9: ObjectIsUseless
```

This problem has been resolved.

Incompatibility with 6.1.1 and prior releases:
The fix for this issue involved some changes that make shared-memory segments in applications incompatible with those in 6.1.1 (and earlier) versions.

[RTI Issue ID CORE-12097]

[Critical] Unbounded memory growth in Spy when discovering multiple endpoints with the same Topics and types

Each time *DDS Spy* discovered an endpoint, it unnecessarily made a copy of the TypeCode that was associated with the endpoint's *Topic*, leading to unbounded memory growth. This issue has been fixed.

[RTI Issue ID CORE-12136]

[Major] Types containing Typedefs were sent without the typedefs in discovery when using DynamicData

When an application was using a DynamicDataReader or DynamicDataWriter and using a type that contained a typedef, the type that was sent during endpoint discovery for that endpoint did not contain the typedef. While this did not cause any mismatches or communication failure, it did cause a number of issues that may have been noticeable depending on what other products you may have also been using.

See "Unbounded memory growth in Spy when discovering multiple endpoints with the same Topics and types," below, for details about the specific issues that you may have encountered. The *RTI Admin Console Release Notes* and *RTI Routing Service Release Notes* also have related information. (See ADMINCONSOLE-997 and ROUTING-971, respectively.)

This issue has been resolved, meaning that the exact type definition that is registered with the participant, containing typedefs, is sent during discovery. This is a change in behavior from 6.0.0-based applications, which sent the type definitions without the typedef information.

[RTI Issue ID CORE-12107]

[Major] Unnecessary discovery traffic related to IP mobility events on interfaces irrelevant to the transport

When there is a change on a network interface (an IP mobility event), a *Connext* application will update and resend its discovery information to include these changes. The transport can consider a change on an interface irrelevant (for example, changes on interfaces in the **deny_interfaces_list** of the transport). In this case, the new discovery messages are exactly the same as announced before, generating unnecessary discovery traffic that could affect the performance of the application.

This problem has been fixed. Now *Connext* only updates and resends its discovery information if there was a change on an interface relevant to the transport.

[RTI Issue ID CORE-12664]

7.6.3 Fixes Related to Transports

[Critical] Communication problems with applications using shared memory on INTEGRITY systems

If an application on an INTEGRITY platform used the shared-memory transport, the *Connext* libraries sometimes incorrectly assessed that a shared-memory segment was stale and could be reclaimed, when in fact it was not stale.

This situation caused problems with communication between *DomainParticipants*, since information could be sent to a shared-memory segment that did not get dequeued by the intended recipient.

You may have seen error messages like these and the application may have hung while deleting the *DomainParticipant*:

```
<Target Output> ERROR RTIOsapiSharedMemoryBinarySemaphore_take:OS␣
↪WaitForSemaphore() failure, error 0XD: ObjectClosed
<Target Output> ERROR NDDS_Transport_Shmem_receive_rEA:!take semaphore
<Target Output> ERROR RTIOsapiSharedMemoryBinarySemaphore_take:OS␣
↪WaitForSemaphore() failure, error 0X9: ObjectIsUseless
```

This problem has been resolved.

Incompatibility with 6.1.1 and prior releases:

The fix for this issue involved some changes that make the shared memory segments incompatible with those in 6.1.1 (and earlier) versions.

[RTI Issue ID CORE-12097]

[Critical] Race condition could cause unbounded memory growth in TCP Transport Plugin

Due to a race condition, the TCP Transport Plugin may have leaked memory when creating a new connection if the creation happened at the same time the *DomainParticipant* was being destroyed. The cause of the leak was the TCP Transport Plugin reallocating memory that was already released by the *DomainParticipant*. The race condition was unlikely to happen. However, in a system that frequently creates and destroys entities (and, therefore, TCP connections) and that runs for long enough, it may have lead to unbounded memory growth. The issue has been resolved.

[RTI Issue ID COREPLG-618]

7.6.4 Fixes Related to Filtering and TopicQuery

[Critical] Connext application using filtering feature may have crashed after running out of memory

In release 6.1.1.2, a *Connext* application using filtering features (that is, ContentFilteredTopic, QueryConditions, or TopicQuery) may have crashed after running out of memory. This problem has been resolved.

[RTI Issue ID CORE-12661]

[Critical] Creation of a ContentFilteredTopic or reception of TopicQuery samples may have taken long time for complex types

The creation of a ContentFilteredTopic or reception of TopicQuery samples, may have taken a long time for complex types. This issue has been resolved.

[RTI Issue ID CORE-12179]

[Critical] rti::topic::find_registered_content_filters led to infinite recursion

The function **rti::topic::find_registered_content_filters()** was incorrectly implemented and would lead to infinite recursion and stack overflow in any application that called it. This problem has been resolved. This function returns the names of previously registered custom content filters. It is a little-used feature and does not affect the commonly used SQL content filter.

[RTI Issue ID CORE-12512]

[Critical] Incorrect results for Unions when using DynamicData or Content Filters

When using a DynamicDataReader, samples containing a union may have had incorrect or invalid data after deserialization if the *DataReader's* type contained members that were not present in the *DataWriter's* type and those members had non-zero default values.

When using content filters, the filter results may have been incorrect if the type contained a union and the filter expression filtered on fields within the union that were present in the *DataReader's* type but were not present in the *DataWriter's* type and those members had non-zero default values.

For example, see this *DataWriterType*:

```
struct innerStructPub {
    short shortMember;
};
@mutable
union ComplexUnionTypePub switch(long) {
    case 0:
        long longMember;
    case 1:
        innerStructPub structMember;
};
```

and this *DataReaderType*:

```
struct innerStructPub {
    short shortMember;
};
@mutable
union ComplexUnionTypePub switch(long) {
    case 0:
        long longMember;
    case 1:
        innerStructPub structMember;
};
struct innerStructSub {
    short shortMember;
    @default(5) long longMemberWithDefault;
};
@mutable
union ComplexUnionTypeSub switch(long) {
    case 0:
        long longMember;
    case 1:
        innerStructSub structMember;
};
```

In the above types, the member **longMemberWithDefault** is only present in the *DataReader's* type and has a default value of 5, so any sample that is received from the *DataWriter* should have this value set to 5 when read from the *DataReader's* queue. Instead, the value was incorrectly 0 when using DynamicData.

In addition, if this member was used as part of a content filter expression, a *DataReader* always used the value of 0 instead of 5 when evaluating a sample from a *DataWriter* using the DataWriterType which could lead to incorrect filter results. These issues have been fixed.

[RTI Issue ID CORE-12517]

[Major] Unnecessary repair traffic for DataWriters using TopicQueries and asynchronous publishing

Samples that are sent in response to a TopicQuery are directed to the *DataReader* that created that TopicQuery. This means that those samples are only sent to the *DataReader* that made the request and have that *DataReader's* GUID attached to each sample in the sample's metadata. All other *DataReaders* receive GAP protocol messages, indicating to them that a given sequence number or set of sequence numbers is not meant for them.

Due to a defect, when a *DataReader* sent a NACK message requesting some TopicQuery samples to be repaired, if the requested sequence numbers included samples that were meant for a different *DataReader*, the *DataWriter* did not filter these samples and resend a GAP message. Instead, the *DataWriter* sent the *DataReader* samples that were not meant for it and the *DataReader* had to filter these samples out itself. As a result, the *DataReaders* may have received samples that should have been filtered out on the *DataWriter* side, leading to an increase in network traffic.

The problem only affected repair traffic. When a sample was filtered out by the *DataWriter* because it was directed to a different *DataReader*, the *DataWriter* sent a GAP protocol message to the *DataReader*. If the GAP message was lost, the *DataReader* NACKed for the sample; instead of sending a new GAP message, the *DataWriter* sent the sample. This problem has been resolved.

[RTI Issue ID CORE-12589]

[Major] Continuous creation of TopicQueries may have led to unnecessary memory fragmentation in OS memory allocator

In releases 6.0.x and 6.1.x, the continuous creation of TopicQueries may have led to unnecessary memory fragmentation in the OS memory allocator of the applications that receive the TopicQuery requests and dispatch responses. This issue may have resulted in an unexpected increase of the resident set size (RSS) memory of the application receiving and dispatching the TopicQueries compared to previous *Connext* releases. This problem has been fixed.

[RTI Issue ID CORE-12352]

[Major] Samples may have been unnecessarily filtered by Connext DataReader when DataWriter was from different DDS vendor

A Connext *DataReader* using a ContentFilteredTopic unnecessarily evaluated its filter on samples coming from a different vendor *DataWriter* that already marked the samples as passing the *DataReader* filter. This issue may have led to an increase in CPU utilization on the *DataReader* side, but it did not affect functional correctness or bandwidth utilization.

The problem occurred because *Connext* was not compliant with the way a filter signature is calculated according to the Section 9.6.4.1, *Content filter info (PID_CONTENT_FILTER_INFO)*, in the Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPSTM) Specification version 2.5).

This problem has been resolved.

[RTI Issue ID CORE-12531]

[Minor] Unnecessary sample filtering on a DataReader for samples already filtered by a DataWriter

When doing writer-side filtering, a late-joining *DataReader* using a ContentFilteredTopic may have spent unnecessary CPU cycles evaluating samples that pass the ContentFilteredTopic's expression. When using writer-side filtering, the filter evaluation is done by the *DataWriter* and it should not be necessary for the *DataReader* to do it again on samples that pass the filter expression. This problem, which only occurred for late-joining *DataReaders*, has been fixed.

[RTI Issue ID CORE-11084]

7.6.5 Fixes Related to Group Presentation

[Critical] Application may not have received samples of coherent set when using GROUP access scope and TRANSIENT_LOCAL durability

An application using **GROUP** access scope and **TRANSIENT_LOCAL** (or higher) durability may not have received the samples for some coherent sets, or it may have received the samples with delay.

Assume a coherent set '**CS1**' published by a set of *DataWriters* that are part of the same group. This coherent set was not provided to the application if all the following conditions were true:

1. The *DataReaders* receiving '**CS1**' matched with the *DataWriters* publishing '**CS1**' after the coherent set was published.
2. '**CS1**' did not contain samples for some of the *DataWriters* in the group, or the samples were removed after applying the Lifespan QoS Policy. If '**CS1**' contained at least one sample per *DataWriter* in the group, this problem did not occur.
3. The application did not publish a new coherent set after '**CS1**'; or, if it did, the new coherent set did not contain samples from at least one of the *DataWriters* that were missing samples from '**CS1**'.

If the third condition was not met, then the delivery of the coherent set would be delayed instead of the coherent set not being provided.

[RTI Issue ID CORE-12350]

[Critical] Segmentation fault when using GROUP_PRESENTATION_QOS or HIGHEST_OFFERED_PRESENTATION_QOS and setting filter_redundant_samples to FALSE on DataReader

An application generated a segmentation fault if it created a *DataReader* with the following valid configuration:

- subscriber_qos.presentation.access_scope = DDS_GROUP_PRESENTATION_QOS or DDS_HIGHEST_OFFERED_PRESENTATION_QOS
- datareader_qos.availability.max_data_availability_waiting_time = DDS_DURATION_ZERO
- datareader_qos.availability.max_endpoint_availability_waiting_time = DDS_DURATION_ZERO
- **datareader_qos.property** contained **dds.data_reader.state.filter_redundant_samples** with the value "**false**"

This problem has been resolved by allowing the *DataReader* to be created.

[RTI Issue ID CORE-12771]

[Major] Application may stop receiving samples from DataReaders using GROUP_PRESENTATION_QOS

An application may have stopped receiving samples from *DataReaders* that were part of a *Subscriber* using **GROUP_PRESENTATION_QOS** under the following scenario:

- The *Publisher's* group contained at least one keyed *DataWriter* and one unkeyed *DataWriter*.
- The *Subscriber's* group contained only keyed *DataReaders* or unkeyed *DataReaders*, but not both.

This problem has been resolved.

[RTI Issue ID CORE-12161]

7.6.6 Fixes Related to XML Configuration

[Major] Parsing error loading XML configuration file containing a const whose expression refers to an enumerator

Connext failed to load an XML configuration file containing a const whose expression referred to an enumerator. For example:

```
<enum name="Enum1">
    <enumerator name="Enumerator1" value="1"/>
</enum>
<const name="Const1" type="int32" value="Enumerator1+1"/>
```

Loading this XML failed with an error similar to this:

```
DDS_XMLConst_initialize:Parse error at line 10: type 'Enum1' is not typedef
```

This problem has been fixed.

[RTI Issue ID CORE-5553]

[Major] Parsing error loading an XML configuration file with enum type containing enumerator whose value was an expression

Connext failed to load an XML configuration file with an enum type containing an enumerator whose value was an expression. For example:

```
<enum name="Enum1">
    <enumerator name="Enumerator1" value="1 + 1"/>
</enum>
```

Loading this XML failed with an error similar to this:

```
DDS_XMLEnum_on_start_tag:Parse error at line xy: integer expected
```

This problem has been fixed.

[RTI Issue ID CORE-10269]

[Major] Parsing error loading an XML configuration file with an enum type containing an enumerator whose value was an expression referring to another enumerator

Connext failed to load an XML configuration file with an enum type containing an enumerator whose value was an expression using another enumerator. For example:

```
<enum name="Enum1">
    <enumerator name="Enumerator1" value="0"/>
</enum>

<enum name="Enum2">
    <enumerator name="Enumerator2" value="Enumerator1"/>
</enum>
```

Loading this XML would have failed with an error similar to this:

```
DDS_XMLEnum_on_start_tag:Parse error at line xy: integer expected
```

This problem has been fixed.

[RTI Issue ID CORE-12781]

[Minor] Discrepancy between range defined by schema and that defined by API

There were discrepancies between the ranges defined by the schema files and those defined by the API for certain elements. This problem has been resolved. Now, validating an XML against the XSD should not fail when setting a value that is inside the range as defined by the API.

[RTI Issue ID CORE-7099]

[Minor] Parsing error loading XML configuration file with enum type containing enumerator whose value was an expression referring to a const

Connext failed to load an XML configuration file with an enum type containing an enumerator whose value was an expression referring to a const. For example:

```
<const name="Const1" type="int32" value="10"/>
<enum name="Enum1">
    <enumerator name="Enumerator1" value="Const1"/>
</enum>
```

Loading this XML failed with an error similar to this:

```
DDS_XMLEnum_on_start_tag:Parse error at line xy: integer expected
```

This problem has been fixed.

[RTI Issue ID CORE-10060]

[Minor] Type limits not checked for some attributes of XML types definition

When XML was used for defining types (for example, when using DynamicData), type limits were not checked for some attributes. If the specified value for any of the attributes was too large or too small, a variable overflow occurred, leading to undefined behavior.

This problem is fixed. Type limits are checked, throwing a meaningful error when they are not met.

The affected attributes were as follows:

- **value** in union's **caseDiscriminator**. Valid values should be between -2147483648 and 2147483647.
- **sequenceMaxLength**. Valid values should be between 0 and 2147483647. -1 (unbounded) is also allowed.
- **stringMaxLength**. Valid values should be between 0 and 2147483647. -1 (unbounded) is also allowed.
- **arrayDimensions**. Valid values should be between 1 and 4294967295.

[RTI Issue ID CORE-12181]

[Trivial] Removed some elements in the XSD that were not supported internally but could be defined in XML

The following elements were configurable in XML although internally they are not supported:

Publisher QoS:

- presentation.drop_incomplete_coherent_set
- asynchronous_publisher.thread.cpu_list
- asynchronous_publisher.thread.cpu_rotation
- asynchronous_publisher.asynchronous_batch_thread.cpu_list
- asynchronous_publisher.asynchronous_batch_thread.cpu_rotation
- asynchronous_publisher.topic_query_publication_thread.cpu_list
- asynchronous_publisher.topic_query_publication_thread.cpu_rotation

Participant QoS:

- discovery_config.publication_reader.min_app_ack_response_keep_duration
- discovery_config.subscription_reader.min_app_ack_response_keep_duration
- discovery_config.asynchronous_publisher.thread.cpu_list
- discovery_config.asynchronous_publisher.thread.cpu_rotation
- discovery_config.asynchronous_publisher.disable_asynchronous_batch
- discovery_config.asynchronous_publisher.asynchronous_batch_thread
- discovery_config.asynchronous_publisher.disable_topic_query_publication
- discovery_config.asynchronous_publisher.topic_query_publication_thread

EventQosPolicy:

- thread.cpu_list
- thread.cpu_rotation

DatabaseQosPolicy:

- thread.cpu_list
- thread.cpu_rotation

Those elements have been removed from the XSD and are no longer configurable in XML.

[RTI Issue ID CORE-12366]

[Trivial] Builtin Discovery Plugins was not treated as a mask by the XSD file

Because of a bug in the XML Schema Definition (XSD), if you specified more than one value for the **DiscoveryConfigQosPolicy::builtin_discovery_plugins** mask, your XML editor reported that the expression was not valid when it should have been.

For example, according to the XSD, this expression was not allowed:

```
<domain_participant_qos>
    <discovery_config>
        <builtin_discovery_plugins>SPDP|SEDP</builtin_discovery_plugins>
    </discovery_config>
</domain_participant_qos>
```

This issue has been fixed, and the XSD now accepts expressions containing more than one Builtin Discovery Plugin. This issue occurred only while editing XML files because of the schema. If you ran an application with the above configuration, it did not fail.

[RTI Issue ID CORE-12740]

7.6.7 Fixes Related to Vulnerabilities

Fixes related to Connext

This release fixes some potential vulnerabilities, including RTI Issue IDs CORE-12510 and CORE-12752.

Fixes related to third-party dependencies

This release fixes some potential vulnerabilities related to third-party dependencies, described below.

[Critical] Potential crash or leak of sensitive information in Core Libraries XML parser due to vulnerabilities in Expat

The Core Libraries XML parser had a third-party dependency on Expat version 2.4.4, which is known to be affected by a number of publicly disclosed vulnerabilities.

These vulnerabilities have been fixed by upgrading Expat to the latest stable version, 2.4.8. See "Third-Party Software Upgrades" in RTI Connext Core Libraries What's New.

The impact on *Connext* applications of using the previous version varied depending on your *Connext* application configuration:

- With Security (enabling RTPS protection):
-
  - Exploitable through a compromised local file system containing malicious XML/DTD files.
  - Could lead to arbitrary code execution.
  - CVSS v3.1 Score: 8.4 HIGH

  - CVSS v3.1 Vector: AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- Without Security:
-
  - Exploitable through a compromised local file system containing malicious XML/DTD files.
  - Remotely exploitable through malicious RTPS messages.
  - Could lead to arbitrary code execution.
  - CVSS v3.1 Score: 9.8 CRITICAL
  - CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

[RTI Issue ID CORE-12872]

[Critical] Potential memory corruption when using Zlib compression due to vulnerability in Zlib

The user-data compression feature in the Core Libraries had a third-party dependency on Zlib version 1.2.11, which is known to be affected by a publicly disclosed vulnerability.

This vulnerability has been fixed by upgrading Zlib to the latest stable version, 1.2.12. See "Third-Party Software Upgrades" in RTI Connext Core Libraries What's New.

The impacts on *Connext* applications of using the previous version were as follows:

- Exploitable by triggering the compression of a sample containing a malicious payload.
- The application could crash.
- CVSS v3.1 Score: 7.5 HIGH
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

[RTI Issue ID CORE-12877]

7.6.8 Fixes Related to APIs

[Major] Copy of SampleInfo::coherent_set_info field was not supported

SampleInfo::coherent_set_info was not available when using take/read operations that did not loan the samples. The **SampleInfo::coherent_set_info** field was always set to NULL when you called the take/read operations that did not loan the samples. To get the **coherent_set_info** value, you had to use the read/take operations that loan the data.

In addition, the copy constructor and assignment operator in the Traditional C++ and Modern C++ APIs did not copy the **SampleInfo::coherent_set_info** field. This field was always set to NULL; it was your responsibility to make the copy and handle memory allocation and deletion for this field.

This problem has been fixed. If you work with the C API, starting with this release you will have to use the following functions to manipulate SampleInfo structures:

- DDS_SampleInfo_initialize()

- DDS_SampleInfo_copy()
- DDS_SampleInfo_finalize()

[RTI Issue ID CORE-11213]

[Major] Corruption of LoanedDynamicData object when moved in some situations (Modern C++ API only)

Given a DynamicData sample, accessing a nested member within another nested member via **loan_value()** and then moving the latter may have corrupted the former. For example, given a sample such that "my_sample.a.b" is a member of a constructed type (struct or union):

```
DynamicData my_sample(my_dynamic_type);
LoanedDynamicData loan1 = my_sample.loan_value(""a"");
LoanedDynamicData loan2 = loan1.get().loan_value(""b"");
// The following corrupts loan2
LoanedDynamicData loan1_moved = std::move(loan1);
```

This may have affected applications that explicitly move-constructed a double-nested LoanedDynamicData or that otherwise indirectly called the move constructor in this situation (for example, by resizing a std::vector of LoanedDynamicData elements).

The LoanedDynamicData's move constructor and move-assignment operators have been fixed.

[RTI Issue ID CORE-12272]

[Major] Calling DynamicData::set_complex_member with an aliased type failed

Calling **DynamicData::set_complex_member()** with an aliased type failed. For example, given the following types:

```
struct Foo {
long x;
long y;
};
typedef Foo TypedefFoo;
struct MyType {
Foo my_inner_struct;
TypedefFoo my_typedef_struct;
};
```

The following code should have worked to set the my_typedef_struct member:

```
struct Foo {
long x;
long y;
};
typedef Foo TypedefFoo;
struct MyType {
Foo my_inner_struct;
TypedefFoo my_typedef_struct;
};
DDS_DynamicData *data = DDS_DynamicData_new(
    MyType_get_typecode(),
    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
    DDS_DynamicData *inner_data = DDS_DynamicData_new(
    TypedefFoo_get_typecode(),
    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);

// This call fails. If the above call used Foo_get_typecode instead then it␣
↪would work

retcode = DDS_DynamicData_set_complex_member(data, ""my_typedef_struct"", 0,␣
↪inner_data);
if (retcode != DDS_RETCODE_OK) {
    fprintf(stderr, ""_set_complex_member %d\n"", retcode);
    return -1;
}
```

But instead, it failed with these errors:

```
DDS_DynamicData2_copy: Objects have different types. self type = TypedefFoo,␣
↪other type = TypedefFoo
DDS_DynamicData2_finalize_ex: finalizing object bound to a member,␣
↪automatically unbinding now.
DDS_DynamicData2_set_complex_member:ERROR: Failed to copy value
DDS_DynamicData2_unbind_complex_member:ERROR: Bad parameter: self has no␣
↪bound member
DDS_DynamicData2_set_complex_member:!unbind complex member
```

This issue has been fixed. Now, using either the aliased type (TypedefFoo in our example) or the original type (Foo in our example) works to set a complex member using the DynamicData API.

[RTI Issue ID CORE-12273]

[Major] Possible wrong results when adding Time or Duration objects that used very large numbers

Adding Time or Duration objects could have previously produced wrong results when using very large numbers. Necessary checks are now in place to ensure that wrong results do not occur.

[RTI Issue ID CORE-12413]

[Major] Java API did not support RtpsReliableReaderProtocol_t.receive_window_size

This QoS setting was ignored by the Java API, and readers were always created with the default value (256). This problem has been resolved.

[RTI Issue ID CORE-12451]

[Minor] Input parameters to Property and DataTag helper functions do not have "const"

In the C API, the following functions were incorrectly missing a **const** before the **policy** parameter:

- DDS_PropertyQosPolicyHelper_lookup_property()
- DDS_PropertyQosPolicyHelper_lookup_property_with_prefix()
- DDS_PropertyQosPolicyHelper_get_properties()
- DDS_DataTagQosPolicyHelper_lookup_tag()

This problem has been fixed. The policies are now "const" because these functions do not change the policy.

[RTI Issue ID CORE-3166]

[Minor] Standard 64-bit integer types are now supported (Modern C++ API)

Previous releases of the Modern C++ API had platform-specific definitions for 64-bit integers, defined in **rti::core::int64** and **rti::core::uint64**. This was required to support certain pre-C++11 platforms.

This release redefines those two types as **std::int64_t** and **std::uint64_t**.

[RTI Issue ID CORE-10913]

[Minor] Assigning DataWriter and DataReaderQos from a TopicQos caused a build error

DataWriterQos and DataReaderQos could not be constructed from a TopicQos assignment. You may have seen a compiler error such as:

```
error: conversion from 'TEntityQos<rti::topic::qos::TopicQosImpl>'  to
non-scalar type 'TEntityQos<rti::pub::qos::DataWriterQosImpl>' requested.
```

This problem has been resolved. Now this type of assignment works correctly. Any fields that are not in the TopicQos will use the default for the DataWriterQos or DataReaderQos.

[RTI Issue ID CORE-11185]

[Minor] In XML-based applications, generated IDL types did not take precedence over XML DynamicTypes (C# API)

In the C# API in previous releases, if a type was declared in XML as a dynamic type and also generated and registered by the application, the XML dynamic type took precedence. This led to the DataReaders or DataWriters using DynamicData instead of the generated C# user class. This behavior was unintuitive and inconsistent with the other language APIs. It has been resolved.

[RTI Issue ID CORE-11389]

[Minor] Namespaces ignored when a type was explicitly registered in C# for XML-based applications

When a type was explicitly registered (this is only necessary to support generated IDL types with XML-Based Application Creation) as follows:

```
DomainParticipantFactory.RegisterType<A.B.Foo>()
```

The registered type name was to set to “Foo” instead of the expected “A::B::Foo”. In some situations, this may have stopped applications written in other languages to communicate with a C# application, if the regular algorithm of type matching was disabled.

[RTI Issue ID CORE-12074]

7.6.9 Fixes Related to Crashes

[Critical] DataReader C++ application crashed if it received tampered sample with unsupported encapsulation ID

If a C++ application with a *DataReader* received a sample with a tampered or malformed encapsulation kind, a segmentation fault occurred when the *DataReader* attempted to deserialize the sample, leading to an application crash. This problem has been fixed.

[RTI Issue ID CORE-12356]

[Critical] Segmentation fault after calling DomainParticipant::register_durable_subscription with a group containing a long role_name

An application using the API **DomainParticipant::register_durable_subscription**() may have experienced a segmentation fault if the **role_name** of the input group was NULL or had a length greater than 512 bytes. This problem has been fixed.

[RTI Issue ID CORE-12460]

[Critical] Segmentation fault when application using MultiChannel ran out of memory

A *Connext* application using MultiChannel might have produced a segmentation fault if the system ran out of memory. This problem has been fixed.

[RTI Issue ID CORE-12493]

[Critical] Application crashed when capturing traffic for a DomainParticipant created before enabling network capture

To capture network traffic, you must enable this feature before creating the *DomainParticipants* that will capture the traffic. Applications not satisfying this requirement crashed when starting, pausing, or resuming the capture.

This problem has been fixed. *Connext* will no longer crash in this situation, but will fail and log messages such as the following:

```
ERROR NDDS_Utility_start_network_capture_w_params_for_participant:!get␣
↪network capture manager for DomainParticipant. Network capture must be␣
↪enabled before creating the DomainParticipant

ERROR NDDS_Utility_start_network_capture_for_participant:!network capture␣
↪could not be started for the participant

ERROR NDDS_Utility_run_network_capture_operation_for_all_participants:!failed␣
↪to run network capture operation for participant

ERROR NDDS_Utility_start_network_capture_w_params:!error starting network␣
↪capture for all participants

ERROR NDDS_Utility_start_network_capture:!start network capture for all␣
↪participants. There was at least one participant that could not be started
```

[RTI Issue ID CORE-12511]

[Critical] Possible crash when writing a sample

Due to an internal error, an application could crash when writing a sample using either a best-effort or reliable *DataWriter*. Before the crash, an error message in either of the following functions was printed:

```
* COMMENDBeWriterService_write

* COMMENDSrWriterService_write
```

This problem has been resolved.

[RTI Issue ID CORE-12561]

[Critical] Potential crash during type registration if system ran out of memory

A crash may have occurred during type registration if the application ran out of memory. This problem has been resolved.

[RTI Issue ID CORE-12734]

[Critical] Segmentation fault after calling DomainParticipant::delete_durable_subscription with a group containing a long role_name

An application using the API **DomainParticipant::register_durable_subscription()** may have experienced a segmentation fault if the **role_name** of the input group was NULL or had a length greater than 512 bytes. This problem has been fixed.

[RTI Issue ID CORE-12787]

[Critical] Potential crash or memory corruption if user application using thread-specific storage

Starting with release 6.1.0, there was an issue that could lead to a potential crash or memory corruption if the user application was using thread-specific storage.

In particular, when using Activity Context or Heap Monitoring, a race condition could have been triggered upon creating a thread with the ThreadFactory at the same time the DomainParticipantFactory instance was initialized or finalized. When this race condition was triggered, *Connext* might have overwritten the user application's thread-specific storage, leading to memory corruption or crashes.

This issue is now fixed. If the race condition that led to the issue happens in an application, the following benign warning will be logged:

```
Unexpected RTIOsapiContextSupport_g_tssKey value. This could mean that this␣
↪thread was
created at the same time you are destroying the DDSDomainParticipantFactory.
```

If that is the case, Activity Context and Heap Monitoring won't be available for that thread.

[RTI Issue ID CORE-12966]

[Minor] Simultaneous deletion of an entity by multiple threads caused a crash when using Java

When two threads deleted an entity at the same time, in Java, this may have caused a crash with the following backtrace:

```
#7 0x00007f7c630dad3b in REDAWeakReference_getReferent (reference=0x78,␣
↪slNode=0x7f7c4407f988, frOut=0x0, tableWithStartedCursor=0x7f7c6452c000) at␣
↪WeakReference.c:144

#8 0x00007f7c630d2ff3 in REDACursor_gotoWeakReference (c=0x7f7c4407f950,␣
↪fr=0x0, wr=0x78) at
Cursor.c:230
#9 0x00007f7c62d5ed46 in PRESPsService_destroyLocalEndpoint␣
↪(me=0x7f7c64367cc0, failReason=0x7f7cb0136fc0, group=0x7f7c64dbb340,␣
↪endpoint=0x7f7c644f0e88, worker=0x7f7c44015f70) at PsService.c:2130

#10 0x00007f7c62b6fc26 in PRESParticipant_destroyLocalEndpoint␣
↪(me=0x7f7c64368a00, failReason=0x7f7cb0136fc0, group=0x7f7c64dbb340,␣
↪endpoint=0x7f7c644f0e88, worker=0x7f7c44015f70) at Participant.c:5882
#11 0x00007f7c636fcc32 in DDS_DataReader_deleteI (reader=0x7f7c644f1070) at␣
↪DataReader.c:4250
#12 0x00007f7c6372667e in DDS_Subscriber_delete_datareader␣
↪(self=0x7f7c64dbb620, reader=0x7f7c644f1070) at Subscriber.c:1159

#13 0x00007f7c63daf24b in Java_com_rti_dds_subscription_SubscriberImpl_DDS_
↪1Subscriber_1delete_1datareader (env=0x7f7c781061f8, self_
↪class=0x7f7cb0137148, self=140172244792864, readerL=140172235575408) at␣
↪SubscriberImpl.c:790
```

This issue has been resolved. Now one thread will remove the entity and the other thread will throw an exception with the error code **com.rti.dds.infrastructure.RETCODE_ALREADY_DELETED**.

[RTI Issue ID CORE-10768]

7.6.10 Other Fixes

[Critical] Serialization/deserialization of non-primitive sequences and arrays for XCDR2_DATA_REPRESENTATION did not follow Extensible Types specification

The serialization/deserialization of sequences and arrays with non-primitive members for XCDR2_DATA_REPRESENTATION did not follow the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. This led to compatibility issues with other DDS implementations.

This problem has been fixed, although the new behavior is not enabled by default, in order to keep backward compatibility with previous *Connext* releases. You can configure a *DomainParticipant* to align with the specification by setting **dds.type_plugin.dheader_in_non_primitive_collections** to true in the *DomainParticipant's* PROPERTY QoS Policy for all the *DomainParticipants* created by your *Connext* applications.

[RTI Issue ID CORE-12464]

[Critical] Possible hang when using best-effort writers and asynchronous publishing

Due to an internal error, an application hung when using a best-effort writer and asynchronous publishing. Before the hang, the following error message was printed:

```
COMMENDBeWriterService_write:!retrieveJob
 This problem is now fixed.
```

[RTI Issue ID CORE-12562]

[Critical] Runtime error when using debug libraries for QNX x86 platform

When using the i86QNX6.6qcc_cpp4.7.3 debug libraries, your application may have had a runtime error and hung. This was because the debug libraries included the symbol for a math function ("**isinff**") that was discontinued in QNX 6.3.

This problem has been resolved. The debug libraries now include "**isinf**" instead, which is supported.

A full list of the math functions that were discontinued in QNX 6.3 can be found here: http://www.qnx.com/developers/docs/6.6.0.update/index.html#com.qnx.doc.neutrino.lib_ref/topic/whats_new_630.html.

Note: QNX platforms on x86 are not supported in Connext 7.0.0.

[RTI Issue ID CORE-12695]

[Critical] Pushed samples may not have been received by reliable DataReader when DataWriter published Type that supports Zero Copy transfer over shared memory

A reliable *DataReader* may not have received pushed samples from a *DataWriter* publishing a *Topic* on a type configured with the zero-copy transfer over shared memory **@transfer_mode(SHMEM_REF)**. This may have led to significant performance degradation because the *DataReader* has to continuously NACK the missing samples.

This problem only occurred when the following three conditions were true:

1. The *DataWriter* ran in a different host, or the *DataReader* did not have the builtin SHMEM transport enabled.
2. The *DataReader* used a ContentFilteredTopic, and the *DataWriter* did writer-side filtering, or the *DataReader* created TopicQueries.
3. The *DataWriter* was not configured to use an asynchronous publisher. This problem has been resolved.

[RTI Issue ID CORE-12775]

[Critical] Potential truncation of application-level acknowledgment response data

Connext enforced a wrong maximum length for application-level acknowledgment response data. Specifically, *Connext* incorrectly allowed setting the DATA_READER_RESOURCE_LIMITS QosPolicy's **max_app_ack_response_length** longer than the maximum serializable data, which resulted in the truncation of data when the length got close to 64kB.

This problem has been resolved: *Connext* now enforces a maximum length of 32kB for **max_app_ack_response_length** as part of *DataReader* QoS consistency checks, and it will log an error if you try to set **max_app_ack_response_length** longer than 32kB.

[RTI Issue ID CORE-12450]

[Critical] Potential Valgrind invalid read when logging a message or enabling heap monitoring

When activity context was enabled in logging, or when heap monitoring was enabled, a Valgrind invalid read similar to the following one may have been reported:

```
==1344490== Invalid read of size 4
==1344490== at 0x4A3FA0A: RTIOsapiActivityContext_skipResourceGuid␣
↪(ActivityContext.c:246)
==1344490== by 0x4A417B3: RTIOsapiActivityContext_getString (ActivityContext.
↪c:820)
```

This issue has been resolved. The Valgrind invalid read error no longer appears.

[RTI Issue ID CORE-12537]

[Major] Various issues with RtpsReliableWriterProtocol_t::nack_suppression_duration

There were various issues with the **RtpsReliableWriterProtocol_t::nack_suppression_duration** QoS:

- NACKs were being incorrectly suppressed with asynchronous publishing or non-zero **min/max_nack_response_delay** if two NACK messages were received within the **nack_suppression_duration** window, even if they were NACKing for different sets of sequence numbers. The **nack_suppression_duration** is only meant to suppress NACKs with the same leading sequence number that are received within the **nack_suppression_duration** window. If two consecutive NACKs have different leading sequence numbers, this indicates that the reader is making progress and the second one should not be suppressed, regardless of the **nack_suppression_duration**. Incorrect suppression of NACKs was not an issue if **min/max_nack_response_delay was zero and PublishModeQosPolicy.kind was SYNCHRONOUS_PUBLISH_MODE_QOS.**.
- If a NACK was received and suppressed due to the **nack_suppression_duration** before the previous NACK was responded to, then the NACK that had not been responded to yet, along with all NACKs for the duration of the **nack_suppression_duration**, were incorrectly suppressed. This problem did not occur if **min/max_nack_response_delay** was zero and **PublishModeQosPolicy.kind** was **SYNCHRONOUS_PUBLISH_MODE_QOS**.
- When **PublishModeQosPolicy.kind** was **ASYNCHRONOUS_PUBLISH_MODE_QOS**, if there were no GAP messages sent in response to a NACK, the **nack_suppression_duration** had no effect and NACKs were never suppressed. (GAP messages are sent to a *DataReader* to indicate that a sample or a set of samples are not meant for that *DataReader*. This can happen, for example, because the *DataWriter* has applied writer-side filtering or no longer has those samples in its queue.)

These issues have been resolved.

[RTI Issue ID CORE-12603]

[Major] Possible error message printed during Entity disposal

Upon the disposal of an entity, an error message from a callback associated with an event may have been printed. An excerpt of what the error may have looked like this:

```
ERROR [0x01013D3F,0x79453D76,0xA3558BB2:0x00000000|REMOVE REMOTE DR␣
↪0x01013D3F,0x79453D76,0xA3558BB2:0x80000007]␣
↪OnReliableReaderActivityChangedCallback:An exception was thrown: Omg.Dds.
↪Core.DdsException: DDS operation failed:
  at Rti.Dds.NativeInterface.Helpers.ReturnCode.CheckResult(IntPtr result)

...
```

The disposal of entities has now been modified to ensure this error does not happen.

[RTI Issue ID CORE-12641]

[Major] Source IP on Spy was not correct when DataWriters with same Topic were on different machines

The source IP on Spy may not have been correct when *DataWriters* with the same Topic were on different machines. This issue has been fixed. Now the source IP is per Entity, not per Topic, and the output will look like this:

```
11:35:13 New reader from 10.200.130.20 : topic=""Example app"" type=""app""
11:35:18 New writer from 10.200.129.195 : topic=""Example app"" type=""app""
11:35:16 New writer from 10.200.130.3 : topic=""Example app"" type=""app""
11:42:58 New data from 10.200.129.195 : topic=""Example app"" type=""app""
11:42:58 New data from 10.200.130.3 : topic=""Example app"" type=""app""
11:43:00 New data from 10.200.129.195 : topic=""Example app"" type=""app""
11:43:00 New data from 10.200.130.3 : topic=""Example app"" type=""app""
```

[RTI Issue ID CORE-12169]

[Minor] Unbounded memory growth in Monitoring Library when creating and deleting endpoints

Each time a *DataWriter* or *DataReader* is created in an application that has *RTI Monitoring Library* enabled, a new instance is created in the *DataWriters* of the library. Since, by default, the maximum number of instances the *DataWriter* can handle is unlimited, and the instances of already deleted endpoints were not cleaned up automatically, unbounded memory growth was possible in the library's *DataWriters* when constantly creating and deleting endpoints in an application that had *Monitoring Library* enabled.

This problem has been fixed by setting the **writer_data_lifecycle::autopurge_disposed_instances_delay** QoS to **DDS_DURATION_ZERO** by default in the *DataWriters* for the Monitoring Library. That way, disposed instances will be instantly cleared.

[RTI Issue ID MONITOR-244]

[Minor] Unexpected behavior when two threads crashed at the same time on Windows systems

When two threads crashed at the same time on Windows systems, *Connext* may have concurrently called the function **SymInitialize()** from **DbgHelp** from two crashing threads.

SymInitialize() is not thread safe, so the application may have run into unexpected behavior or memory corruption under this scenario.

This has been resolved, *Connext* no longer calls **SymInitialize()** from a crashing thread. Instead, **SymInitialize()** is now called during DomainParticipantFactory initialization.

[RTI Issue ID CORE-10066]

[Minor] DataReaders setting reader_qos.protocol.expects_inline_qos to TRUE incorrectly matched with DataWriters

Connext DataWriters matched *DataReaders* that set **reader_qos.protocol.expects_inline_qos** to TRUE. This behavior was incorrect because *Connext DataWriters* do not support sending inline QoS, and they were not honoring the *DataReaders*' requests.

This issue has been fixed. The behavior has changed so that *DataWriters* will not match *DataReaders* that request inline QoS (i.e., that set **reader_qos.protocol.expects_inline_qos** to TRUE).

[RTI Issue ID CORE-10501]

[Minor] Writer using durable writer history may not have blocked after send window filled up when disable positive ACKs was enabled

In previous releases, a reliable *DataWriter* configuring a finite send window size may not have blocked when the send window filled up if all these conditions were met:

- *DataWriter* was configured to use durable writer history.
- *DataWriter* was configured to use disable positive ACKs.
- *DataWriter* was configured with **writer_qos.reliability.acknowledgment_kind** set to AUTO or EXPLICIT, or **writer_qos.availability.enable_required_subscriptions** was set to TRUE.

Because of this issue, the reliability protocol for the *DataWriter* may have been less efficient. This problem has been resolved.

[RTI Issue ID CORE-12225]

[Trivial] Error messages displayed that should not have been, when printing DataReaderQoS objects

When printing DataReaderQoS objects, and the contained DDSOwnershipQosPolicy or DDS_TransportMulticastQosPolicy policies were printed, some error messages were displayed that should not have been. These error messages could have been safely ignored by an application. These error messages are no longer printed.

[RTI Issue ID CORE-12462]

[Trivial] Unnecessary sockets created during initialization of library

The initialization of the *Connext* libraries always created a socket to obtain the IP address of the first valid interface of the machine. This IP address is used to generate identifiers when **DDS_DomainParticipantQos::wire_protocol::rtps_auto_id_kind** is **DDS_RTPS_AUTO_ID_FROM_IP**, which is not the default value. Therefore, the creation of this socket was unnecessary in most cases. This problem has been solved, and now the socket is only created when it is needed.

[RTI Issue ID CORE-12587]

[Trivial] Malformed IDL printed if multiple labels used for default case of a union

The IDL produced by the C API's **DDS_TypeCode_print_IDL()** function (or the equivalent in other APIs) may have been malformed if multiple labels were assigned to the default case of a union. All of the labels were printed as "default: ", instead of their true value. This problem has been resolved.

[RTI Issue ID CORE-12624]

Chapter 8

Known Issues

Note: For an updated list of critical known issues, see the Critical Issues List on the RTI Customer Portal at https://support.rti.com.

This section includes:

8.1 Known Issues with Discovery (SPDP2)

The following known issues apply to the Simple Participant Discovery Protocol 2.0, which is an alternative version of the Simple Participant Discovery Protocol, designed for decreased bandwidth usage and improved reliability. See Simple Participant Discovery 2.0, in the RTI Connext Core Libraries User's Manual for more information.

8.1.1 Features under future consideration for SPDP2

Note: RTI does not guarantee the following features for any release or timeline. If any of these enhancements is of interest to you, please provide that feedback through your account team.

The following features, which are not currently supported, are being considered for SPDP2 in future releases:

- Use of SPDP2 with custom security plugins (for example, those implemented with the SECURITY PLUGINS SDK *(RTI Security Plugins SDK)*), the *Lightweight Builtin Security Plugins*, or *Builtin Security Plugins*'s *HMAC-Only mode*. Only the *Builtin Security Plugins* are supported in combination with SPDP2.
- SPDP and SPDP2 compatibility mode. The compatibility mode will allow some *DomainParticipants* to simultaneously communicate with *DomainParticipants* that are using SPDP and SPDP2. *DomainParticipants* that are using the compatibility mode will be able to communicate with *DomainParticipants* that are using SPDP and other *DomainParticipants* that are using SPDP2. For now, you can use *RTI Routing Service* to achieve this communication; see this Knowledge Base article on the RTI Community Forum.

- Improved configuration update behavior. Currently, when a *DomainParticipant* changes its configuration (partition, locators, etc.), it sends out:
  - If SPDP is enabled: a single Data(p) to all peers (matched or potential).
  - If SPDP2 is enabled: a single reliable message to matched peers, a single bootstrap message to unmatched initial peers. RTI will add an option to send multiple Data(p)s/bootstrap messages, since these messages are sent best-effort and can get lost, delaying configuration change updates in remote participants until the next periodic message.

[RTI Issue IDs CORE-12929, CORE-13884, and CORE-12930]

8.1.2 Participants using SPDP2 and allow_unauthenticated_participants fail to communicate if only one participant fails authentication

DomainParticipants using SPDP2 with **allow_unauthenticated_participants** set to TRUE fail to communicate if both participants are using security and only one participant fails authentication.

[RTI Issue ID SEC-2348]

8.2 Known Issues with Serialization and Deserialization

8.2.1 Some parameters cannot be received multiple times within same SPDP sample

The OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5 allows in general that "The ParameterList may contain multiple Parameters with the same value for the parameterId." *RTI Connext*, however, does not support receiving the following parameterId values multiple times within the same Simple Participant Discovery Protocol (SPDP) discovery sample:

- PID_USER_DATA
- PID_PROPERTY_LIST
- PID_ENTITY_NAME
- PID_ROLE_NAME
- PID_PARTITION
- PID_DOMAIN_TAG
- PID_IDENTITY_TOKEN
- PID_PERMISSIONS_TOKEN
- PID_TRANSPORT_INFO_LIST

[RTI Issue ID CORE-13680]

8.2.2 Connext not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types

By default, *Connext* is not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types. To configure *Connext* to be compliant, you have a few options, described in Extended CDR (encoding version 2), in the *Extensible Types Guide*.

8.3 Known Issues with Usability

8.3.1 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]

8.3.2 DataWriter's Listener callback on_application_acknowledgment() not triggered by late-joining DataReaders

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

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

[RTI Issue ID CORE-5181]

8.3.3 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* 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]

8.3.4 Memory leak if Foo:initialize() called twice

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

[RTI Issue ID CORE-7678]

8.3.5 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]

8.3.6 Type Consistency enforcement disabled for structs with more than 10000 members

TypeObjects cannot be created from structs with more than 10000 members. Applications that publish or subscribe to such types may see errors like the following:

```
RTICdrStream_serializeNonPrimitiveSequence:sequence length (10005) exceeds␣
↪maximum (10000)
RTICdrTypeObjectTypeLibraryElement_getTypeId:serialization error: Type
RTICdrTypeObject_fillType:!get TypeId
RTICdrTypeObject_assertTypeFromTypeCode:!create Structure Type
RTICdrTypeObject_createFromTypeCode:!create TypeObject
```

When the TypeObject can't be serialized, the type compatibility check between a reader and a writer falls back to exact type-name matching.

See the section "Verifying Type Consistency: Type Assignability" in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

[RTI Issue ID CORE-8158]

8.3.7 Escaping special characters in regular/filter expressions not supported in some cases

Escaping special characters is not supported in expressions when using the following features:

- Partitions
- MultiChannel

Every occurrence of a backslash (`\`) will be considered its own character and not a way to escape the character that follows. For example: `A\?` does not match `A?` because the first expression is considered an expression with three characters.

[RTI Issue ID CORE-11858]

8.4 Known Issues with Code Generation

8.4.1 Examples and generated code for Visual Studio 2017 and later may not compile (Error MSB8036)

The examples provided with *Connext* and the code generated for Visual Studio 2017 and later will not compile out of the box if the Windows SDK version installed is not a specific number like 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.)

For further details, see the Windows chapter of the *Core Libraries Platform Notes*.

[RTI Issue ID CODEGENII-800]

8.5 Known Issues with Instance Lifecycle

8.5.1 RECOVER_INSTANCE_STATE_CONSISTENCY setting not fully supported by RTI Infrastructure Services

The RECOVER_INSTANCE_STATE_CONSISTENCY option in the **instance_state_consistency_kind** field, in the RELIABILITY QoS policy, is not fully supported by the *RTI Infrastructure Services* products.

RTI Routing Service inputs cannot route instance state transitions from NOT_ALIVE_NO_WRITERS to ALIVE after regaining liveliness with a *DataWriter*. However, a *Routing Service* output *DataWriter* can be configured to use the RECOVER_INSTANCE_STATE_CONSISTENCY setting and respond to matching *DataReaders* if they request instance state updates after a reconnection.

Persistence Service, *Queuing Service*, *Recording Service*, and *Replay Service* do not support being configured with the RECOVER_INSTANCE_STATE_CONSISTENCY setting, since they do not support storing or publishing ALIVE instance state transitions with no associated data.

[RTI Issue ID CORE-13337]

8.5.2 Persistence Service DataReaders ignore serialized key propagated with dispose updates

Persistence Service DataReaders ignore the serialized key propagated with dispose updates. *Persistence Service DataWriters* cannot propagate the serialized key with dispose, and therefore ignore the **serialize_key_with_dispose** setting on the *DataWriter* QoS.

[RTI Issue ID PERSISTENCE-221]

8.5.3 instance_state_consistency_kind QoS cannot be modified before containing entity is enabled

The **instance_state_consistency_kind** field in the RELIABILITY QoS policy cannot be modified once the containing DDS entity is created, even if the containing entity is created disabled. Trying to modify the QoS setting in this case will result in the `set_qos` operation returning `DDS_RETCODE_IMMUTABLE_POLICY` and an error message being logged.

[RTI Issue ID CORE-13349]

8.6 Known Issues with Reliability

8.6.1 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]

8.7 Known Issues with Content Filters and Query Conditions

8.7.1 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]

8.7.2 filter_sample_* statistics in DDS_DataWriterProtocolStatus not updated correctly

The **filter_sample_*** statistics in the **DDS_DataWriterProtocolStatus** are not updated correctly. The values that you get after calling the following APIs may be smaller than the actual values:

- DDS_DataWriter::get_datawriter_protocol_status
- DDS_DataWriter::get_matched_subscription_datawriter_protocol_status
- DDS_DataWriter::get_matched_subscription_datawriter_protocol_status_by_locator

[RTI Issue ID CORE-5157]

8.8 Known Issues with TopicQueries

8.8.1 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]

8.9 Known Issues with Transports

8.9.1 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* 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 the "Application Acknowledgment" section in the *Core Libraries User's Manual*.

[RTI Issue ID CORE-5329]

8.9.2 DataReader cannot persist AppAck messages greater than 32767 bytes

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

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

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

[RTI Issue ID CORE-5360]

8.9.3 Discovery with Connext Micro fails when shared memory transport enabled

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

[RTI Issue ID EDDY-1615]

8.9.4 Communication may not be reestablished in some IP mobility scenarios

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

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

[RTI Issue ID CORE-8260]

8.9.5 Corrupted samples may be forwarded through Routing Service when using Zero-Copy transfer over shared memory

When using *Zero Copy transfer over shared memory* together with *RTI Routing Service*, *Routing Service* avoids an additional copy of the data by passing a reference to the sample from the input to the output of a route. If the sample is reused and rewritten by the original application *DataWriter* during the time between when the sample was received on the route input and copied into the route output buffer, the forwarded sample will contain the updated, and now invalid, values for the original sample.

This situation can be avoided in a few different ways, with various tradeoffs.

Use automatic application acknowledgment

Using automatic application acknowledgment (**acknowledgment_mode** = APPLICATION_AUTO_ACKNOWLEDGMENT in the Reliability QoS Policy) between the *Routing Service* input *DataReader* and its matching *DataWriters* will avoid the issue.

When using Zero Copy transfer over shared memory, *DataWriters* must loan samples using the **get_loan** API. Only samples that have been fully acknowledged will be returned by the **get_loan** API. This means that if automatic application acknowledgment is turned on, that only samples that the *Routing Service* has already copied and written to the route output will be available for reuse by the original *DataWriter*, because *Routing Service* does not return the loan on a sample until after it is forwarded to the route outputs.

The drawback to this approach is that it requires RELIABLE Reliability. In addition, application-level acknowledgments are not supported in *Connext Micro*, so this approach will not work if *Connext Micro* is the source of the Zero Copy samples.

Ensure that the number of available samples accounts for Routing Service processing time

Regardless of whether you are using *Routing Service*, it is important when using Zero Copy transfer over shared memory to size your resources so that your application can continue to write at the desired rate while the receiving applications receive and process the samples. If you are using *Routing Service* and cannot, or do not wish to, use automatic application acknowledgments, you must take into account the amount of time it will take to receive and forward a sample when setting **writer_loaned_sample_allocation** in the DATA_WRITER_RESOURCE_LIMITS QoS Policy and managing the samples in your application.

[RTI Issue ID CORE-10050]

8.9.6 Network Capture does not support frames larger than 65535 bytes

Network capture does not support frames larger than 65535 bytes. This limitation affects the TCP transport protocol if the **message_size_max** property is set to a value larger than the default one.

[RTI Issue ID CORE-11083]

8.9.7 Shared memory transport in QNX 7.0 and earlier can result in priority inversion

When using QNX 7.0 and earlier with *Connext*, the shared memory transport uses a kind of mutex that does not support priority inheritance. For this reason, using the shared memory transport in QNX can result in priority inversion. Starting with QNX 7.1, priority inheritance is applied to the mutex for the shared memory transport, as described in the fix for *[Minor] QNX applications using shared-memory transport may have led to thread priority inversion issues*.

[RTI Issue ID CORE-13745]

8.9.8 Ungracefully terminated QNX processes using SHMEM transport prevents startup of new processes due to unclosed POSIX semaphores (QNX 7.0 and earlier)

If a QNX 7.0 or earlier application using the shared-memory transport was ungracefully shut down, crashed, or otherwise had an abnormal termination while holding a POSIX semaphore used by the transport (for example, while sending data through the shared-memory transport), *Connext* applications launched after that point on the same domain may wait forever for that semaphore to be released.

Workaround for QNX 7.0 and earlier: to enable new applications to start, RTI recommends stopping all applications, then cleaning up the Inter-Process Communication (IPC) resources before starting new applications.

This problem is resolved for QNX 7.1, as described in the fix for *[Critical] Ungracefully terminated QNX processes using SHMEM transport prevented startup of new processes due to unclosed POSIX semaphores*.

[RTI Issue ID CORE-9434]

8.10 Known Issues with FlatData

8.10.1 FlatData language bindings do not support automatic initialization of arrays of primitive values to non-zero default values

RTI FlatData™ language bindings do not support the automatic initialization of arrays of primitive values to non-zero default values, unless the primitive is an enumeration. It is possible to declare an alias to a primitive member with a default value using the @default annotation, and then to declare an array of that alias. For example:

```
@default(10)
typedef int32 myLongAlias;

struct MyType {
    myLongAlias myLongArray[25];
};
```

The default values of each member of the array in this case should be 10, but in FlatData they will all be set to 0.

[RTI Issue ID CORE-9986]

8.10.2 Flat Data: plain_cast on types with 64-bit integers may cause undefined behavior

The function **rti::flat::plain_cast** is allowed on FlatData samples containing int64_t members, but those members are not guaranteed to have an 8-byte alignment (a 4-byte alignment is guaranteed). Memory checkers such as Valgrind may report errors when accessing such members from the pointer returned by **plain_cast**.

[RTI Issue ID CORE-10092]

8.11 Known Issues with Coherent Sets

8.11.1 Some coherent sets may be lost or reported as incomplete with batching configurations

If *Connext* 6.1.0 receives coherent sets from *Connext* 6.0.0 or lower using batching, coherent sets that are fully received and complete may be lost or marked as incomplete. (If the QoS **subscriber_qos.presentation.drop_incomplete_coherent_set** is set to FALSE, then the samples marked as incomplete won't be dropped.)

[RTI Issue ID CORE-9691]

8.11.2 Copy of SampleInfo::coherent_set_info field is not supported

SampleInfo::coherent_set_info is not available when using take/read operations that do not loan the samples. The **SampleInfo::coherent_set_info** is always set to NULL when you call the take/read operations that do not loan the samples. To get the **coherent_set_info** value, make sure you use the read/take operations that loan the data.

In addition, the copy constructor and assignment operator in the Traditional C++ and Modern C++ APIs do not copy the **SampleInfo::coherent_set_info** field. It is always set to NULL. It is your responsibility to make the copy and handle memory allocation and deletion for this field.

[RTI Issue ID CORE-11215]

8.11.3 Other known issues with coherent sets

Coherent sets are not propagated through *RTI Routing Service* [RTI Issue ID ROUTING-657].

Group coherent sets are not persisted by *RTI Persistence Service* [RTI Issue ID PERSISTENCE-191].

Group coherent sets cannot be stored or replayed with *RTI Recording Service* [RTI Issue ID RECORD-1083].

8.12 Known Issues with Dynamic Data

8.12.1 Conversion of data by member-access primitives limited when converting to types that are not supported on all platforms

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 int64 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 int64s from a 32-bit or smaller integer type is supported on all Windows and Linux architectures, and any additional 64-bit architectures. Conversion to 128-bit long doubles from a float or double is not supported.

[RTI Issue ID CORE-2986]

8.12.2 Types that contain bit fields not supported

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-2977]

8.12.3 Long double not supported for DynamicData in Java API

The Java API does not have DynamicData APIs to handle long double fields. That is, `get_longdouble` and `set_longdouble` don't exist. As a result, it's not possible to work with long double fields of DynamicData samples in the Java API.

[RTI Issue ID CORE-14091]

8.12.4 Limitation for C# recursive types

Recursive types (types whose definition contains themselves at any level) are supported in the C# API. However, there is one limitation. The DynamicType property of the type support generated for a recursive IDL type (such as `FooTypeSupport.Instance.DynamicType`) is not available. Trying to access it will fail with `NotSupportedException`. This property is only needed for applications that inspect the type dynamically or create DynamicData objects. If that is required, you can define the type in XML and load it with `QosProvider.GetType` or in code using the DynamicTypeFactory.

[RTI Issue ID CORE-14407]

8.13 Known Issues with Logging

8.13.1 Possible crash when closing a logger device while it is used

Due to a concurrency issue in the *Connext* logging infrastructure, there is a small possibility of a crash in an application when a logger device is closed at the same time it is being used for logging a message.

[RTI Issue ID CORE-10546]

8.14 Known Issues with RTI Monitoring Library

The following known issues occur in *RTI Monitoring Library*, not in *RTI Monitoring Library 2.0*.

8.14.1 Problems with NDDS_Transport_Support_set_builtin_transport_property() if Participant Sends Monitoring Data

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

[RTI Issue ID MONITOR-222]

8.14.2 Participant's CPU and memory statistics are per application

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

[RTI Issue ID CORE-7972]

8.14.3 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, *Monitoring library* will fail.

[RTI Issue ID MONITOR-220]

8.15 Other Known Issues

8.15.1 Possible Valgrind still-reachable leaks when loading dynamic libraries

If you load any dynamic libraries, you may see "still reachable" memory leaks in "dlopen" and "dlclose". These leaks are a result of a bug in Valgrind (https://bugs.launchpad.net/ubuntu/+source/valgrind/+bug/1160352).

This issue affects the *Core Libraries*, SECURITY PLUGINS, and *TLS Support*.

[RTI Issue IDs CORE-9941, SEC-1026, and COREPLG-510]

8.15.2 64-bit discriminator values greater than (2^31-1) or smaller than (-2^31) not supported

Unions with a 64-bit integer discriminator type containing discriminator values that cannot fit in a 32-bit value are not supported when using the following language bindings:

- C
- Traditional C++
- Modern C++
- C#
- Java
- Python
- DynamicData (regardless of the language)

They are also not supported with ContentFilteredTopics, regardless of the language binding.

Using label values greater than 32-bit may lead to receiving samples with invalid content or to filtering samples incorrectly.

[RTI Issue ID CORE-11437]

8.15.3 Creating multiple DataReaders for the same Topic under the same Subscriber configured with Group Ordered Access is not supported

Creating multiple *DataReaders* for the same *Topic* under the same *Subscriber* configured with PresentationQosPolicy **access_scope** = GROUP and **ordered_access** = TRUE is not supported. If you try to create a second reader in this situation, it will fail to be created and this error will be printed:

```
ERROR [0x0101E967,0x5C3A43B1,0x99D71EB7:0x80000309{Entity=Su,Domain=0}|CREATE␣
↪DR WITH TOPIC FooTopic|LC:Discovery]PRESPsService_createLocalEndpoint:NOT␣
↪SUPPORTED | Creating more than one reader for the same topic within a␣
↪single subscriber using GROUP presentation and ordered_access=true.
```

Instead, in this situation, you will need to use only one *DataReader*, or you will need to create a new *Subscriber* and *DataReader* in the same *DomainParticipant*.

[RTI Issue ID CORE-12448]

8.15.4 With DISALLOW_TYPE_COERCION and Types containing unbounded members, other vendor DataWriters/DataReaders will not match Connext DataWriters/DataReaders

When **reader_qos.type_consistency.kind** is set to DISALLOW_TYPE_COERCION, a *Connext DataReader/DataWriter* for a type containing unbounded collection members will not match with a *DataReader/DataWriter* from another vendor.

A possible workaround is to use ALLOW_TYPE_COERCION or disable type validation by not sending TypeObject information.

[RTI Issue ID CORE-14367]

Chapter 9

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>>**.

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 features may be only available in a subset of the supported languages and for a subset of the supported platforms.
- Experimental features may change in the future.
- 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/).