RTI Data Distribution Service

The Real-Time Publish-Subscribe Middleware

Release Notes

Version 4.5c

Printed in U.S.A. First printing.
June 2010.

Trademarks

Copy and Use Restrictions

Technical Support

Real-Time Innovations, Inc.
385 Moffett Park Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: http://www.rti.com/support

Contents

1 System Requirements 2

1.1 Supported Operating Systems 2

1.2 Platforms Removed from RTI Data Distribution Service 4.5b 5

1.3 Disk and Memory Usage 5

1.4 Networking Support 5

2 Compatibility 14

2.1 Wire Protocol Compatibility 14

2.1.1 General Information on RTPS (All Releases) 14

2.1.2 Release-Specific Information for RTI Data Distribution Service 4.5 14

2.2 Code Compatibility 16

2.2.1 General Information (All Releases) 16

2.2.2 Release-Specific Information for RTI Data Distribution Service 4.5 16

2.3 Data-Format Compatibility and Extensibility 20

2.4 ODBC Database Compatibility 21

2.5 Build Compatibility 22

2.5.1 Windows C/C++ Build Compatibility 22

3 What's Fixed in 4.5c 23

3.1 Fixes Related to Reliability 23

3.1.1 DataWriter May Not Have Properly Switched between Normal and Fast Heartbeat Modes 23

3.1.2 Reliable DataWriters did not Repair Requested Data Fragments 23

3.1.3 wait_for_historical_data Returned Before All Historical Data Received 23

3.2 Fixes Related to Content Filtering 24

3.2.1 DataWriter May Have Crashed if Writer-Side Content Filtering Enabled 24

3.2.2 Statistics on Filtered Samples not Updated for Best-effort Writer and Reader-side Filtering 24

3.2.3 Prematurely Removed Destination for a Reliable DataWriter with Writer-side Content Filtering 24

3.2.4 Memory Leak when Using Writer-side Filtering with CORBA/C++ or Dynamic Data 24
3.2.5 Asynchronous Publisher Unnecessarily Evaluated Content Filter 25
3.3 Fixes Related to QueryConditions and ReadConditions 25
3.3.1 Performance Problems when Using ReadConditions 25
3.3.2 Potential Incorrect Behavior if New QueryCondition Created After Deleting Previous QueryCondition 25
3.3.3 Possible Segmentation Fault when QueryCondition Deleted 25
3.4 Fixes Related to Java 25
3.4.1 DataReader Crashed if DataWriter disable_inline_keyhash Enabled —Java Only 25
3.4.2 Null Pointer Exception if No Type Code Sent when Using ContentFilter —Java Only 26
3.4.3 Setting Null Listener with Non-Empty Mask should be Allowed—Java Only 26
3.4.4 Memory Leak when using DynamicData Sequence with Read/Take API —Java Only 26
3.4.5 Errors When Deleting DomainParticipant After Unregistering Built-in Content Filters—Java Only 26
3.4.6 Creation of DataWriters and DataReaders Took a Long Time with Complex IDL Types with Sequences and Arrays—Java Only 27
3.4.7 DataReader Creation Slow for Topics with Union Type if Discriminator is Enumeration—Java Only 27
3.5 Fixes Related to rtiddsgen 27
3.5.1 Incorrect Output Format from TypeSupport_print_data() for Character Data Type 27
3.5.2 Possible Exception from TypeSupport_print_data() when printing Character Data Type—with Debug Libraries on Windows Systems Only 28
3.5.3 Error from rtiddsgen If No Preprocessor Specified—Windows Systems Only 28
3.5.4 Incorrect Redefinition Compilation Error if const with Same Name was Defined Twice in Different Scope 28
3.5.5 Compilation Errors with Types Contained in Module with Same Name 29
3.5.6 Incorrect Parsing of Keys in Typedefs by rtiddsgen 29
3.5.7 Improper Constructor in Generated TypeSupport Code for Value Types whose Keys are Only in Base Type—Java Only 30
3.5.8 Compilation Error when using rtiddsgen -use42eAlignment—Java Only 30
3.5.9 Possible Failure in rtiddsgen if Path to IDL File Contained Whitespace 30

3.6 Fixes Related to .NET 30
3.6.1 Incorrect Content Filter Information in SubscriptionBuiltinTopicData —.NET Only 30
3.6.2 Extraneous 'Lost Data' Calls for Best-Effort, Content-Filtered Readers —.NET Only 31
3.7 Fixes Related to Statuses and Listeners 31
3.7.1 Incorrect DataReaderProtocolStatus with Matched Publication 31
3.7.2 Reliable DataReader duplicate_sample_count not Updated 31
3.7.3 Warning and Incorrect Statistics if max_gather_destinations Exceeded 31
3.7.4 NULL Listener with INCONSISTENT_TOPIC_STATUS in DomainParticipant Listener Mask was Incorrectly Disallowed—Debug Mode Only 31
3.8 Fixes Related to QoS and Other Miscellaneous Fixes 32
3.8.1 Keyed DataReader May Have Crashed if Samples Received Out-of-Order and DestinationOrder kind Set to BY_SOURCE_TIMESTAMP 32
3.8.2 Possible Memory Corruption Due to Improper PropertyQosPolicy Consistency Check 32
3.8.3 Unregister_instance() May Crash when Given Invalid Instance Handle 32
3.8.4 Transport Priority Not Always Used by Transport Plugins 32
3.8.5 DomainParticipant Default Multicast Locator not Applied to DataReaders 32
3.8.6 Error Deleting DomainParticipant if PublishModeQosPolicy Changed After DataWriter Created 33
3.8.7 Calls to delete_contained_entities() Failed if DomainParticipant not Enabled—Debug Mode Only 33
3.8.8 Potential Crash when Deleting DataReader on Unkeyed Topic when Using Batching 33
3.8.9 Possible Crash when a Cloned TypeCode was Deleted—Only with Dynamic Data 34

4 What's Fixed in 4.5b 35
4.1 Fixes Related to Java 35
4.1.1 get_qos_profile_libraries() and get_qos_profiles() May Have Crashed if it was First Call on DomainParticipantFactory—Java only 35
4.1.2 Root Permission Required to Modify Real-Time Thread Priorities—Java Only 35
4.1.3 Real-Time Thread Priority Could Not be Set to 5—Java Only 36
4.2 Fixes Related to .NET 36
4.2.1 Crash when Copying a Sequence of Samples—.NET Only 36

4.2.2 Generated Code for copy_from() Failed to Perform Deep Copy—.NET Only 36
4.3 Fixes Related to Content Filtered Topics 36
4.3.1 DataWriter May Have Crashed when Writing to Multiple DataReaders with ContentFilteredTopic 36
4.3.2 'Serialize Key' Error when Reader Filtered Out First Sample Received for Instance—Dynamic Data, CORBA/C++, or .NET 37
4.3.3 Error Between Java and C/C++/C# Applications when Content Filter is Applied on Discriminator Member of Union Type 37
4.4 Fixes Related to Dynamic Data 37
4.4.1 Serialization Error or Data Loss after Series of 'set' Operations when Using Dynamic Data with 'long long' or 'double' Member 37
4.4.2 Serialization Error or Data Loss when Sending Dynamic Data for a Sparse Data Type in which not all Members Explicitly Initialized 37
4.4.3 Incorrect Access of String Arrays and Sequences with Maximum String Length in Dynamic Data 38
4.4.4 'Internal Error' When Using Dynamic Data on 64-bit Architectures 38
4.4.5 Inability to Access Fields of Union Data Type by Name in Dynamic Data in Certain Cases 38
4.4.6 Message may be Printed when Accessing Union Member of Dynamic Data with Default Discriminator Case 39
4.4.7 DynamicData's get_member_info() May Have Returned Incorrect Data for Unions 39
4.4.8 Dynamic Data Member Info May Have Incorrectly Returned Element Kind TK_ALIAS 39
4.4.9 Print Method Failed for Invalid Enum Data in Dynamic Data 40
4.5 Fixes Related to Batching 40
4.5.1 Reader Configured with KEEP_LAST History May Have Crashed on Receiving a Batch Sample if max_samples was Exceeded 40
4.5.2 Queue of KEEP_LAST Reliable DataReader Not Fully Utilized when Batching Enabled 40
4.5.3 Precondition Error on Write when Batching Used for Writer Enabled via Participant's or Publisher's enable() API 41
4.5.4 Resource Contention Error When Using Batching and KEEP_LAST on Writer Side 41
4.5.5 Incorrect Precondition Error on DataWriter When Using Batching with Debug Libraries 41

4.6 Fixes Related to rtiddsgen 42
4.6.1 XSD Generation Failed if IDL File Contained Constant Initialized with Enum Value 42
4.6.2 Compilation Failures in C++ Generated Code Using Namespaces if Typedef Referred to Type in Different Module 42
4.6.3 Incorrect Code Generated For Enum's with Partially Defined Enumerator Values—Java Only 43
4.6.4 Potentially Incorrect Maximum Key-Size Calculated by rtiddsgen for Types with Complex Keys—Java Only 43
4.6.5 Deserialization Error in Java/CORBA with Types Containing Enumerations 43
4.6.6 Typedef TypeCode Generated for C# was Incorrect 44
4.6.7 Incorrect CPP/CLI Code Generated by rtiddsgen if IDL Referred to Other IDL 44
4.6.8 Exception Caused by Object Creation in .NET for Types Containing Arrays of Wstrings 44
4.6.9 Incorrect Code Generated to Serialize/Deserialize Arrays of Sequences —.NET Only 44
4.6.10 Possible Incorrect Size from get_min_size_serialized()—.NET and C++/CORBA Only 44
4.6.11 Incorrect Example Code Generated for Publisher's unregister_instance() Call —C# Only 45
4.7 Fixes Related to Other Utilities 45
4.7.1 rtiddsping Reader Incorrectly Reported Missing Samples at Startup 45
4.7.2 Occasional Incomplete Discovery on Startup in rtiddsspy 45
4.8 Fixes Related to QoS and Other Miscellaneous Fixes 45
4.8.1 Reliable Reader Configured with KEEP_LAST History May Have Stopped Receiving Samples while Reader Queue Full 45
4.8.2 Possible Reader Crash if max_samples_per_instance Exceeded on Reliable Reader and Matching Writer was Shutdown 46
4.8.3 Potential Race Condition when Reader is Enabled 46
4.8.4 Problem Getting Protocol or Cache Status within Reader/Writer Listener Callbacks 46
4.8.5 Error when Accessing Cache or Protocol Status for Disabled Writer or Reader 47
4.8.6 Failure When Allocating Buffers Larger Than 4 GB 47
4.8.7 Problem Creating a Publisher/Subscriber with create_*_with_profile() after Loading Profile with is_default_qos = true 47
4.8.8 Possible Deadlock when Calling get_qos_profile_libraries() and get_qos_profiles() 47

4.8.9 Issues with XSD Schema Distributed with RTI Data Distribution Service 47
4.8.10 Possible Memory Corruption when Copying PropertyQosPolicy 48
4.8.11 Possible Segmentation Fault when Calling set_qos() Multiple Times after Setting PropertyQosPolicy to Different Values 48
4.8.12 DomainParticipantFactoryQos_copy() May Have Incorrectly Returned OK Even if ProfileQosPolicy Copy Failed 48
4.8.13 Changing Enabled DomainParticipant's QoS May Have Caused False Immutable QoS Error 48
4.8.14 Inconsistent QoS Error When Duration Set to Infinite 49
4.8.15 'nanosec' Portion of Duration May Have Been Assumed as 0 when 'sec' was 0 49
4.8.16 Locator Kinds in Builtin Topic Data May Have Been Inconsistent 49
4.8.17 Some Logging Output Sent to Console Instead of File 50

5 Known Issues 50

5.1 Estimate of Message Overhead For Large Data May Be Exceeded 50
5.2 Writer-side Filtering May Cause a Deadline to be Missed 50
5.3 Disabled Interfaces on Windows Systems 50
5.4 Wrong Error Code After Timeout on write() from Asynchronous Publisher 51
5.5 Incorrect Content Filtering for Valuetypes and Sparse Types 51
5.6 Multi-dimensional Arrays of Sequences Not Supported when Generating .NET Code 51
5.7 Best-Effort Writer-Side Content-Filtered Samples Reported As Lost 51
5.8 License File May Not be Found if NDDSHOME Set with Quotation Marks 52
5.9 Issues with Dynamic Data 52

Release Notes

This document includes the following sections:

- System Requirements (Section 1)
- Compatibility (Section 2)
- What's Fixed in 4.5c (Section 3)
- What's Fixed in 4.5b (Section 4)
- Known Issues (Section 5)

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

For more information, visit the RTI Knowledge Base, accessible from http://www.rti.com/support, to see sample code, general information on *RTI® Data Distribution Service*, performance information, troubleshooting tips, and technical details. By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send e-mail to **support@rti.com**. We can only guarantee a response to customers with a current maintenance contract or subscription. You can purchase a maintenance contract or subscription by contacting your local RTI representative (see http://www.rti.com/company/contact.html), sending an e-mail request to **sales@rti.com**, or calling +1 (408) 990-7400.

1 System Requirements

1.1 Supported Operating Systems

RTI Data Distribution Service requires a multi-threaded operating system. This section describes the host and target systems supported by *RTI Data Distribution Service*.

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

Table 1.0 lists the platforms available with *RTI Data Distribution Service* 4.5c.

Table 1.0 **Platforms Available with Release 4.5c**

Platform	Operating System	Reference
AIX®	AIX 5.3	Table 1.1 on page 6
INTEGRITY®	INTEGRITY 5.0	Table 1.2 on page 6
Linux® (Cell BE™)	Fedora™ 9	Table 1.3 on page 6
Linux (Intel®)	Fedora 10 Red Hat® Enterprise Linux 4.0, 5.0 Red Hat Linux 8.0 and 9.0 SuSE® Linux Enterprise Server 10.1 (2.6 kernel)	Table 1.4 on page 7
Linux (PowerPC®)	SELinux (2.6.27.14) Yellow Dog™ Linux 4.0	Table 1.5 on page 8
LynxOS®	LynxOS 4.0, LynxOS 4.2, LynxOS 5.0, LynxOS-SE 3.0,	Table 1.6 on page 8
Mac OS®	Mac OS X	Table 1.7 on page 9
Solaris™	Solaris 2.9 and 2.10	Table 1.8 on page 9
VxWorks®	VxWorks 5.4.2, 5.5.1, 6.0 - 6.7	Table 1.9 on page 10

Table 1.0 **Platforms Available with Release 4.5c**

Platform	Operating System	Reference
Windows®	Windows 7 Windows 2000 with service pack 2 or higher Windows 2003 and Windows 2003 x64 Edition Windows CE 6.0 Windows Vista® Windows Server 2008 R2 Windows XP Professional and Windows XP Professional x64 Edition	Table 1.10 on page 12

Visual Studio® 2005 — Service Pack 1 Requirement

❑ You must have Visual Studio 2005 Service Pack 1 or the Microsoft Visual C++ 2005 SP1 Redistribution Package installed on the machine where you are *running* an application built with the release libraries of the following RTI architecture packages:

- x64Win64VS2005 built with dynamic libraries
- i86Win32VS2005 built with dynamic libraries
- i86Win32jdk, x64Win64jdk, i86Win32dotnet2.0, and x64Win64dotnet2.0

The Microsoft Visual C++ 2005 Service Pack 1 Redistribution Package can be downloaded from RTI's Customer Portal[1], or obtained from the following Microsoft website:

- For x86 architectures:
  http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en
- For x64 architectures:
  http://www.microsoft.com/downloads/details.aspx?familyID=EB4EBE2D-33C0-4A47-9DD4-B9A6D7BD44DA&displaylang=en

To run an application built with *debug* libraries of the above RTI architecture packages, you must have Visual Studio 2008 Service Pack 1 installed.

1. On the portal, select the Downloads page. The Redistribution Package is in the section labeled, "Windows Target Libraries for RTI Data Distribution Service."

Visual Studio 2008 - Service Pack 1 Requirement

- You must have Visual Studio 2008 Service Pack 1 or the Microsoft Visual C++ 2008 SP1 Redistribution Package installed on the machine where you are *running* an application built with the release libraries of the following RTI architecture packages:
  - x64Win64VS2008 built with dynamic libraries
  - i86Win32VS2008 built with dynamic libraries

  To run an application built with *debug* libraries of the above RTI architecture packages, you must have Visual Studio 2008 Service Pack 1 installed.

  The Microsoft Visual C++ 2008 Service Pack 1 Redistribution Package can be downloaded from RTI's Customer Portal[1], or obtained from the following Microsoft website:
  - For x86 architectures:

    http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en
  - For x64 architectures:

    http://www.microsoft.com/downloads/details.aspx?FamilyID=ba9257ca-337f-4b40-8c14-157cfdffee4e&displaylang=en

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

The following tables provide additional details. See the *RTI Data Distribution Service User's Manual* and *Platform Notes* for more information on compilers and linkers.

- Table 1.1, "AIX Platforms," on page 1-6
- Table 1.2, "INTEGRITY Platforms," on page 1-6
- Table 1.3, "Linux Platforms on Cell BE CPUs," on page 1-6
- Table 1.4, "Linux Platforms on Intel and AMD CPUs," on page 1-7
- Table 1.5, "Linux Platforms on PowerPC CPUs," on page 1-8
- Table 1.6, "LynxOS Platforms," on page 1-8
- Table 1.7, "Mac OS Platforms," on page 1-9
- Table 1.8, "Solaris Platforms," on page 1-9
- Table 1.9, "VxWorks Platforms," on page 1-10
- Table 1.10, "Windows Platforms," on page 1-12

1.2 Platforms Removed from RTI Data Distribution Service 4.5b

These platforms are no longer supported:

- ❏ Fedora 8 (ppc64Linux2.6gcc4.1.2)
- ❏ J2SE 1.4 for most platforms
- ❏ LynxOS 4.0 on i86 CPUs for gcc 2.95.3 (i86Lynx4.0.0gcc2.95.3)
- ❏ LynxOS 4.2 on PPC 604 and PPC 7XX for gcc 3.2.2 (ppc750Lynx4.2.0gcc3.2.2)
- ❏ QNX Neutrino 6.3.0
- ❏ Solaris 2.8
- ❏ Solaris 2.9 on UltraSPARC for CC 5.3 (sparcSol2.9cc5.3)

1.3 Disk and Memory Usage

Disk usage for a typical host-only installation is approximately 250 MB. Each additional architecture (host or target) requires an additional 75 MB.

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.

1.4 Networking Support

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

By default, *RTI Data Distribution Service* uses the UDP/IPv4 and shared-memory transport plug-ins. The shared memory transport is not supported for VxWorks 5.4 and 5.5.

A built-in IPv6 transport is also available (disabled by default) for these platforms:

- ❏ Linux/Fedora: all platforms except Red Hat Linux 8.0 and 9.0
- ❏ Solaris: all platforms
- ❏ Windows: all platforms except Windows CE and Visual C 6.0/7.0

A TCP transport is also available (but is not a *built-in* transport) for the following platforms:

- ❏ RedHat Enterprise Linux 4.0, 5.0, 5.1, and 5.2
- ❏ Windows with Visual Studio 2005, 2008, and 2010.

Supported architectures appear on the following pages, followed by Compatibility (Section 2).

Table 1.1 **AIX Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
AIX 5.3	POWER5 (32-bit mode)	IBM XLC for AIX v9.0	p5AIX5.3xlc9.0
		IBM Java 1.6	p5AIX5.3xlc9.0jdk
	POWER5 (64-bit mode)	IBM XLC for AIX v9.0	64p5AIX5.3xlc9.0
		IBM Java 1.6	64p5AIX5.3xlc9.0jdk

Table 1.2 **INTEGRITY Platforms**

Operating System	CPU	IP Stack[1]	RTI Architecture Abbreviation
INTEGRITY 5.0.7	PPC 74XX	InterNiche (GHnet1) TCP/IP stack	ppc7400Inty5.0.7.mvme5100-7400 [2]
		Interpeak TCP/IP stack with multicast	ppc7400Inty5.0.7.mvme5100-7400-ipk
INTEGRITY 5.0.8	PPC 74XX	InterNiche (GHnet1) TCP/IP stack	ppc7400Inty5.0.7.mvme5100-7400[2]
INTEGRITY 5.0.9, 5.0.10	PPC 74XX	GHnet2 TCP/IP stack	ppc7400Inty5.0.9.mvme5100-7400-ghnet2

1. For additional supported transports, see the online documentation or contact support@rti.com.
2. INTEGRITY 5.0.7 and 5.0.8 share the same architecture (ppc7400Inty5.0.7.mvme5100-7400).

Table 1.3 **Linux Platforms on Cell BE CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Fedora 9 (2.6 kernel)	Cell BE	ppu 4.1.1	cell64Linux2.6ppu4.1.1
		IBM Java JDK 1.5	cell64Linux2.6ppu4.1.1jdk

Table 1.4 **Linux Platforms on Intel and AMD CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Fedora 10 (2.6 kernel)	x64	gcc 4.3.2	x64Linux2.6gcc4.3.2
		IBM Java JDK 1.5	x64Linux2.6gcc4.3.2jdk
Red Hat Linux 8.0 (2.4 kernel)	Pentium class	gcc 3.2	i86Linux2.4gcc3.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2jdk
Red Hat Linux 9.0 (2.4 kernel)	Pentium class	gcc 3.2.2	i86Linux2.4gcc3.2.2[1]
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2.2jdk
Red Hat Enterprise Linux 3.0 (2.4 kernel)	Pentium class	gcc 3.2.2	i86Linux2.4gcc3.2.2[1]
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.4gcc3.2.2jdk
	AMD64	gcc 3.2.3	x64Linux2.4gcc3.2.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.4gcc3.2.3jdk
Red Hat Enterprise Linux 4.0 (2.6 kernel)	Pentium class	gcc 3.4.3	i86Linux2.6gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc3.4.3jdk
	x86_64 and AMD64	gcc 3.4.5	x64Linux2.6gcc3.4.5
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.6gcc3.4.5jdk
Red Hat Enterprise Linux 5.0 (2.6 kernel)	Pentium class	gcc 4.1.1	i86Linux2.6gcc4.1.1
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc4.1.1jdk
	x86_64 and AMD64	gcc 4.1.1	x64Linux2.6gcc4.1.1
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Linux2.6gcc4.1.1jdk
Red Hat Enterprise Linux 5.1, 5.2 (2.6 kernel)	Pentium class	gcc 4.1.2	i86Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Linux2.6gcc4.1.2jdk
	x86_64 and AMD64	gcc 4.1.2	x64Linux2.6gcc4.1.2
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Linux2.6gcc4.1.2jdk

Table 1.4 **Linux Platforms on Intel and AMD CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
SUSE Linux Enterprise Server 10.1 (2.6 kernel)	Pentium class	gcc 4.1.0	i86Suse10.1gcc4.1.0
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Suse10.1gcc4.1.0jdk
	AMD64	gcc 4.1.0	x64Suse10.1gcc4.1.0
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	x64Suse10.1gcc4.1.0jdk

1. Red Hat Linux 9.0 and Red Hat Enterprise Linux 3.0 share the same RTI architecture.

Table 1.5 **Linux Platforms on PowerPC CPUs**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
SELinux (2.6.27.14)	PowerPC440EP	gcc 4.3.3 with GNU libc 2.9	ppc4xxFPLinux2.6gcc4.3.3
Yellow Dog® Linux 4.0 (2.6 kernel) (target only)	PPC 74xx (such as 7410)	gcc 3.3.3	ppc7400Linux2.6gcc3.3.3

Table 1.6 **LynxOS Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
LynxOS 4.0	Pentium class	gcc 3.2.2	i86Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	i86Lynx4.0.0gcc3.2.2jdk
	PPC 74xx (such as 7410)	gcc 3.2.2	ppc7400Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	ppc7400Lynx4.0.0gcc3.2.2jdk
	PPC 604 PPC 7XX (such as 750)	gcc 3.2.2	ppc750Lynx4.0.0gcc3.2.2
		Sun Java Platform Standard Edition JDK 1.4	ppc750Lynx4.0.0gcc3.2.2jdk

Table 1.6 **LynxOS Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
LynxOS 4.2	Pentium class	gcc 3.2.2	i86Lynx4.2.0gcc3.2.2
	PPC 74xx (such as 7410)	gcc 3.2.2	ppc7400Lynx4.2.0gcc3.2.2
LynxOS 5.0	PPC 74xx (such as 7410)	gcc 3.4.3	ppc7400Lynx5.0.0gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.4	ppc7400Lynx5.0.0gcc3.4.3jdk
LynxOS-SE 3.0	Pentium class	gcc 3.4.3	i86LynxOS_SE3.0.0gcc3.4.3

Table 1.7 **Mac OS Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Mac OS X	x64	gcc 4.2.1	x64Darwin10gcc4.2.1
		Java SE 1.6 for Mac OS	x64Darwin10gcc4.2.1jdk

Table 1.8 **Solaris Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Solaris 2.9	Pentium class	gcc 3.3.2	i86Sol2.9gcc3.3.2
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	i86Sol2.9jdk
	UltraSPARC	CC 5.4 (Forte Dev 7, Sun One Studio 7)	sparcSol2.9cc5.4
		gcc 3.2	sparcSol2.9gcc3.2
		gcc 3.3	sparcSol2.9gcc3.3
		Sun Java Platform Standard Edition JDK 1.5 and 1.6	sparcSol2.9jdk

Table 1.8 **Solaris Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Solaris 2.10	AMD64	gcc 3.4.3	x64Sol2.10gcc3.4.3
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Sol2.10jdk
	Pentium class	gcc 3.4.4	i86Sol2.10gcc3.4.4
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Sol2.10jdk
	UltraSPARC	gcc3.4.2	sparcSol2.10gcc3.4.2
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	sparcSol2.10jdk
	UltraS-PARC (with native 64-bit support)	cc 5.8	sparc64Sol2.10cc5.8
		gcc3.4.2	sparc64Sol2.10gcc3.4.2
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	sparc64Sol2.10jdk

Table 1.9 **VxWorks Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 5.4.2	ppc604, ppc750, ppc7400	gcc 2.96	ppc604Vx5.4gcc
VxWorks 5.5.1	Pentium	gcc 2.96	pentiumVx5.5gcc
	ppc405		ppc405Vx5.5gcc
	ppc604, ppc750, ppc7400		ppc604Vx5.5gcc
	ppc603		ppc603Vx5.5gcc
VxWorks 6.0, 6.1, 6.2	Pentium	gcc 3.3.2	For kernel modules: pentiumVx6.0gcc3.3.2 For Real Time Processes: pentiumVx6.0gcc3.3.2_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.0gcc3.3.2 For Real Time Processes: ppc604Vx6.0gcc3.3.2_rtp

Table 1.9 **VxWorks Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 6.3, 6.4	Pentium	gcc 3.4.4	For kernel modules: pentiumVx6.3gcc3.4.4 For Real Time Processes: pentiumVx6.3gcc3.4.4_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.3gcc3.4.4 For Real Time Processes: ppc604Vx6.3gcc3.4.4_rtp
VxWorks 6.5	Pentium	gcc 3.4.4	For kernel modules: pentiumVx6.5gcc3.4.4 For Real Time Processes: pentiumVx6.5gcc3.4.4_rtp
	Any Wind River PPC32 CPU with floating point hardware		For kernel modules: ppc604Vx6.5gcc3.4.4 For Real Time Processes: ppc604Vx6.5gcc3.4.4_rtp
VxWorks 6.6	Pentium	gcc 4.1.2	For Kernel Modules: pentiumVx6.6gcc4.1.2 For Real Time Processes: pentiumVx6.6gcc4.1.2_rtp
	Any Wind River PPC32 CPU with floating point hardware	gcc 4.1.2	For Kernel Modules: ppc604Vx6.6gcc4.1.2 For Real Time Processes: ppc604Vx6.6gcc4.1.2_rtp
	ppc405	gcc 4.1.2	For Kernel Modules: ppc405Vx6.6gcc4.1.2 For Real Time Processes: ppc405Vx6.6gcc4.1.2_rtp

Table 1.10 **Windows Platforms**

Operating System	CPU	Compiler or Software Development Kit[1] [2]	RTI Architecture
Windows 7	x86	Visual Studio 2010	i86Win32VS2010
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	i86Win32dotnet2.0
	x64	Visual Studio 2010	x64Win64VS2010
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	x64Win64dotnet2.0
Windows 2000	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2005 SP1	i86Win32VS2005
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	i86Win32dotnet2.0
Windows 2003	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2008 SP1	i86Win32VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	i86Win32dotnet2.0
	x64	Visual Studio 2005 SP1	x64Win64VS2005
		Visual Studio 2008 SP1	x64Win64VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	x64Win64dotnet2.0
Windows CE 6.0 (target only)[3] [4]	armv4	Visual Studio 2005 (C++ 8.0) (Service Pack 1)	armv4WinCE6.0 VS2005
	x86		i86WinCE6.0VS2005

Table 1.10 **Windows Platforms**

Operating System	CPU	Compiler or Software Development Kit[1] [2]	RTI Architecture
Windows Server 2008 R2	x64	Visual Studio 2010	x64Win64VS2010
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	x64Win64dotnet2.0
Windows Vista	x86	Visual Studio 2005 SP1	i86Win32VS2005
		Visual Studio 2008 SP1	i86Win32VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	i86Win32dotnet2.0
	x64	Visual Studio 2005 SP1	x64Win64VS2005
		Visual Studio 2008 SP1	x64Win64VS2008
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	x64Win64dotnet2.0
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
Windows XP Professional [5]	x86	Visual C 6.0	i86Win32VC60
		Visual C 7.0	i86Win32VC70
		Visual Studio 2003	i86Win32VS2003
		Visual Studio 2005 SP1	i86Win32VS2005
		Visual Studio 2008 SP1	i86Win32VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Win32jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	i86Win32dotnet2.0
	x64	Visual Studio 2005 SP1	x64Win64VS2005
		Visual Studio 2008 SP1	x64Win64VS2008
		Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Win64jdk
		Visual Studio 2005 (C++, C# 8.0 or 9.0) SP 1	x64Win64dotnet2.0

1. On Windows XP: If you are using JDK 5.0 and want to use Intel's HyperThreading technology, use JDK 5.0 Update 6 (build 1.5.0_06), which includes fixes to JNI and HyperThreading. (If you must use Update 5 (build 1.5.0_05), you should disable HyperThreading.)
2. The RTI .NET assemblies are supported for both the C++/CLI and C# languages. The type support code generated by rtiddsgen is in C++/CLI; compiling the generated type support code requires Microsoft Visual C++. Calling the assembly from C# requires Microsoft Visual C#.

3. The Windows CE network stack does not support the IP_TOS socket option.
4. The Windows CE device must be connected directly to the network, not through a Windows PC using ActiveSync. RTI Data Distribution Service does not support Windows CE when used with ActiveSync
5. Windows XP does not support IP_TOS unless registry changes are made. See http://support.microsoft.com/kb/248611, http://www.microsoft.com/technet/technetmag/issues/2007/02/CableGuy/default.aspx.

2 Compatibility

RTI strives to provide a seamless upgrade path when the product is updated. When upgrading to a new version of *RTI Data Distribution Service*, there are 4 components to consider:

- Wire Protocol Compatibility (Section 2.1)
- Code Compatibility (Section 2.2)
- Data-Format Compatibility and Extensibility (Section 2.3)
- ODBC Database Compatibility (Section 2.4)
- Build Compatibility (Section 2.5)

2.1 Wire Protocol Compatibility

2.1.1 General Information on RTPS (All Releases)

RTI Data Distribution Service communicates over the wire using a formal Real-time Publish-Subscribe (RTPS) protocol.

RTPS has been developed from the ground up with performance, interoperability and extensibility in mind. The RTPS protocol is an international standard managed by the OMG. The RTPS protocol has built-in extensibility mechanisms that enable new revisions to introduce new message types, extend the existing messages, or extend the Quality of Service settings in the product—without breaking interoperability.

RTPS 1.0 was introduced in 2001. The current version is 2.1. RTI plans to maintain interoperability between middleware versions based on RTPS 2.x.

2.1.2 Release-Specific Information for RTI Data Distribution Service 4.5

RTI Data Distribution Service 4.5 is compatible with 4.2 - 4.4, except as noted below.

2.1.2.1 RTPS Versions

RTI Data Distribution Service 4.5 supports RTPS 2.1. Earlier releases (4.2c and lower) supported RTPS 1. Because the two RTPS versions are incompatible with each other, appli-

cations built with 4.2e and higher will not interoperate with applications built using 4.2c or lower.

2.1.2.2 double, long long, unsigned long long or long double Wire Compatibility

If your *RTI Data Distribution Service* 4.3 or higher application's data type uses a 'double,' 'long long,' 'unsigned long long,' or 'long double,' it will not interoperate with *RTI Data Distribution Service* applications built with version 4.2e or lower, unless you use the **-use42eAlignment** flag when generating code with *rtiddsgen*.

2.1.2.3 Sending 'Large Data' between 4.4d and Older Releases

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

This issue has been resolved, starting with *RTI Data Distribution Service* 4.4d. As a result, by default, large data in *RTI Data Distribution Service* 4.4d and higher is not compatible with older versions. You can achieve backward compatibility by setting the following properties to "1".

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

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

For example:

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

2.2 Code Compatibility

2.2.1 General Information (All Releases)

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

RTI Data Distribution Service primarily consists of a library and a set of header files. In most cases, upgrading simply requires you to recompile your source using the new header files and link the new libraries. In some cases, minor modifications to your application code might be required; any such changes are noted in this document.

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

While this is not the common case, some upgrades require you to regenerate the code produced by *rtiddsgen*. The regeneration process is very simple; you only need to run the new version of *rtiddsgen* using the original input IDL file. This process will regenerate the header and source files, which can then be compiled along with the rest of your application.

2.2.2 Release-Specific Information for RTI Data Distribution Service 4.5

2.2.2.4 Type Support and Generated Code Compatibility

❑ **long long Native Data Type Support**

Starting with release 4.5c, we assume all platforms natively support the 'long long' data type. There is no longer a need to define RTI_CDR_SIZEOF_LONG_LONG to be 8 on some platforms in order to map the DDS 'long long' data type to a native 'long long' type.

❑ **double, long long and unsigned long long Code Generation**

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

❑ **Changes in Generated Type Support Code**

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

❑ **Cross-Language Instance Lookup when Using Keyed Data Types**

In *RTI Data Distribution Service* 4.3, keys were serialized with the incorrect byte order when using the Java and .NET API for the user-defined data type, resulting in incorrect behavior in the **lookup_instance()** and **get_key()** methods when using keyed data-types to communicate between applications in these languages and other programming languages. This issue was resolved in Java starting in version 4.3e rev. 01 and starting in .NET in 4.4b.

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

This issue does not impact systems using 4.2 and earlier versions of *RTI Data Distribution Service*.

❑ **Data Type with Variable Size Keys**

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

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

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

- Samples with the same key may be identified as different instances. (For the case in which the *DataWriter* uses 4.4d or higher, this can happen only if the *DataWriter's* **disable_inline_keyhash** field (in the DataWriterProtocolQosPolicy) is true (this is not the default case).
- Calling **lookup_instance()** on the *DataReader* may return HANDLE_NIL even if the instance exists.

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

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

2.2.2.5 Other API and Behavior Changes

❑ **New on_instance_replaced() method on DataWriterListener**

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

❑ **Counts in Cache Status and Protocol Status changed from Long to Long Long**

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

❑ **Changes in RtpsReliableWriterProtocol_t**

Starting in release 4.4c, two fields in DDS_RtpsReliableWriterProtocol_t have been renamed:

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

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

❑ **Tolerance for Destination-Ordering by Source-Timestamp**

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

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

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

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

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

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

Starting with release 4.4d, the default XML QoS Profiles file has been renamed and is installed in a new directory:

- Old location/name:
  $NDDSHOME/resource/xml/NDDS_QOS_PROFILES.xml
- New location/name:
  $NDDSHOME/resource/qos_profiles_<version>/xml/NDDS_QOS_PROFILES.example.xml (where <version can be 4.4d, for example)

If you want to use this QoS profile, you need to set up your **NDDSHOME** environment variable at run time, and rename the file **NDDS_QOS_PROFILES.example.xml** to **NDDS_QOS_PROFILES.xml** (i.e., by default, even if your **NDDSHOME** environment variable is set, this QoS profile is not used.) See Section 15.2 in the *User's Manual* for details.

❑ **Changes in the default value for the max_objects_per_thread field**

Starting with release 4.4d, the default value for the **max_objects_per_thread** field in the SystemResourceLimitsQosPolicy has been changed from 512 to 1024.

❑ **Type Change in Constructor for SampleInfoSeq—.NET Only**

The constructor for SampleInfoSeq has been changed from SampleInfoSeq(UInt32 maxSamples) to SampleInfoSeq(Int32 maxSamples). This was to make it consistent with other sequences.

2.3 Data-Format Compatibility and Extensibility

With *RTI Data Distribution Service*, you can define your own data types, which will be used to communicate between applications on a network.

Developers sometimes extend or modify their data types. When new fields are added, it is a common requirement that applications that use the extended types must still communicate with applications using the old types.

To accomplish this, RTI provides a dynamic data API, which allows applications to define data types at runtime without code generation. Specifically, this API supports the concept of a *sparse type*, one for which every data sample need not contain a value for every field defined in the type. By dynamically defining a type as sparse, an application

can add new fields to it later without breaking existing components that may be unable to fill in those new fields.

For customers that may want to handle extensibility in a more custom way, such as with XML payloads or custom serialization and deserialization, RTI provides built-in string and opaque data types.

2.4 ODBC Database Compatibility

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

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

Starting with version 4.5c, we have tested the following drivers:

- Oracle TimesTen 11.2.1
- MySQL ODBC 5.1

Performance will be better with Oracle TimesTen, because it is an in-memory database.

If you choose MySQL, you will also need the MySQL ODBC 5.1.6 (or higher) driver. For non-Windows platforms, UnixODBC 2.2.12 (or higher) is also required.

The Durable Writer History and Durable Reader State features have been tested with the following architectures:

- **AIX:** p5AIX5.3xlc9.0, 64p5AIX5.3xlc9.0 (AIX architectures only support Oracle TimesTen)
- **Linux:** i86Linux2.6gcc3.4.3, x64Linux2.6gcc3.4.5, i86Linux2.6gcc4.1.1 and x64Linux2.6gcc4.1.1
- **Solaris:** sparcSol2.10gcc3.4.2 and sparc64Sol2.10gcc3.4.2
- **Windows:** i86Win32VS2005, i86Win32VS2008, i86Win32VS2010, and x64Win64VS2010

For more information on database setup, please see the *Addendum for Database Setup* (**RTI_DDS_GettingStarted_DatabaseAddendum.pdf**).

2.5 Build Compatibility

2.5.1 Windows C/C++ Build Compatibility

When building C/C++ applications on Windows systems, there is an additional dependency on **Iphlpapi.lib** that was not required with *RTI Data Distribution Service* 4.4d or previous releases.

Therefore if you rebuild an existing 4.4d or older project, you will see the following link error for static builds.

```
nddscorez[d].lib (Host.obj): error LNK2019: unresolved external symbol
_GetAdaptersAddresses@20 referenced in function
_RTIOsap.Host_getHWAddress*.
```

To correct this error, you must add the new link dependency **Iphlpapi.lib** to the appropriate Visual Studio project. You can do this manually, or you can delete the project files that were created with 4.4d or older and then run **rtiddsgen -example** to replace them.

When using i86Win32VC60: A newer Platform SDK that has **Iphlpapi.lib** must be used, as the Platform SDK included with the default Visual C++ 6.0 installation does not have it.

3 What's Fixed in 4.5c

This section includes:

- ❏ Fixes Related to Reliability (Section 3.1)
- ❏ Fixes Related to Content Filtering (Section 3.2)
- ❏ Fixes Related to QueryConditions and ReadConditions (Section 3.3)
- ❏ Fixes Related to Java (Section 3.4)
- ❏ Fixes Related to rtiddsgen (Section 3.5)
- ❏ Fixes Related to .NET (Section 3.6)
- ❏ Fixes Related to Statuses and Listeners (Section 3.7)
- ❏ Fixes Related to QoS and Other Miscellaneous Fixes (Section 3.8)

3.1 Fixes Related to Reliability

3.1.1 DataWriter May Not Have Properly Switched between Normal and Fast Heartbeat Modes

When sending data in bursts, a *DataWriter* may not have properly switched between normal and fast heartbeat periods. This could have resulted in the *DataWriter* being in normal rather than fast heartbeat mode, and vice versa. If the normal heartbeat period was very long, and piggyback heartbeats are infrequent, reliability consequently could have been stuck until the next periodic heartbeat. This issue has been resolved.

[RTI Bug# 13106]

3.1.2 Reliable DataWriters did not Repair Requested Data Fragments

Previously, reliable *DataWriters* had a problem repairing large data, where requests for missing data fragments (NACK_FRAG submessages) were not serviced and consequently reliable communication halted. This problem has been resolved.

[RTI Bug # 13252]

3.1.3 wait_for_historical_data Returned Before All Historical Data Received

The *DataReader's* **wait_for_historical_data()** operation may have returned before all historical data was received. The problem occurred whenever **wait_for_historical_data()** was called before a *DataReader* received any Heartbeat messages. Consequently, **wait_for_historical_data()** would return prematurely upon receiving the next sample, instead of upon receiving all historical samples. This problem has been resolved.

[RTI Bug # 12250]

3.2 Fixes Related to Content Filtering

3.2.1 DataWriter May Have Crashed if Writer-Side Content Filtering Enabled

Previously, a *DataWriter* with writer-side content-filtering enabled may have crashed when writing to several *DataReaders* that shared the same destination address. This problem has been resolved.

[RTI Bug # 13446]

3.2.2 Statistics on Filtered Samples not Updated for Best-effort Writer and Reader-side Filtering

Any samples being filtered, either due to a ContentFilteredTopic or TimeBasedFilter QoS policy, on either the DataWriter or DataReader, should be captured in the DataWriterProtocolStatus or DataReaderProtocolStatus when getting reader status.

However in the previous release, the **DataReaderProtocolStatus.filtered_sample_[count/bytes]** was always zero. Additionally, the **filtered_sample** counts for best-effort *DataWriters* were not being updated, so filtered samples were not being accounted for. These problems have been resolved.

[RTI Bug # 13282]

3.2.3 Prematurely Removed Destination for a Reliable DataWriter with Writer-side Content Filtering

A race condition during unmatching of a reliable content-filtering *DataReader* from a reliable *DataWriter* with writer-side content-filtering enabled could have resulted in a destination shared between this unmatched reader and another existing reader being incorrectly removed. Consequently, the *DataWriter* would have been unable to reach the existing *DataReader* on that (removed) destination. This issue has been resolved.

[RTI Bug # 13422]

3.2.4 Memory Leak when Using Writer-side Filtering with CORBA/C++ or Dynamic Data

Previously, for applications using CORBA (C++) or Dynamic Data, a *DataWriter* using writer-side content-filtering leaked memory with each sample written. This caused the application to prematurely run out of memory.

A workaround for this issue was to turn off writer-side filtering by setting **writer_resource_limits.max_remote_reader_filters** (in DDS_DataWriterQos) to zero. This problem has been resolved and the workaround is no longer necessary.

[RTI Bug # 13224]

3.2.5 Asynchronous Publisher Unnecessarily Evaluated Content Filter

A *DataWriter* with asynchronous publish-mode enabled does not support writer-side content-filtering. However, in the previous release, such a *DataWriter* would evaluate a content filter as if it was performing writer-side filtering. This inefficiency has been fixed.

[RTI Bug # 13449]

3.3 Fixes Related to QueryConditions and ReadConditions

3.3.1 Performance Problems when Using ReadConditions

In the previous release, you may have experienced some performance degradation when using a read/take_w_condition API on a *DataReader* if there were a lot of samples to go through. ReadConditions have been re-implemented to improve performance.

[RTI Bug # 11270]

3.3.2 Potential Incorrect Behavior if New QueryCondition Created After Deleting Previous QueryCondition

Deleting a QueryCondition filter did not clear filter status bits for existing samples, leaving the samples to be erroneously picked up and requeued if another QueryCondition filter was recreated in the same slot. This problem has been resolved.

This problem has been resolved.

[RTI Bug # 13229]

3.3.3 Possible Segmentation Fault when QueryCondition Deleted

Deleting a QueryCondition may have resulted in a segmentation fault. The problem exhibited mainly if QueryConditions were repetitively created and destroyed. This problem has been resolved. This problem has been resolved.

[RTI Bug # 13258]

3.4 Fixes Related to Java

3.4.1 DataReader Crashed if DataWriter disable_inline_keyhash Enabled—Java Only

A Java *DataReader* receiving from a *DataWriter* with disabled inline keyhash (**writer_qos.protocol.disable_inline_keyhash** = true) either failed to receive samples or crashed due to deserialization errors. This problem has been resolved.

[RTI Bug # 13374]

3.4.2 Null Pointer Exception if No Type Code Sent when Using ContentFilter—Java Only

If a content filter was used in Java and the type code received in discovery data was null (either because the type code was not sent by the sender, or the received type code was larger than the **type_code_max_serialized_length** in the DomainParticipantResourceLimitsQosPolicy), a null pointer exception would occur. This problem has been resolved.

[RTI Bug # 13342]

3.4.3 Setting Null Listener with Non-Empty Mask should be Allowed—Java Only

In the previous release in the Java API, having a null listener with a mask that was not none was incorrectly treated as an error. It should have been treated as a no-op listener. This problem has been resolved.

[RTI Bug # 13230]

3.4.4 Memory Leak when using DynamicData Sequence with Read/Take API— Java Only

If an empty DynamicData sequence was used to call a read or take operation in Java, there was a leak to the native memory after calling **return_loan()**. This problem has been resolved.

[RTI Bug # 13051]

3.4.5 Errors When Deleting DomainParticipant After Unregistering Built-in Content Filters—Java Only

Calls to **delete_participant()** may have failed and thrown the following exception if any of the built-in content filters (SQL or StringMatch) were unregistered before the *DomainParticipant* was deleted.

```
com.rti.dds.infrastructure.RETCODE_ERROR
        at com.rti.dds.util.Utilities.rethrow
        at com.rti.dds.infrastructure.RETCODE_ERROR.check_return_codeI
        at com.rti.dds.domain.DomainParticipant-
   Impl.unregister_contentfilter
        at com.rti.dds.domain.DomainParticipantImpl.before_delete_native
        at com.rti.dds.infrastructure.NativeFactoryMixin.delete_entityI
        at com.rti.dds.domain.DomainParticipantFactory-
   Impl.delete_participant
```

This problem has been resolved.

[RTI Bug # 13448]

3.4.6 Creation of DataWriters and DataReaders Took a Long Time with Complex IDL Types with Sequences and Arrays—Java Only

The creation of a *DataWriter* and a *DataReader* took a long time when the associated IDL type contained multiple nested sequences and arrays or single sequences and arrays with many elements.

For example:

```
struct A {
    sequence<B, 1000000000> m1;
};
```

This problem has been resolved.

[RTI Bug # 13209]

3.4.7 DataReader Creation Slow for Topics with Union Type if Discriminator is Enumeration—Java Only

The creation of a *DataReader* on a Topic with a union type where the discriminator is an enumeration was sometimes slow. For example:

```
enum MyEnum {
    ENUM_1 = 0,
    ENUM_2 = 1000000000
};
union MyUnion switch (MyEnum) {
    case ENUM_1:
        long member1;
    default:
        short member2;
};
```

This problem has been resolved by changing the code generated by *rtiddsgen*.

[RTI Bug# 13450]

3.5 Fixes Related to rtiddsgen

3.5.1 Incorrect Output Format from TypeSupport_print_data() for Character Data Type

TypeSupport_print_data() printed a character as a 4-byte hex number (such as <ffffff90> instead of the correct output of <90>). This problem has been resolved.

[RTI Bug # 13191]

3.5.2 Possible Exception from TypeSupport_print_data() when printing Character Data Type—with Debug Libraries on Windows Systems Only

On Windows systems when compiling against debug libraries, **TypeSupport_print_data()** threw an exception if the type contained a DDS_Char field and the value of the DDS_Char being printed was < 0. This problem has been resolved.

[RTI Bug # 13192]

3.5.3 Error from rtiddsgen If No Preprocessor Specified—Windows Systems Only

If you used the version of *rtiddsgen* included with *RTI Data Distribution Service* 4.5b and no preprocessor was found, you would have seen error messages such as:

```
>rtiddsgen MyTypes.idl
Cannot run program "CL.EXE": CreateProcess error=2, The system can-
not find the file specified
```

This problem has been resolved. In this release, *rtiddsgen* will print a message to inform you to use the **-ppDisable** command-line option to disable the preprocessor.

[RTI Bug # 13376]

3.5.4 Incorrect Redefinition Compilation Error if const with Same Name was Defined Twice in Different Scope

rtiddsgen will report a redefinition error if a constant with the same name is defined twice in different scopes and one of the definitions is within an interface.

For example, the compilation of the following IDL failed with a redefinition error for the constant l:

```
module M1
{
  module M2
  {
    const short l = 4;
    interface A
    {
      const short l = 4;
      long foo(in long x);
    };
  };
};
```

Since the two **l** constants are defined in different scopes, this should not have caused an error. This problem has been resolved.

[RTI Bug # 13239]

3.5.5 Compilation Errors with Types Contained in Module with Same Name

If IDL files had a type contained in a module with the same name than the type, the generated code was incorrect and in many cases did not compile.

For example, the generated code for this example did not compile in C++ with namespaces.

```
module A {
    module B {
        struct A {
            short s;
        };
    };
};
```

The generated code for the following example did not compile in any language.

```
module A {
    module B {
        struct A {
            short s;
        };
        struct C {
            A::B::A
        };
    };
};
```

This problem has been resolved.

[RTI Bug # 13275 and # 13311]

3.5.6 Incorrect Parsing of Keys in Typedefs by rtiddsgen

When only some of the fields of a typedef type were used as keys, *rtiddsgen* did not parse the key fields correctly.

In the following example, both **key.m1** and **key.m2** were considered keys. However, only **key.m1** is a key.

```
struct MyType {
    long m1; //@key
    long m2;
};
typedef MyType MyTypedef;
struct MyKeyedType {
    MyTypedef key; //@key
};
```

This problem has been resolved.

[RTI Bug # 13415]

3.5.7 Improper Constructor in Generated TypeSupport Code for Value Types whose Keys are Only in Base Type—Java Only

Value types where keys are only present in the base value type were not treated as keyed types in Java. For example:

```
valuetype MyBaseType {
    public long m1; //@key
};
valuetype MyDerivedType: MyBaseType {
    public long m2;
};
```

In the above example, MyDerivedType was not considered a keyed type in Java. This problem has been resolved.

[RTI Bug # 13441]

3.5.8 Compilation Error when using rtiddsgen -use42eAlignment—Java Only

The code generated from **'rtiddsgen -use42eAlignment -language Java'** did not compile. This problem has been resolved.

[RTI Bug # 12991]

3.5.9 Possible Failure in rtiddsgen if Path to IDL File Contained Whitespace

rtiddsgen could have failed to generate code if the IDL file was in a location that had whitespace in its path. This issue has been resolved.

[RTI Bug # 13205]

3.6 Fixes Related to .NET

3.6.1 Incorrect Content Filter Information in SubscriptionBuiltinTopicData—.NET Only

In the previous release, the **related_topic_name** field in the ContentFilterProperty_t structure contained the name of the **content_filtered_topic_name** instead of the related topic name. This problem has been resolved.

[RTI Bug # 13232]

3.6.2 Extraneous 'Lost Data' Calls for Best-Effort, Content-Filtered Readers—.NET Only

In the previous release, best-effort *DataReaders* using content-filtered topics would incorrectly consider some samples to be lost, resulting in extraneous onLostData callbacks. This problem has been resolved.

[RTI Bug # 13323]

3.7 Fixes Related to Statuses and Listeners

3.7.1 Incorrect DataReaderProtocolStatus with Matched Publication

The *DataReader's* **get_matched_publication_datareader_protocol_status()** operation retrieved the wrong DataReaderProtocolStatus. This problem has been resolved.

[RTI Bug # 13279]

3.7.2 Reliable DataReader duplicate_sample_count not Updated

When a reliable *DataReader* received samples with identical sequence numbers, in most cases the **DDS_DataReaderProtocolStatus.duplicate_sample_count** was not incremented. This problem has been resolved.

[RTI Bug # 13447]

3.7.3 Warning and Incorrect Statistics if max_gather_destinations Exceeded

A reliable *DataWriter* would print the following warning if the number of destinations for a sample was greater than **participant.resource_limits.max_gather_destinations**:

```
COMMENDSrWriterService_write: !modify srw writer locator table
```

In addition to this warning, the peer locator statuses were not calculated correctly. This problem has been resolved.

[RTI Bug # 13238]

3.7.4 NULL Listener with INCONSISTENT_TOPIC_STATUS in DomainParticipant Listener Mask was Incorrectly Disallowed—Debug Mode Only

Setting a NULL listener with a non-empty mask should act as a no-op listener without resetting status. In the previous release, we incorrectly did not allow the setting of a NULL *DomainParticipant* Listener with INCONSISTENT_TOPIC_STATUS turned on in the listener mask after the *DomainParticipant* was created. This problem, which only occurred when using debug mode, has been resolved.

[RTI Bug # 13378]

3.8 Fixes Related to QoS and Other Miscellaneous Fixes

3.8.1 Keyed DataReader May Have Crashed if Samples Received Out-of-Order and DestinationOrder kind Set to BY_SOURCE_TIMESTAMP

If a keyed *DataReader* with BY_SOURCE_TIMESTAMP DestinationOrder kind received a sample with a timestamp that was earlier than the last received sample (as determined by the source timestamp) but the sample was not dropped because it was of a different instance and still in-order for that instance, it may have caused a crash. This was more likely to happen if there were multiple *DataWriters* on different hosts whose clocks were not synchronized, or if the data was not taken immediately by the *DataReader* when available (for example, if the data was polled periodically by the *DataReader*).

[RTI Bug # 13240]

3.8.2 Possible Memory Corruption Due to Improper PropertyQosPolicy Consistency Check

The PropertyQosPolicy consistency check did not take into account the associated resource limits in **DomainParticipantQos.resource_limits.** This lack of checking may have caused memory corruption with the release libraries and an error to be printed with the debug libraries. This problem has been resolved. Now, the middleware will report an inconsistency error.

[RTI Bug # 13420]

3.8.3 Unregister_instance() May Crash when Given Invalid Instance Handle

unregister_instance() might have crashed if given an invalid, non-NIL instance handle. You may have seen this error message:

```
Run-Time Check Failure #3 - The variable "sampleCount_out" is being
used without being initialized.
```

This problem has been resolved. We now return RETCODE_BAD_PARAMETER in this case.

[RTI Bug # 13214]

3.8.4 Transport Priority Not Always Used by Transport Plugins

The transport priority value in the TransportPriority QoS policy was not always used by the transport plugins, even for the cases in which we could have used it.

This problem has been resolved. [RTI Bug # 13278]

3.8.5 DomainParticipant Default Multicast Locator not Applied to DataReaders

The default multicast locator parameter sent in *DomainParticipant* discovery announcement messages was not correctly processed. Consequently, *DataReaders* did not inherit

any default multicast locators from *DomainParticipants*, making our implementation not fully compliant with the RTPS specification. This problem has been resolved.

[RTI Bug # 13443]

3.8.6 Error Deleting DomainParticipant if PublishModeQosPolicy Changed After DataWriter Created

The PublishModeQosPolicy is documented as immutable, meaning it cannot be changed after the *DataWriter* is created. However, the previous release did not enforce this rule. This caused a problem if you changed the QoS policy after the *DataWriter* was created and later tried to delete the *DomainParticipant*. In this scenario, you would have gotten an error such as the following:

```
[D0056|DELETE Participant|D0056|DELETE
FlowController]PRESParticipant_destroyFlowController:!delete flowController
[D0056|DELETE Participant|D0056|DELETE
FlowController]DDS_FlowController_destroyI:!delete PRESFlowController
[D0056|DELETE
Participant]DDS_DomainParticipant_deleteBuiltinFlowControllersI:!delete
DDS_DEFAULT_FLOW_CONTROLLER_NAME
[D0056|DELETE Participant]DDS_DomainParticipant_destroyI:!delete builtin
flow controllers
[D0056|DELETE
Participant]DDS_DomainParticipantFactory_delete_participant:!delete partic-
ipant
```

This problem has been resolved. In this release, the middleware enforces the rule that the PublishModeQosPolicy is immutable. This in turn prevents the above error during *DomainParticipant* deletion.

[RTI Bug # 13451]

3.8.7 Calls to delete_contained_entities() Failed if DomainParticipant not Enabled—Debug Mode Only

In the previous release, in debug mode only, calls to **delete_contained_entities()** on a *Publisher, Subscriber,* or *DomainParticipant* would fail if the *DomainParticipant* was not enabled. This problem has been resolved.

[RTI Bug # 13471]

3.8.8 Potential Crash when Deleting DataReader on Unkeyed Topic when Using Batching

When deleting a *DataReader* on an unkeyed topic when using batching, there was a rare case in which this may have caused the application to crash. In this case, the following message would have been logged prior to deletion of the *DataReader*:

```
"PRESPsReaderQueue_newData:node already created".
```

This problem has been resolved. [RTI Bug # 13055]

3.8.9 Possible Crash when a Cloned TypeCode was Deleted—Only with Dynamic Data

Your application may have crashed if the TypeCode returned by the call **DDS_DynamicDataTypeSupport_get_data_type()** was cloned using the API **DDS_TypeCodeFactory_clone_tc()** and the new TypeCode was deleted afterwards. This problem has been resolved.

[RTI Bug # 13255]

4 What's Fixed in 4.5b

This section includes:

- Fixes Related to Java (Section 4.1)
- Fixes Related to .NET (Section 4.2)
- Fixes Related to Content Filtered Topics (Section 4.3)
- Fixes Related to Dynamic Data (Section 4.4)
- Fixes Related to Batching (Section 4.5)
- Fixes Related to rtiddsgen (Section 4.6)
- Fixes Related to Other Utilities (Section 4.7)
- Fixes Related to QoS and Other Miscellaneous Fixes (Section 4.8)

4.1 Fixes Related to Java

4.1.1 get_qos_profile_libraries() and get_qos_profiles() May Have Crashed if it was First Call on DomainParticipantFactory—Java only

In the previous release, if **get_qos_profile_libraries()** or **get_qos_profiles()** was the first call on the DomainParticipantFactory, your application may have crashed or hung. You may have seen an error message such as:

```
# An unexpected error has been detected by Java Runtime Environment:
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x02dd5d0d, pid=440,
tid=560
# Java VM: Java HotSpot(TM) Client VM (11.3-b02 mixed mode, sharing
windows-x86)
# Problematic frame:
# C [nddsjava.dll+0x15d0d]
```

This problem has been resolved.

[RTI Bug # 13073]

4.1.2 Root Permission Required to Modify Real-Time Thread Priorities—Java Only

In the previous release when using Java, you needed to be *root* to run DDS threads with real-time priority.

This problem has been resolved. You no longer have to be *root*; you can be granted **rtprio** privileges via **/etc/security/limits.conf** to run real-time threads.

[RTI Bug # 12915]

4.1.3 Real-Time Thread Priority Could Not be Set to 5—Java Only

In the previous release when using Java, you could not specify a DDS thread's ThreadSettingsKindMask to include THREAD_SETTINGS_REALTIME_PRIORITY and set its priority to 5. This problem has been resolved.

[RTI Bug # 12916]

4.2 Fixes Related to .NET

4.2.1 Crash when Copying a Sequence of Samples—.NET Only

A *DataReader's* **read()** and **take()** methods place received data samples into application-provided sequences. When copying these sequences, previous versions of *RTI Data Distribution Service* could crash under the following circumstances:

Passing sequences of already allocated samples to the **read()** or **take()** method indicates that the middleware should copy over those samples. In this case, the copy could cause a crash within the context of the **read()** or **take()** call.

After the **read()** or **take()** call returned, a subsequent call to the sequence's **copy_from()** method in application code could cause a crash.

These problems have been resolved. [RTI Bug # 13030]

4.2.2 Generated Code for copy_from() Failed to Perform Deep Copy—.NET Only

In the previous release, .NET IDL types with complex members were not copied correctly with the **copy_from()** code generated by *rtiddsgen.* Only the reference to the complex members was copied, not the actual data.This problem has been resolved.

[RTI Bug # 13047]

4.3 Fixes Related to Content Filtered Topics

4.3.1 DataWriter May Have Crashed when Writing to Multiple DataReaders with ContentFilteredTopic

A *DataWriter* may have crashed if many matching content-filtered *DataReaders* were joining and leaving the system. Specifically, when the *DataWriter* decided whether to do writer-side content-filtering for a *DataReader*, its internal state was incorrectly maintained such that memory corruption could occur when the *DataWriter* switched between filtering and not filtering, due to the changing number of matched readers. This problem has been resolved.

[RTI Bug # 13193]

4.3.2 'Serialize Key' Error when Reader Filtered Out First Sample Received for Instance—Dynamic Data, CORBA/C++, or .NET

If the first sample received for an instance was filtered out by a *DataReader*, you may have seen the following error messages:

```
PRESCstReaderCollator_addRegisteredInstanceEntry:!serialize key
PRESCstReaderCollator_addInstanceEntry:!add registered instance
```

This problem only affected *DataReaders* using dynamic data, or a .NET type plugin, or a CORBA/C++ type plugin. This problem has been resolved.

[RTI Bug # 13156]

4.3.3 Error Between Java and C/C++/C# Applications when Content Filter is Applied on Discriminator Member of Union Type

In the previous release, if you wanted to apply a content filter on the discriminator member of a union type in C/C++/C#, the discriminator was referred to as "**_d**", while in Java, it was referred to as "**discriminator**". However, since the filter expression is not language-portable, this did not work when the Java application was communicating with the C++/C/C# application using a content filter on the discriminator member of a union data type.

This problem has been resolved. In this release, the discriminator should always be referred to as "**_d**" in the filter expression, regardless of the language.

[RTI Bug # 12926]

4.4 Fixes Related to Dynamic Data

4.4.1 Serialization Error or Data Loss after Series of 'set' Operations when Using Dynamic Data with 'long long' or 'double' Member

It was possible for a series of 'set' operations on variable-sized members (strings or sequences) on a dynamic data object to result in inconsistent data if there were also 8-byte-aligned members (long long or double). This could have resulted in a failure to serialize on write, or a loss of data. This problem has been resolved. [RTI Bug # 13006]

4.4.2 Serialization Error or Data Loss when Sending Dynamic Data for a Sparse Data Type in which not all Members Explicitly Initialized

If not all members were explicitly initialized in dynamic data for a sparse data type on the writer, this may have caused inconsistent data, resulting in failure in serializing data or loss of data when sending data. This problem has been resolved. [RTI Bug # 13005]

4.4.3 Incorrect Access of String Arrays and Sequences with Maximum String Length in Dynamic Data

Dynamic data may have failed to properly access a data member that was an array or sequence of strings if there was a string of maximum length. It also failed to properly access any data members following the string array/sequence. For example, consider this type:

```
struct Foo {
  long x;
  string<10> arrstr[2];
  long y;
};
```

If either member of **arrstr** was set to a string of length 10 (e.g., "TestString"), then calling **bind/get_complex_member()** on **arrstr** may have returned an empty or truncated data structure. You may also have had problems setting or getting the value of **y**. Note that this issue did not affect proper printing of the value of the sample with **DynamicData::print()** or serializing the value. This problem has been resolved.

[RTI Bug # 12940]

4.4.4 'Internal Error' When Using Dynamic Data on 64-bit Architectures

When using Dynamic Data, particularly on 64-bit architectures, you may have seen the following error message:

```
DDS_DynamicDataStream_assert_primitive_member: internal error try-
ing to unexpected size
```

This problem has been resolved.

[RTI Bug # 13144]

4.4.5 Inability to Access Fields of Union Data Type by Name in Dynamic Data in Certain Cases

In the previous release, you could not access members of a union data-type by name in certain cases. For example, consider the following data type:

```
union CommandDetails switch(long) {
    case 1:
        long service_mask;
    case 2:
    case 3:
    case 4:
        EntityReference entity;
    case 5:
        Response result;
};
```

```
struct Command {
    ...
    CommandDetails myunion;
}
```

If you received dynamic data of the above type that had discriminator 3 for the union, you would have gotten errors from the following calls:

- **myunion.member_exists("entity", 0)**
- **myunion.get_member_info("entity", 0).member_id**
- **myunion.get_complex_member("entity", 0)**

This problem has been resolved.

[RTI Bug # 12949]

4.4.6 Message may be Printed when Accessing Union Member of Dynamic Data with Default Discriminator Case

When accessing dynamic data of types that contained union members with a defined default discriminator case, you may have seen the following message:

```
"DDS_TypeCode_get_type_serialized_min_size:typecode error 7 on
member_type"
```

This could have occurred when writing to a sample of such a data type (specifically, a member after the union), or when calling **get_key_value()** on the reader or writer of this data type. This problem has been resolved.

[RTI Bug # 12929]

4.4.7 DynamicData's get_member_info() May Have Returned Incorrect Data for Unions

In the previous release, **get_member_info()** returned a member_exists value of true for every member of a union. Now the correct values are returned.

[RTI Bug # 13016]

4.4.8 Dynamic Data Member Info May Have Incorrectly Returned Element Kind TK_ALIAS

In the previous release, **get_member_info()** and **get_member_info_by_index()** may have incorrectly returned a data type element kind of TK_ALIAS if there is a typedef in the type.

Specifically, this may have been an issue if the type was an array or a sequence of a type defined as a typedef. In these cases, standard array/sequence access members worked but the element kind had to be deduced directly from the type of the member.

If TK_ALIAS was seen in the return member info, you could have used the following workaround:

1. Call **get_member_type()** for the member ID and/or name requested.
2. Call **TypeCode_content_type()** to get the contained element type.
3. Get the **TypeCode_kind** of the content type.

   While this kind is TK_ALIAS:

   - Get the **TypeCode_content_type** to resolve the alias.
   - Check the kind again.

This problem has been resolved; the above workaround is no longer necessary.

[RTI Bug # 12913]

4.4.9 Print Method Failed for Invalid Enum Data in Dynamic Data

DynamicData_print() failed if an enum member was found with an ordinal value that did not exist in the type definition. The failure was reported with this message:

```
DDS_DynamicDataStream_print_primitive_member: typecode error 7 on
member_name
```

This problem has been resolved.

[RTI Bug # 12912]

4.5 Fixes Related to Batching

4.5.1 Reader Configured with KEEP_LAST History May Have Crashed on Receiving a Batch Sample if max_samples was Exceeded

A *DataReader* configured with KEEP_LAST history where (**depth*****max_instances** + maximum-size-of-a-batch-in-samples) >= **max_samples**, may have crashed if it received a batch sample while the reader's queue was full. This problem has been resolved.

[RTI Bug # 13133]

4.5.2 Queue of KEEP_LAST Reliable DataReader Not Fully Utilized when Batching Enabled

In the previous release, when using reliable *DataReaders* with KEEP_LAST History and batching enabled, the full history depth may not have been fully utilized before a previously received sample was replaced by a newly received sample. This problem has been resolved.

[RTI Bug # 13140]

4.5.3 Precondition Error on Write when Batching Used for Writer Enabled via Participant's or Publisher's enable() API

Calling **write()** on a *DataWriter* configured to use batching and enabled using the *DomainParticipant's* or *Publisher's* **enable()** operation produced a precondition error similar to this:

```
PRESWriterHistoryDriver_finalizeBatchSample:!precondition
WriterHistoryMemoryPlugin_removeEntryFromSession:!finalize sample in
batch WriterHistoryMemoryPlugin_removeSingleInstanceEntry:!remove
session samples WriterHistoryMemoryPlugin_ackSample:!remove virtual
sample WriterHistoryMemoryPlugin_changeFirstReclaimableSn:!ack vir-
tual sample
PRESWriterHistoryDriver_changeFirstUnackedSn:!change_first_non_recla
imable_sn PRESPsWriter_write:!srw update first unacked sn
REDAInlineList_removeNodeEA:!precondition: list == ((void *)0) ||
node == ((void *)0) || node->inlineList != list
```

This problem has been resolved.

[RTI Bug # 12998]

4.5.4 Resource Contention Error When Using Batching and KEEP_LAST on Writer Side

In the previous release, you may have seen a resource contention error, followed by the application hanging on the writer side on a call to **write()**. This problem occurred when all of the following were true on the writer side:

- ❏ Batching was enabled
- ❏ History's **kind** was KEEP_LAST or Lifespan was used on the Writer side
- ❏ **max_samples** in the ResourceLimits QoS policy was a finite number
- ❏ The number of samples for a given instance in a batch was < History's **depth**

The error reported was:

```
PRESWriterHistoryDriver_flush:!resource contention
PRESPsWriter_flushBatchWithCursor:unexpected outstanding loans
```

This problem has been resolved.

[RTI Bug # 12979]

4.5.5 Incorrect Precondition Error on DataWriter When Using Batching with Debug Libraries

In the previous release when using batching with the debug libraries, if **resource_limits.max_samples** was less than **protocol.rtps_reliable_writer.heartbeats_per_max_samples** in the *DataWriter's* QoS, you may have seen a precondition error. This problem has been resolved.

[RTI Bug # 12980]

4.6 Fixes Related to rtiddsgen

4.6.1 XSD Generation Failed if IDL File Contained Constant Initialized with Enum Value

XSD generation using *rtiddsgen's* **-convertToXsd** option failed if the input IDL contained a constant that was initialized using an enum value. For example:

```
enum MyEnum {
    ENUM_VALUE_1=0,
    ENUM_VALUE_2
};
const long MY_CONST = ENUM_VALUE_1;
```

This problem has been resolved.

[RTI Bug # 13103]

4.6.2 Compilation Failures in C++ Generated Code Using Namespaces if Typedef Referred to Type in Different Module

C++ code generated by *rtiddsgen* using the **-namespace** flag did not compile if a symbol was redefined using a typedef in a different module. For example:

```
module A {
    struct MyStruct {
        long member;
    };
};
module B {
    typedef A::MyStruct MyStruct;
};
```

Compiling the generated code with **gcc** produced the following errors:

```
...
MyIDL.cxx: In function 'int B::MyStruct_initialize(B::MyStruct*)
MyIDL.cxx: error: call of overloaded
'MyStruct_initialize_ex(B::MyStruct*&, int)' is ambiguous
...
```

This problem has been resolved.

[RTI Bug # 12542]

4.6.3 Incorrect Code Generated For Enum's with Partially Defined Enumerator Values—Java Only

If the IDL contained an enum with some (but not all) of the enumerator values defined, *rtiddsgen* generated incorrect code for Java. You would have seen a compiler error. For example, this enum would have caused an error:

```
enum Foo {
    ENUM1 = 1,
    ENUM2
};
```

This problem has been resolved.

[RTI Bug # 13130]

4.6.4 Potentially Incorrect Maximum Key-Size Calculated by rtiddsgen for Types with Complex Keys—Java Only

When using *rtiddsgen* for Java on a type with a complex key, if not all the fields within the key type were marked as keys, the maximum key's size may have been calculated incorrectly. As a result, space may have been wasted in the *RTI Data Distribution Service* libraries. For example:

```
struct Bar {
    long i; //@key
    long j;
};
struct Foo {
    Bar bar; //@key
};
```

The key size for the above type was calculated as if all the fields within 'Bar' were marked as keys. This problem has been resolved.

[RTI Bug # 12972]

4.6.5 Deserialization Error in Java/CORBA with Types Containing Enumerations

In the previous release, you may have seen deserialization errors in Java/CORBA when the type contained members that were enums. For example:

```
enum MyEnum {
    ENUM_VALUE_1,
    ENUM_VALUE_2
};
struct MyStruct {
    long member1;
    MyEnum member2;
```

```
    double member4;
    string member5;
};
```

This problem has been resolved.

[RTI Bug # 13074]

4.6.6 Typedef TypeCode Generated for C# was Incorrect

rtiddsgen generated typecodes for typedefs for C# as if they were structures. This problem has been resolved.

[RTI Bug # 13015]

4.6.7 Incorrect CPP/CLI Code Generated by rtiddsgen if IDL Referred to Other IDL

In the previous release, **#include** directives were not processed correctly by *rtiddsgen* when generating code for CPP/CLI. As a consequence, the generated code may have not compiled in some cases.This problem has been resolved.

[RTI Bug # 13033]

4.6.8 Exception Caused by Object Creation in .NET for Types Containing Arrays of Wstrings

The constructor of a .NET type generated by *rtiddsgen* that contains arrays of wstrings would have thrown an exception in the previous release. For example:

```
struct MyType {
    long m1;
    wstring<128> member1[4];
};
```

This problem has been resolved.

[RTI Bug # 13085]

4.6.9 Incorrect Code Generated to Serialize/Deserialize Arrays of Sequences—.NET Only

In the previous release, *rtiddsgen* generated incorrect .NET code for serializing/deserializing arrays of sequences. You would not have seen a compilation error, but your application may have thrown an exception and/or crashed on serialization/deserialization.

This problem has been resolved.

[RTI Bug # 13086 and 13087]

4.6.10 Possible Incorrect Size from get_min_size_serialized()—.NET and C++/CORBA Only

The generated method **get_min_size_serialized()** may have returned the wrong size in .NET and C++/CORBA. This method is only used only with batching configurations.

If the returned value was smaller than the expected value (C++/CORBA only), batches sized using batch.max_data_bytes may have been flushed before max_data_bytes was reached.

If the returned value was bigger than expected (.NET and C++/CORBA), *RTI Data Distribution Service* used more memory than needed, but correctness was not affected.

These problems have been resolved.

[RTI Bug # 13084 and 13075]

4.6.11 Incorrect Example Code Generated for Publisher's unregister_instance() Call—C# Only

The previous version of *rtiddsgen* generated incorrect C# example code for the *Publisher's* **unregister_instance()** call (in the file **<*Type*>_publisher.cs**). By default, the call to unregister is commented out, but if the user decides to use it, the code will not compile.

This problem has been resolved.

[RTI Bug # 13100]

4.7 Fixes Related to Other Utilities

4.7.1 rtiddsping Reader Incorrectly Reported Missing Samples at Startup

The reader for the *rtiddsping* utility mistakenly reported missed samples at startup, when no samples were actually missed. This problem has been resolved.

[RTI Bug # 13149]

4.7.2 Occasional Incomplete Discovery on Startup in rtiddsspy

The *rtiddsspy* utility sometimes did not discover all DDS entities at startup. This problem has been resolved.

[RTI Bug # 13215]

4.8 Fixes Related to QoS and Other Miscellaneous Fixes

4.8.1 Reliable Reader Configured with KEEP_LAST History May Have Stopped Receiving Samples while Reader Queue Full

A reliable *DataReader* configured with KEEP_LAST History, where **depth** * **max_instances** >= **max_samples**, may have stopped receiving samples while the reader queue was full (**max_samples** was reached). If a matching *DataWriter* was configured with KEEP_ALL History and limited **max_samples**, it would end up blocking. This problem has been resolved.

[RTI Bug # 13139]

4.8.2 Possible Reader Crash if max_samples_per_instance Exceeded on Reliable Reader and Matching Writer was Shutdown

If a *DataWriter* sent a sample to a reliable *DataReader* configured with KEEP_ALL_HISTORY_QOS and that sample could not be provided to the application because **max_samples_per_instance** was exceeded, a subsequent shutdown of the *DataWriter* may have caused the reader to crash.

This failure was more likely to appear if there were multiple writing applications that were repeatedly created and gracefully destroyed while the subscribing application was running.

This problem has been resolved.

[RTI Bug # 13148]

4.8.3 Potential Race Condition when Reader is Enabled

In the previous release, there was a potential race condition when a *DataReader* was enabled. This race condition may have caused samples to be received before the DataReaderListener's **on_subscription_matched()** callback was called.

The race condition may have also caused corrupted values in the statuses in the DataReaderListener's **on_requested_incompatible_qos()**, **on_liveliness_changed()** and **on_subscription_matched()** callbacks.

The race condition is resolved in this release. The DataReaderListener's **on_subscription_matched()** callback is now guaranteed to be called before any samples are received.

[RTI Bug # 12970]

4.8.4 Problem Getting Protocol or Cache Status within Reader/Writer Listener Callbacks

In the previous release, calling a *DataReader's* **get_datareader_protocol_status()**, **get_datareader_cache_status()**, or **get_matched_publication_datareader_protocol_status()** within a DataReaderListener callback, or calling a *DataWriter's* **get_datawriter_cache_status()**, **get_datawriter_protocol_status()**, **get_matched_subscription_datawriter_protocol_status()**, or **get_matched_subscription_datawriter_protocol_status_by_locator()** within a DataWriterListener callback, may have resulted in an error, such as:

```
COMMENDSrReaderService_getLocalReaderStatistics:!copyReadWriteArea
of local SRR stats PRESPsReader_getDataReaderProtocolStatus:!srr
getLocalReaderStatistics
DDS_DataReader_get_datareader_protocol_status:!getDataReaderProtocol
Status
```

This problem has been resolved.

[RTI Bug # 12982]

4.8.5 Error when Accessing Cache or Protocol Status for Disabled Writer or Reader

In the previous release, your application may have reported a "precondition error" or crashed if you attempted to access cache status (**get_<entity>_cache_status()**) for a *DataReader* or *DataWriter* that had not yet been enabled. This problem has been resolved.

[RTI Bug # 12997]

4.8.6 Failure When Allocating Buffers Larger Than 4 GB

The allocation of a buffer larger than 4 GB—whose size was controlled by resource limits of a participant, writer, or reader—would have failed in the previous release; this has since been resolved.

[RTI Bug # 12958]

4.8.7 Problem Creating a Publisher/Subscriber with create_*_with_profile() after Loading Profile with is_default_qos = true

In the previous release, the *DomainParticipant* operations **create_publisher_with_profile()** and **create_subscriber_with_profile()** would hang if a QoS Profile with **is_default_qos** = true was loaded. This problem has been resolved.

[RTI Bug # 13172]

4.8.8 Possible Deadlock when Calling get_qos_profile_libraries() and get_qos_profiles()

In the previous release if you were not loading an XML QoS Profile file, your application may have hung after calling the DomainParticipantFactory's **get_qos_profile_libraries()** or **get_qos_profiles()** operations. This problem has been resolved.

[RTI Bug # 13040]

4.8.9 Issues with XSD Schema Distributed with RTI Data Distribution Service

In the previous release, there were two issues with the XSD schema:

- The upper limit for **max_samples_per_read** was incorrect.
- **propagate_dispose_of_unregistered_instances** was not included.

These problems have been resolved.

[RTI Bug # 13041 and 13050]

4.8.10 Possible Memory Corruption when Copying PropertyQosPolicy

The operation that copies the PropertyQosPolicy may have produced the wrong result and lead to memory corruption.

The failure condition only happened when the length of the destination property was smaller than the length of the source property.

There were three user-visible manifestations of this bug:

- The PropertyQosPolicyHelper operation **get_properties()** may have produced the wrong result and caused the application crash.
- The operation **get_qos()** may have produced the wrong result and caused the application crash.
- In Java, the creation of multiple DDS entities of the same kind with different PropertyQosPolicy values may have caused the application to crash.

This problem has been resolved.

[RTI Bug # 13032]

4.8.11 Possible Segmentation Fault when Calling set_qos() Multiple Times after Setting PropertyQosPolicy to Different Values

In the previous release, if you called **set_qos()** multiple times on a *DataWriter*, *DataReader*, or *DomainParticipant*, and you changed the content of the PropertyQosPolicy each time, *RTI Data Distribution Service* may have crashed. This problem has been resolved.

[RTI Bug # 13034]

4.8.12 DomainParticipantFactoryQos_copy() May Have Incorrectly Returned OK Even if ProfileQosPolicy Copy Failed

If **DDS_DomainParticipantFactoryQos_copy()** encountered an error while copying DDS_ProfileQosPolicy, it mistakenly returned RETCODE_OK when it should have returned RETCODE_ERROR. This problem has been resolved.

[RTI Bug # 13150]

4.8.13 Changing Enabled DomainParticipant's QoS May Have Caused False Immutable QoS Error

If a *DomainParticipant* was created with a non-default setting for the any of the following QoS, then a subsequent change to any QoS value would fail. You would have seen an error stating that the QoS is immutable, even if this was not the QoS policy that you subsequently modified.

- ❏ **Database QoS shutdown_cleanup_period**
- ❏ **Resource Limits QoS remote_participant_allocation.incremental_count**
- ❏ **Resource Limits QoS resource_limits.writer_property_string_max_length**
- ❏ **Resource Limits QoS resource_limits.reader_property_string_max_length**

This problem has been resolved.

[RTI Bug # 13110]

4.8.14 Inconsistent QoS Error When Duration Set to Infinite

Duration settings (such as in the Lifespan QoS policy) have two parts: **sec** and **nanosec**. In the previous release, setting **sec** to INFINITE and **nanosec** to 0 (or any number) caused an error (Inconsistent QoS).

This problem has been resolved. Now, if **sec** is set to INFINITE, the value in **nanosec** is ignored.

For example, the following caused in Inconsistent QoS error:

```
<lifespan>
    <duration>
        <sec>DURATION_INFINITE_SEC</sec>
        <nanosec>0</nanosec>
    </duration>
</lifespan>
```

This problem has been resolved.

[RTI Bug # 12880]

4.8.15 'nanosec' Portion of Duration May Have Been Assumed as 0 when 'sec' was 0

Duration settings (such as **lease_duration** in the Liveliness QoS policy) have two parts: **sec** and **nanosec**. In the previous release, if you set **sec** to 0, the value of **nanosec** may have been ignored and assumed to be zero.

This problem has been resolved. Now, if you can set **sec** set to 0 and still specify a value for **nanosec**.

[RTI Bug # 12996]

4.8.16 Locator Kinds in Builtin Topic Data May Have Been Inconsistent

Previously, if you examined the locators in the builtin topic data, the locator kind values may have been inconsistent with the constants like DDS_LOCATOR_KIND_UDPv4, but it did not affect functionality. This was due to an inconsistency in how the kinds were stored internally.

This inconsistency has been corrected, so the locator kinds for locators in the discovery data match the constants.

[RTI Bug # 12894]

4.8.17 Some Logging Output Sent to Console Instead of File

If you called the NDDSConfigLogger's **set_output_file()** operation before the first participant was created, some initial output may have been sent to the console instead of to the log file. This problem has been resolved.

[RTI Bug # 12497]

5 Known Issues

5.1 Estimate of Message Overhead For Large Data May Be Exceeded

RTI Data Distribution Service underestimates the message overhead of samples with large data payloads (on the order of 64 KB). If not properly compensated by the transport's **message_size_max** property, the send operation will fail.

To work around this issue, **message_size_max** should be sized to account for this overhead. As a rule-of-thumb, 64 KB requires an additional 1024 bytes of overhead.

[RTI Bug # 11995]

5.2 Writer-side Filtering May Cause a Deadline to be Missed

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

[RTI Bug # 10765]

5.3 Disabled Interfaces on Windows Systems

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

However, if **NDDS_DISCOVERY_PEERS** only contains unicast addresses, the *DomainParticipant* will be successfully created even if all the interfaces are disabled. The cre-

ation of a *DataReader* will fail if its TransportMulticastQosPolicy contains a UDPv4 or UPDv6 multicast address.

5.4 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 Bug # 11362]

5.5 Incorrect Content Filtering for Valuetypes and Sparse Types

Content filters may not filter correctly if (a) the type is a valuetype or sparse type using inheritance and (b) the filters refer to members of a derived class.

This issue exists for Topics using DynamicData type support. It may also affect filtering of valuetypes using the .NET or C APIs, or CORBA-compatible C++ type plugins.

[RTI Bug # 12606]

5.6 Multi-dimensional Arrays of Sequences Not Supported when Generating .NET Code

The .NET code generated by *rtiddsgen* for multi-dimensional arrays of sequences is not correct and will not compile.

For example:

```
struct MyStruct {
    sequence<short, 4> m1[3][2];
};
```

[RTI Bug # 13088]

5.7 Best-Effort Writer-Side Content-Filtered Samples Reported As Lost

A sample that is content-filtered by a best-effort *DataWriter* may incorrectly trigger a *DataReader's* sample-lost status. Normally, best-effort *DataWriters* communicate to *DataReaders* which samples have been writer-side filtered by sending GAP submessages. However, in the case when either the *DataWriter's* liveliness lease-duration is infinite (**DDS_LivelinessQosPolicy.lease_duration**) or the liveliness kind is automatic (**DDS_LivelinesQosPolicy.kind** = DDS_AUTOMATIC_LIVELINESS_QOS), the GAP submessages are not sent. Consequently, a *DataReader* will not be notified of which samples have been filtered by the *DataWriter* and will report them as lost in its sample lost status.

[RTI Bug #13476]

5.8 License File May Not be Found if NDDSHOME Set with Quotation Marks

Some distributions of *RTI Data Distribution Service* require a license. One of the places where *RTI Data Distribution Service* may look for a license is the file **rti_license.dat** in the directory specified by the environment variable NDDSHOME. However, if you have set NDDSHOME using quotation marks to enclose the directory path (such as **"C:\Program Files\RTI\ndds.4.5c"**), the license file will not be found there. Instead, you should set NDDSHOME without using quotation marks (such as **C:\Program Files\RTI\ndds.4.5c**).

For more information on where *RTI Data Distribution Service* may look for a license file, see the *RTI Data Distribution Service Getting Started Guide*.

[RTI Bug #13477]

5.9 Issues with Dynamic Data

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

  [RTI Bug # 12647]

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

  [RTI Bug # 12855]

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

  [RTI Bug # 12897]