RTI Connext DDS

Core Libraries

What's New in

Version 5.2.0

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

Chapter 1 What's New in 5.2.0

This document highlights new features and improvements.

For details on fixed bugs, please see the *RTI Connext DDS Core Libraries Release Notes*.

1.1 New Platforms

This release adds support for the following target platforms:

Table 1.1 New Platforms

Operating System	CPU	Compiler or SDK	RTI Architecture Abbreviation
AIX 7.1	POWER class (32-bit mode)	IBM xlC_r for AIX v12.1	p7AIX7.1xlc12.1
Android 2.3 - 4.4	ARMv7a	gcc 4.8	armv7aAndroid2.3gcc4.8
		Java Platform, Standard Edition JDK 1.7 or 1.8	
INTEGRITY 11.0.4	P4080	Multi 6.1	p4080Inty11.devtree-fsl-e500mc.comp2012.1
		Multi 6.1.4	p4080Inty11.devtree-fsl-e500mc.comp2013.5.4
	x86	Multi 6.1.4	pentiumInty11.pcx86-smp
Mac OS X 10.10	x64	clang 6.0	x64Darwinclang6.0
Red Hat Enterprise Linux 6.5	x86	gcc 4.4.5	i86Linux2.6gcc4.4.5
	x64	gcc 4.4.5	x64Linux2.6gcc4.4.5

Table 1.1 New Platforms

Operating System	CPU	Compiler or SDK	RTI Architecture Abbreviation
Red Hat Enterprise Linux 7.0	x86	gcc 4.8.2	i86Linux3gcc4.8.2
	x64	gcc 4.8.2	x64Linux3gcc4.8.2
SUSE Linux Enterprise Server 11 SP3 (2.6 kernel)	x64	gcc 4.3.4	x64Linux2.6gcc4.3.4
VxWorks 6.9.4	PPC (e500v2)	gcc 4.3.3	For Kernel Modules: ppce500v2Vx6.9.4gcc4.3.3 For Real-Time Processes: ppce500v2Vx6.9.4gcc4.3.3_rtp
VxWorks 7.0	Pentium (32 bit)	gcc 4.3.3	For Kernel Modules: pentiumVx7.0gcc4.3.3 For Real Time Processes: pentiumVx7.0gcc4.3.3_rtp
Ubuntu 14	x86	gcc 4.8.2	i86Linux3gcc4.8.2
	x64	gcc 4.8.2	x64Linux3gcc4.8.2
Windows 8, 8,1	x86	Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2013	x64Win64VS2013
Windows Server 2012 R2	x86	Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2013	x64Win64VS2013

Please see the updated *RTI Connext DDS Core Libraries Platform Notes* for details on using these platforms.

1.2 Removed Platforms

These platforms are no longer supported:

- Fedora
- INTEGRITY 10.0.2 for p4080
- SELinux
- VxWorks MILS
- Windows platforms using Visual Studio 2005

1.3 Unified Directory Structure

This release unifies the directory structure across all Connext DDS tools and libraries. Some files have changed locations and may require you to update your build infrastructure to find scripts in the new location.

- **bin** - Scripts to run tools, services, and utilities. These scripts set up the environment correctly for RTI's applications.
- **doc** - Documentation for all installed products, including manuals and APIs
- **include** - Header files for all installed products
- **lib** - All Connext DDS libraries in the RTI SDK
    - java Location of jar files in the classpath of Java applications
    - <architecture> SDK libraries to link into your application.
        - Java libraries (libnddsjava.so/nddsjava.dll): We now ship one Java shared library for every supported compiler, instead of shipping a single version in a separate jdk folder.
        - .NET libraries: There is no longer a separate architecture folder for .NET architectures. VS2008 has .NET 2.0 libraries, VS2010 has .NET 4.0 libraries, VS2012 has .NET 4.5 libraries, and VS2013 has .NET 4.5.1 libraries
- **resource** - Location of XML files, IDL files, example templates, additional installers

1.4 Backup Process

When installing new RTI packages that may overwrite the contents of previous packages—for example, when installing a patch—the RTI Package Installer will create a zip file that contains a backup of all the files that are overwritten.

To avoid using unnecessary disk space, the installer will delete the previous backup file after the latest backup zip file is created. This means that there will only be one backup at a time.

To revert changes made by patching, you can unzip this file and copy files back to their original location. Note that this manual step will not be reflected in RTI Launcher, which will still show the last installation version.

1.4.1 Special Backup of RTI Libraries

When installing a new RTI package that overwrites the libraries in the **<NDDSHOME>/lib/<architecture>** directory, a backup will be created of all the previous libraries in the **<NDDSHOME>/lib/<architecture>** directory. That backup will be created in **<NDDSHOME>/lib/<architecture>/<current_installed_version>**.

For example, if you install a patch version 5.2.0.1 to the RTI core libraries for i86Win32VS2010, your 5.2.0 libraries will be copied into the following directory before the 5.2.0.1 libraries are installed: **<NDDSHOME>/lib/i86Win32VS2010/5.2.0**.

If you install another patch later, before overwriting the 5.2.0.1 libraries, they will be copied into **<NDDSHOME>/lib/i86Win32VS2010/5.2.0.1**.

1.5 Changes to Installation Process

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

To install these bundles, you will run the host bundle (such as rti_connext_dds-5.2.0-core-host-x64Linux.run). The installer will walk you through installing the host bundle. Then you will install your target(s). To do so, you can use the new RTI Package Installer utility that's available in RTI Launcher. This utility allows you to select one or more packages to install. Or you can install from the command line by using the **rtipkginstall** script in **<install directory>/bin**. For example, to install a target bundle from a command line:

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

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

1.6 Modern C++ API

This release includes a brand new C++ programming API: the RTI Connext DDS Modern C++ API. This API is based on the ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx) specification and contains the RTI extension features available in other languages. The “traditional” C++ API is still available.

The modern C++ API provides substantially different programming paradigms and patterns. The traditional API could be considered as simply "C with classes," while the modern API incorporates modern C++ techniques, most notably:

- Generic programming
- Integration with the standard library
- Automatic object lifecycle management, providing full value types and reference types
- C++11 support (for some platforms), such as move operations, initializer lists, and support for range for-loops.

RTI Code Generator includes two new language options that generate code for the new API: **-language C++03** and **-language C++11**. Using C++11 will generate a different example and include the flags to activate C++11 in your compiler, if needed. To use the traditional API, continue using **-language C++**.

When you run **<RTI Connext DDS installation directory>/bin/rtiddsgen -language C++11 -example <your architecture> Foo.idl**, you will get this example code:

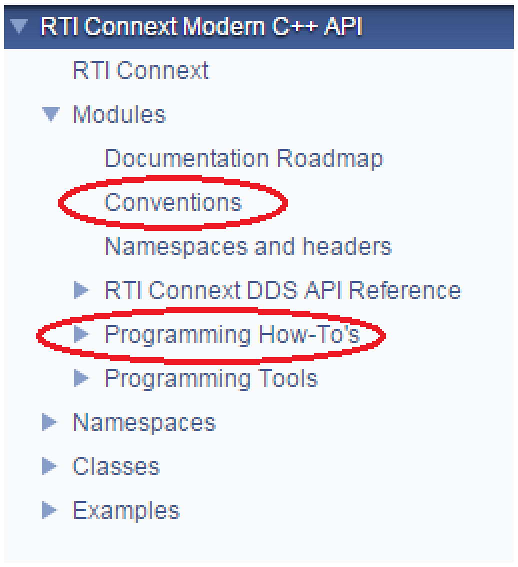
Foo_publisher.cxx (example publisher):

```
void publisher_main(int domain_id, int sample_count)
{
    // Create a DomainParticipant with default QoS
    dds::domain::DomainParticipant partipant(domain_id);

    // Create a Topic -- and automatically register the type
    dds::topic::Topic<Foo> topic (participant, "Example Foo");

    // Create a DataWriter with default QoS (Publisher created inline)
    dds::pub::DataWriter<Foo>
    writer(dds::pub::Publisher(participant), topic);

    Foo sample;
    for (int count = 0; count < sample_count || sample_count == 0;
         count++; {
        // Modify the data to be written here

        std::cout << "Writing Foo, count " << count << std::endl;
        writer.write(sample);
        rti::util::sleep(dds::core::Duration(4));
    }
}
```

Foo_subscriber.cxx (example subscriber):

```
// ...

void subscriber_main(int domain_id, int sample_count)
{
    // Create a DomainParticipant with default QoS
    dds::domain::DomainParticipant participant(domain_id);

    // Create a Topic == and automatically register the type
    dds::topic::Topic<Foo> topic(participant, "Example Foo");

    // Create a DataReader with default QoS (Subscriber created inline)
    dds::sub::DataReader<Foo>
        reader(dds::sub::Subscriber(participant), topic);

    // Create a ReadCondition for any data on this reader
    // and associate it with a handler
    int count = 0;
    dds::sub::cond::ReadCondition read_condition(
        reader,
        dds::sub::status::DataState::any(),
        [&reader, &count]()
    {
        // Take all samples
        dds::sub::LoanedSamples<Foo> samples = reader.take();

        for (auto sample : samples) {
            if (sample.info().valid()) {
                count++;
                std::cout << sample.data() << std::endl;
            }
        }
    } // The LoanedSamples destructor returns the loan
    );

    // Create a WaitSet and attach the ReadCondition
    dds::core::cond::WaitSet waitset;
    waitset += read_condition;

    while(count < sample_count || sample_count == 0) {
        // Dispatch will call handlers associated with waitset
        // conditions when they activate
        std::cout <<
            "Foo subscriber sleeping for 4 sec..." << std::endl;
        // wait up to 4s each time
        waitset.dispatch(dds::core::Duration(4));
    }
}
```

For more information about the Modern C++ API, see the API Reference HTML documentation:



In the *RTI Connext DDS Core Libraries User's Manual*, the following sections describe aspects of the API that differ significantly with respect to the other RTI Connext DDS language APIs:

- Section 3.3 Creating User Data Types with IDL
- Section 3.7 Interacting Dynamically with User Data Types
- Section 3.8 Working with DDS Data Samples
- Section 4.1.1 Creating and Deleting DDS Entities
- Section 7.4 Using DataReaders to Access Data (Read & Take)

Buildable source code examples are available from the RTI Community Portal (https://-community.rti.com/kb/modern-c-api-code-examples)

1.7 Changes to Connext DDS Java Packaging on Windows Platforms

In previous releases of Connext DDS, all Java applications for Windows platforms depended on the Visual Studio® 2005 libraries, regardless of which Windows platform bundle you installed.

In this release, the version of Visual Studio libraries that the Connext DDS Java library depends on is based on the specific Visual Studio version noted (in the Platform Notes) for your Windows target architecture, as long as the correct version of Connext DDS libraries path is in your PATH environment variable.

To debug with the debug version of a Connext DDS DLL, you still need to have the full Visual Studio package installed on your system. Otherwise, you only need the Visual Studio Redistributable C++ libraries installed.

1.8 Support for Custom Content Filters in .NET API

This release adds support for custom content filters in the .NET API. To implement a custom filter, create a class that implements one of the following interfaces:

- IContentFilter: This is the minimum API that must be implemented by a custom filter
- IWriterContentFilter: This interface provides a set of APIs that allows you to implement scalable filters on the DataWriter side

For more information, please see the *RTI Connext DDS Core Libraries User's Manual* (Section 5.4.8, Custom Content Filters) and the API Reference HTML documentation.

1.9 Improved Liveliness QoS Policy Behavior

This release introduces changes to the LivelinessQosPolicy behavior to make it more robust and configurable. In particular, this release introduces two main changes to Liveliness:

- The LivelinessQosPolicy has a new field, **assertions_per_lease_duration**. This parameter allows you to configure the rate at which assertions are sent to remote entities when the liveliness **kind** is DDS_AUTOMATIC_LIVELINESS_QOS. For details, see the *RTI Connext DDS Core Libraries User's Manual* (Section 6.5.13 LIVELINESS QosPolicy).
- Liveliness assertion messages are now sent using best-effort reliability (instead of using reliable reliability). This change makes liveliness assertions more predictable and easier to configure.

  It is possible to go back to the old behavior by using the field **participant_message_reader_reliability_kind** in the *DomainParticipant*'s DiscoveryConfigQosPolicy. To ensure backward compatibility with previous releases, the participant will automatically switch back to Reliable Liveliness messages when communicating with a DDS Participant from a version prior to 5.2.0.

  For details, see the *RTI Connext DDS Core Libraries User's Manual* (Section 8.5.3 DISCOVERY_CONFIG QosPolicy (DDS Extension)).

1.10 Support for Unbounded Built-in Types in C, C++, and .NET APIs

This release adds support for unbounded built-in types in the C, C++, and .Net APIs.

To configure unbounded support, set the properties **dds.builtin_type.*.max_size** and **dds.builtin_type.*.alloc_size** to 2147483647.

When unbounded support is configured, the middleware will not preallocate the *DataReader* queue's samples to their maximum size. Instead, it will deserialize incoming samples by dynamically allocating and deallocating memory to accommodate the actual size of the sample value.

In addition to setting the properties **dds.builtin_type.*.max_size** and **dds.builtin_type.*.alloc_size** to 2,147,483,647, you must also use the threshold QoS properties **dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size** on the *DataWriter* and **dds.data_reader.history.memory_manager.fast_pool.pool_buffer_max_size** on the **DataReader**.You must also set the QoS value **reader_resource_limits.dynamically_allocate_fragmented_samples** on the *DataReader* to true.

For additional information on these QoS values, see the *RTI Connext DDS Core Libraries User's Manual*.

1.11 Support for External Hardware Load Balancers in TCP Transport Plugin

For two Connext DDS applications to communicate, the TCP Transport plugin needs to establish 4-6 connections between the two communicating applications.

In previous releases, the TCP Transport plugin did not support external load balancers. This was because external load balancers did not forward the traffic to a unique TCP Transport Plugin server, but they divided the connections among multiple servers. Because of this behavior, when an application running a TCP Transport plugin client tries to establish all the connections to an application running a TCP Transport plugin server, the server may not receive all the required connections.

This release adds a new property to NDDS_Transport_TCPv4_Property_t, **negotiate_session_id**. By default, this property is set to FALSE. When set to TRUE, the TCP Transport Plugin will perform a session negotiation that will help external load balancers identify all the connections associated with a particular session between two Connext DDS applications. This keeps the connections from being divided among multiple servers and ensures proper communication.

For more information, see the *RTI Connext DDS Core Libraries User's Manual* (Support for External Hardware Load Balancers in TCP Transport Plugin).

1.12 Connection Liveliness Feature in TCP Transport Plugin

The TCP Transport plugin now supports a new ‘connection-liveliness’ feature. This feature provides a way to detect the disconnection of a connection without relying on notification from the operating system (which may take several minutes, depending on the scenario and OS configuration).

This feature is useful for systems running TCP Transport plugin clients on hosts that do not support the keep-alive or user-timeout features. For details, see the *RTI Connext DDS Core Libraries User's Manual's* section on TCP/TLS Transport Properties.

Enabling this feature will break backward compatibility with TCP Transport plugins that do not include this feature. To enable this feature, use the **connection_liveliness property**, as in the following example:

```
<element>
    <name>
    dds.transport.TCPv4.tcp1.connection_liveliness.enable
    </name>
    <value>1</value>
    <propagate>false</propagate>
</element>
```

1.13 Full Support for Windows I/O Completion Ports with TLS

Connext DDS 5.1.0 added partial support of Windows I/O Completion Ports when using the TLS transport. In particular, the **force_asynchronous_send** property was not supported.

This release provides full support of Windows I/O Completion Ports. Now you can enable **force_asynchronous_send** while using the TLS transport with Windows I/O Completion Ports socket monitoring.

1.14 Added TCP USER TIMEOUT Support to Linux Architectures

The TCP Transport plugin now supports the Linux OS's TCP User Timeout socket option. For details, please see the *RTI Connext DDS Core Libraries User's Manual's* section on TCP/TLS Transport Properties.

This new feature can be enabled through the **user_timeout** property as in the following example:

```
<element>
       <name>dds.transport.TCPv4.tcp1.user_timeout</name>
       <value>5</value>
       <propagate>false</propagate>
</element>
```

1.15 TCP Transport's keep_alive_time Property Now Supported on Mac Platforms

This release adds support on Mac platforms for the TCP transport property, **keep_alive_time** (see Table 36.1, Properties for NDDS_Transport_TCPv4_Property_t, in the RTI Connext Core Libraries User's Manual). This property was previously only available for Linux platforms.

1.16 Improved TCP Transport Plugin Robustness Against Unexpected Control Messages

In previous releases, the TCP Transport plugin would shutdown upon receipt of an unexpected control message. In this release, the TCP Transport plugin is more robust. In particular, if the plugin receives an unexpected control message, it will print an error message and close the associated connection, but it will not trigger a shutdown.

1.17 Logging Level for TCP Transport Windows IOCP Connection-Reset Errors Changed from Exception to Warning

When enabling IOCP monitoring with the property **socket_monitoring_kind**, the TCP Transport plugin logging was too verbose when reporting disconnection errors like the following:

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

In this release, the logging verbosity for those errors has been changed from Exception to Warning.

1.18 Improved Logging of Precondition Errors from TCP Transport Plugin in Debug Mode

Starting with this release, when using the TCP Transport plugin in debug mode, any logged precondition errors will also include the failing precondition expression.

1.19 Partial Support for DurabilityServiceQosPolicy's service_ cleanup_delay

This release includes the ability to purge instances from *Persistence Service.* The **service_cleanup_delay** field of the DurabilityServiceQosPolicy controls when *Persistence Service* is able to remove all information regarding a data instance. The currently supported values for **service_cleanup_delay** are zero or INFINITE. The default **service_cleanup_delay** value is 0, meaning that when an instance is disposed, it will be purged from the persistence service immediately. This will only happen if *Persistence Service* has been configured with **use_durability_service=true**. A value of INFINITE disables the purging of disposed instances.

1.20 Option to Release Resources Associated with Disposed Instance

A new feature offers a way to release the resources associated with a disposed instance in both *DataWriters* and *DataReaders*. When applied to a *DataWriter*, historical samples are also removed potentially saving bandwidth usage.

This feature is enabled in a *DataWriter* through a new field in the WriterDataLifecycleQosPoliy: **autopurge_disposed_instances_delay**. When this feature is enabled, the middleware will clean up all the resources associated with a disposed instance (most notably, the sample history of non-volatile *DataWriters*) when all the instance's samples have been acknowledged by all its live *DataReaders*, including the sample that indicates the disposal.

By default, **autopurge_disposed_instances_delay** is disabled (the delay is INFINITE). If the delay is set to zero, the *DataWriter* will clean up as soon as all the samples are acknowledged after the call to **dispose ()**. A non-zero value is currently not supported. This feature is supported in both the ODBC and in-memory writer-history configurations.

In a *DataReader*, this feature is enabled through a field in ReaderDataLifecycleQosPolicy with the same name. The **autopurge_disposed_instances_delay** in the ReaderDataLifecycleQosPolicy also currently only supported values of zero or INFINITE, with INFINITE being the default. If the delay is set to zero, instances that have been disposed and have no outstanding unread samples, including the dispose sample itself, will be immediately purged from the *DataReader's* queue. The default value of INFINITE disables this feature, and instances will be purged from the *DataReader's* queue under the same conditions as they have been in previous releases. See Section 7.6.3 READER_DATA_LIFECYCLE QoS Policy in the *User's Manual* for a description of when resources associated with samples and instances in the *DataReader* queue can be reclaimed.

1.21 Support for Application-Level Acknowledgment with Response Data

These release adds the ability to add response data to these DataReader operations:

- **acknowledge_sample()**
- **acknowledge_all()**

The response data is provided as a sequence of octets. The maximum size is configurable using the **max_ app_ack_response_length** in the DataReaderResourceLimitsQosPolicy.

For additional information, see the *RTI Connext DDS Core Libraries User's Manual* (Section 7.4.4, Using DataReaders to Access Data (Read & Take)) and the API Reference HTML documentation.

1.22 New DataWriter Status to Receive Notification when Sample is Application-Level Acknowledged

This release includes a new *DataWriter* status to receive notification when a sample is application-level acknowledged by a DataReader. This status triggers a new *DataWriter's* listener callback named **on_ application_acknowledgment()**.

For more information, see the *RTI Connext DDS Core Libraries User's Manual* and the API Reference HTML documentation.

1.23 Ability to See if Sample has been Application-Acknowledged

There is a new *DataWriter* operation, **is_sample_app_acknowledged()**. You can use it to see if a sample has been application-acknowledged by all matching *DataReaders* that were alive when the sample was published.

If a *DataReader* does not enable application acknowledgment (by setting the ReliabilityQosPolicy's **acknowledgment_kind** to a value other than DDS_PROTOCOL_ACKNOWLEDGMENT_MODE), the sample is considered application-acknowledged for that *DataReader*.

1.24 Ability to Prevent Invocation of on_application_acknowledgment () when Response Data Empty or Invalid

The DataWriterProtocolQosPolicy contains a new field called **propagate_app_ack_with_no_response**. When this field is set to FALSE, the callback **on_application_acknowledgment()** will not be invoked if the sample being acknowledged has an empty or invalid response. The default setting is TRUE.

1.25 Performance Optimizations in Application-Level Acknowledgment Protocol

This release introduces significant performance optimizations in the application-level acknowledgment protocol.

1.26 Ability to Provide Threads to Connext DDS in C/C++

Applications can now provide the threads needed by the middleware (i.e., receiving threads, database thread, etc.). By default, these threads are created by Connext DDS using a specific framework and are configured via QoS. A new interface, **ThreadFactory**, can be implemented and plugged into DomainParticipants to create the required threads by Connext DDS. This gives applications full control over how these threads are created and managed. The new APIs are available only in C/C++.

1.27 New TypeSupport Operations in Built-in Types to Serialize Sample into Buffer and Deserialize Sample from Buffer

This release provides two new TypeSupport operations in the Built-in Types to serialize a sample into a buffer and deserialize a sample from a buffer. The sample serialization/deserialization uses CDR representation. This feature is supported in the following languages: C, C++, Java, and .NET.

For example, for the Octets built-in type these operations are:

C:

```
DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer(...)
DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer(...)
```

C++

```
DDS::OctetsTypeSupport::serialize_data_to_cdr_buffer(...)
DDS::OctetsTypeSupport::deserialize_data_from_cdr_buffer(...)
```

Java:

```
import com.rti.dds.type.builtin.BytesTypeSupport;
BytesTypeSupport.get_instance().serialize_to_cdr_buffer(...)
BytesTypeSupport.get_instance().deserialize_from_cdr_buffer(...)
```

C++/CLI:

```
DDS::BytesTypeSupport::serialize_data_to_cdr_buffer(...)
DDS::BytesTypeSupport::deserialize_data_from_cdr_buffer(...)
```

C#:

```
using DDS;
BytesTypeSupport.serialize_data_to_cdr_buffer(...)
BytesTypeSupport.deserialize_data_from_cdr_buffer(...)
```

1.28 New DynamicData Operations to Serialize Sample into Buffer and Deserialize Sample from Buffer—C/C++ APIs Only

This release provides two new DynamicData operations to serialize a DynamicData sample into a buffer and deserialize a DynamicData sample from a buffer:

- **to_cdr_buffer()**
- **from_cdr_buffer()**

These operations are only supported in the C and C++ languages. For more information, see the C or C++ API Reference HTML documentation.

1.29 Out-of-Order Type Definitions in XML Configuration File are Now Allowed

This release allows you to define types and constants out-of-order in the XML configuration files. For example, now you can have this:

```
<struct name="Structure1">
    <member type="boolean" name="m1"/>
    <member type="nonBasic" nonBasicTypeName="Structure2"
     name="m2"/>
</struct>


<struct name="Structure2">
    <member type="boolean" name="m1"/>
</struct>
```

In previous releases, the above XML would have caused a parsing error.

1.30 Ability to Add Metadata Flags to Samples

This release adds the ability to add flags to a sample.

The DDS_WriteParams_t structure, used by the *DataWriter's* **write_w_params()** operation, includes a new field named **flag**, which can be used to set the sample flags. On the *DataReader* side, the flags can be inspected using the field **flag** in the DDS_SampleInfo structure.

RTI reserves the first eight least-significant bits for middleware-specific usage. Of these eight bits, four are already used:

- REDELIVERED_SAMPLE (Bit 1): This bit is used by RTI Queuing Service to mark a sample as redelivered.
- INTERMEDIATE_REPLY_SEQUENCE_SAMPLE (Bit 2): With the Request-Reply communication pattern, this bit can be used to indicate that a response sample is not the last one for a given request. When a request generates multiple responses from a Replier, the Replier should mark all the responses except the last one as INTERMEDIATE_REPLY_SEQUENCE_SAMPLE. If the replier is a *DataWriter*, this flag can be set by updating the **flag** member of the DDS_WriteParams_t parameter that is passed into the *DataWriter's* **write_w_params()** operation. If the replier is a Connext DDS Replier, this flag can be set in the WriteSample provided to the Replier's **send_reply()** operation.
- DDS_REPLICATE_SAMPLE (Bit 3): Indicates if a sample must be broadcast by one RTI Queuing Service replica to other replicas.
- DDS_LAST_SHARED_READER_QUEUE_SAMPLE (Bit 4): Indicates that a sample is the last sample in a SharedReaderQueue for a QueueConsumer *DataReader*.

For more information, please see the *RTI Connext DDS Core Libraries User's Manual* and the API Reference HTML documentation.

1.31 Ability to Enable Manual Endpoint Discovery for Individual Participants

This release adds a new field to the DiscoveryQosPolicy, **enable_endpoint_discovery**. By default, this field is set to TRUE, meaning endpoint discovery is automatically performed with all discovered participants. When set to FALSE, endpoint discovery is initially disabled and a call to the DomainParticipant's new **resume_endpoint_discovery()** operation is required to enable endpoint discovery for a given discovered participant. For more information, see the updated RTI Connext DDS Core Libraries User's Manual (Section 16.4.5, Supervising Endpoint Discovery).

1.32 Ability to Configure Memory Allocation for Instance Keys in DataWriter and DataReader Queues

This release introduces configuration settings that allow more flexible memory-allocation schemas for the key that is stored with every instance in *DataWriter* and *DataReader* queues.

In previous releases, the Connext DDS core pre-allocated the memory for keys in the *DataWriter* and *DataReader* queues. Although this memory allocation policy is suitable for real-time systems where determinism and predictability are key, it leads to higher memory usage.

With the new configuration settings, you can control when to use pre-allocation versus dynamic memory allocation from the heap.

For more details on these configuration parameters, see Chapter 20, Sample-Data and Instance-Data Memory Management, in the *RTI Connext DDS Core Libraries User's Manual* (Sections 20.3 and 20.4).

1.33 Ability to Configure Replacement Policy for Remote Participants Ignored by DomainParticipant

Connext DDS provides a way to ignore remote entities by invoking any of the following *DomainParticipant* operations: **ignore_participant()**, **ignore_publication()**, and **ignore_subscription()**.

When an entity is ignored, Connext DDS adds it to an internal 'ignore' table. The resource limits for this table are configured using the DomainParticipantResourceLimitsQosPolicy's **ignored_entity_allocation**. Every time Connext DDS receives a message from an ignored entity, it will check this table; if the entity is ignored, it will filter the message.

In previous releases, the ignore operation failed if **ignored_entity_allocation.max_count** was exceeded.

This release adds a new field to the DomainParticipantResourceLimitsQosPolicy, **ignored_entity_replacement_kind**. By default, this field is set to **DDS_NO_REPLACEMENT_IGNORED_ENTITY_ REPLACEMENT**, meaning that a call to the *DomainParticipant's* **ignore_participant()/ignore_publication()/ignore_subscription()** operations will fail if the *DomainParticipant* has ignored more entities than the limit set in **ignored_entity_allocation.max_count**.

When **ignored_entity_replacement_kind** is set to **DDS_NOT_ALIVE_FIRST_IGNORED_ ENTITY_REPLACEMENT**, a call to **ignore_participant()** will not fail when **ignored_entity_allocation.max_count** is exceeded, as long as there is one DomainParticipant already ignored. Instead, the call will replace one of the existing *DomainParticipants* in the internal table. The remote DomainParticipant that will be replaced is the one for which the local *DomainParticipant* had not received any message for the longest time.

When a remote *DomainParticipant* is replaced in the 'ignore' table, it becomes un-ignored. Thus, the *DomainParticipant* would have to call to **ignore_participant()** again to re-ignore the replaced entity.

Note that in this release ignored publications and subscriptions are never replaced in the 'ignore' table. Since this table also contains the ignored DomainParticipants, a call to **ignore_participant()** will fail if **ignored_entity_allocation.max_count** is reached and none of the ignored entities is a DomainParticipant.

The following XML snippet shows how to configure the new replacement policy:

```
<qos_profile name="IgnoredEntityReplacement_Profile">
    <participant_qos>
        <resource_limits>
          <ignored_entity_replacement_kind>
               NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT
          </ignored_entity_replacement_kind>
          <!-- Very restrictive example, only 1 ignored entity allowed -->
          <ignored_entity_allocation>
               <initial_count>1</initial_count>
```

```
                <incremental_count>0</incremental_count>
                <max_count>1</max_count>
            </ignored_entity_allocation>
          </resource_limits>
      </participant_qos>
</qos_profile>
```

For more information, please see the *RTI Connext DDS Core Libraries User's Manual* (Section 16.4.4, Resource Limits Considerations for Ignored Entities) and the API Reference HTML documentation.

1.34 Ability to Retrieve PropertyQosPolicy Values for Remote Entities Outside of Built-In Topic Callbacks

In previous releases, the only way to retrieve the PropertyQosPolicy value of a remote entity (*DataWriter-/DataReader/DomainParticipant*) was within corresponding built-in topic callbacks.

In this release, the properties can be retrieved at any time by invoking the DataWriter's **get_matched_subscription_data()**, the *DataReader's* **get_matched_subscription_data()**, and the DomainParticipant's **get_discovered_participant_data()** operations.

1.35 Exception Messages now include Underlying Errors—JAVA, .Net, C++ APIs Only

Previously when the Java, .Net, or C++ APIs threw an exception, the exception message was empty. Now those messages will include the underlying errors.

1.36 Host ID Automatically Generated when no IP Addresses Available

Starting with this release, when no interface IP addresses are available, the host ID will be automatically generated (by configuring the WireProtocol QoS policy's rtps_host_id to DDS_RTPS_AUTO_ID).

1.37 New API to Get Serialized Size for a Given TypeObject

A new API, **get_type_object_serialized_size()** (for a TypeCode), allows you to get the serialized size of the TypeObject. The default buffer size used for storing a TypeObject is 3,072 bytes. For a larger TypeObject, this API can be used to determine the size that needs to be set in the **type_object_max_serialized_length** field of the DomainParticipantResouceLimitsQosPolicy.

1.38 New Field in DataReaderResourceLimitsQosPolicy: keep_minimum_state_for_instances

A new field in the DataReaderResourceLimitsQosPolicy, **keep_minimum_state_for_instances**, indicates whether or not a minimum state will be kept for deleted instances. This minimum state is used to support features and services such as multi-channel *DataWriters*, Durable Reader State and Persistence Service.

When this field is set to TRUE (the default), the minimum state will be kept for up to **max_total_instances** (another DataReaderResourceLimitsQosPolicy resource limit). When set to FALSE, no minimum state will be kept for instances which are purged from the *DataReader*.

Instances are purged from a *DataReader* in two cases:

1. When there are no more known *DataWriters* for an instance and no samples for that instance in the *DataReader's* queue.
2. When an instance has been disposed, there are no more samples for it in the *DataReader's* queue and the ReaderDataLifecycleQosPolicy's **autopurge_disposed_instances_delay** has been set to a finite duration.

1.39 New Field in DataReaderProtocolQosPolicy: propagate_unregister_of_disposed_instances

A new field in the DataReaderProtocolQosPolicy, **propagate_unregister_of_disposed_instances**, indicates whether or not an instance can move to the DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE state without being in the DDS_ALIVE_INSTANCE_STATE state. When set to TRUE, the *DataReader* will receive 'unregister' notifications even if the instance is not alive.

1.40 New Method get_participants() for DomainParticipantFactory in C/C++

There is a new method, **get_participants()**, which returns a sequence of pointers to all the DomainParticipants within the DomainParticipantFactory.

1.41 New QoS Policy to Mark DataReaders and DataWriters as Part of Infrastructure Service

This release adds a new QoS policy named **ServiceQosPolicy**. It is used to mark *DataWriters* and *DataReaders* as part of an infrastructure service. User applications should not modify this policy's value.

The possible values for this policy are:

- DDS_NO_SERVICE_QOS
- DDS_PERSISTENCE_SERVICE_QOS
- DDS_QUEUING_SERVICE_QOS
- DDS_ROUTING_SERVICE_QOS
- DDS_RECORDING_SERVICE_QOS
- DDS_REPLAY_SERVICE_QOS
- DDS_DATABASE_INTEGRATION_SERVICE_QOS

An application can determine the kind of service associated with a discovered *DataWriter* and *DataReader* by looking at a new field named **service** in the PublicationBuiltinTopicData and SubscriptionBuiltinTopicData structures.

For more information, see the updated *RTI Connext DDS Core Libraries User's Manua*l, Section 6.5.21 (ServiceQosPolicy).

1.42 Support for source_guid and related_source_guid

1.42.1 New Fields in DDS_WriteParams_t and SampleInfo

There are new fields in DDS_WriteParams_t, which is used by the **write_w_params()** operation: **related_reader_guid, source_guid** and **related_source_guid**.

There are corresponding new fields in DDS_SampleInfo, which is available when you read/take samples: **related_subscription_guid**, **source_guid**, and **related_source_guid**.

1.42.2 related_reader_guid, and related_subscription_guid

The value of the **related_reader_guid** field identifies a *DataReader* that is logically related to the sample that is being written.

A *DataReader* can inspect the **related_reader_guid** of a received sample by inspecting the content of the **related_subscription_guid** field in the SampleInfo structure.

The main use-case for this field is point-to-point sample distribution using ContentFilteredTopics. *DataReaders* install a ContentFilteredTopic on this metadata field using a unique GUID. For example:

```
@related_reader_guid.value = &hex(00000000000000000000000000000001)
```

Then a *DataWriter* that wants to send the sample to *DataReader* 'n' will call **write_w_params()** and set the field **related_reader_guid** in DDS_WriteParams_t to the value used by *DataReader* 'n' in its filter expression. RTI Queuing Service uses this field to distribute a sample to only one of the Consumer's *DataReaders* attached to a SharedReaderQueue.

For more information, see the *RTI Connext DDS Core Libraries User's Manual* (Table 6.16, DDS_ WriteParams_t and Table 7.18, DDS_SampleInfo).

1.42.3 source_guid and related_source_guid

The new **source_guid** field in DDS_WriteParams_t identifies the application logical data source associated with the sample being written.

The new **related_source_guid** identifies the application logical data source that is related to the sample being written.

A *DataReader* can inspect the **source_guid** and **related_source_guid** of a received sample by inspecting the content of the fields **source_guid** and **related_source_guid** in the SampleInfo structure.

The **source_guid** and **related_source_guid** fields are used by RTI Queuing Service in a request/reply scenario to direct a response to the QueueProducer that generated the request. In this scenario, the QueueProducer's *DataWriter* sends requests by setting the **source_guid** to a unique value. This value is always the same for a QueueProducer even if it is restarted. The QueueProducer's *DataReader* receiving responses install a CFT on the **related_source_guid**.

For more information, see the *RTI Connext DDS Core Libraries User's Manual* (Table 6.16, DDS_ WriteParams_t and Table 7.18, DDS_SampleInfo).

1.43 Transport Priority Configurable for Built-in and User-Created DataReaders

Previously, transport priority was not configurable for built-in DataReaders or user-created DataReaders. Now, the **metatraffic_transport_priority** field in the DiscoveryQosPolicy configures the transport priority of all data sent from built-in DataWriters and DataReaders too. Also, the TransportPriorityQosPolicy has been added to the DataReaderQos to configure the transport priority of all messages sent from user-created DataReaders.

> The TransportPriorityQosPolicy is only supported on a subset of the available platforms, please refer to the RTI Connext DDS Core Libraries Platform Notes for which platforms support this QoS policy.

1.44 Improved Content Filter Evaluation Performance for Types Containing Sequences and Unions

ContentFilteredTopics containing sequences with a large maximum length or complex unions evaluate much faster than in the previous release, 5.1.0.

Before, the maximum length of the sequence determined the order of complexity of the filtering algorithm; now the actual length of each sample determines the order of complexity. As for unions, the complexity now depends on the currently selected member (instead of the most-complex one as happened before).

This performance improvement applies to the following situations:

- Applications using DynamicDataReaders
- Java applications
- Content filters in Routing Service

C, C++, and .NET *rtiddsgen*-generated *DataReaders* were not affected.

1.45 Improved Memory Usage of Content Filters of Types Containing Strings In Some Cases

ContentFilteredTopics containing strings with a large maximum length consume less memory now (note: this was a regression in 5.1.0).

Before, the memory usage was determined by the maximum length of the string; now memory usage is determined by the actual length of each sample.

This memory-usage improvement applies to the following situations:

- Applications using DynamicDataReaders
- Java applications
- Content filters in RTI Routing Service

C, C++ and .NET *rtiddsgen*-generated *DataReaders* were not affected.

1.46 Improved Performance for Key-Only QueryCondition set_query_ parameters()

This release reduces the CPU consumption of the QueryCondition **set_query_parameters()** operation for key-only QueryConditions. This improvement is more significant when the *DataReader* is receiving samples from a high number of instances.

1.47 Reader-Side Performance Improvements when Removing Association with Remote Writer

The algorithm used to remove a remote, keyed *DataWriter* from a *DataReader* has been improved to be more efficient. When a *DataReader* detects that a *DataWriter* has left the system, it purges information about the *DataWriter* and the instances that it was writing. The algorithm that performed that purge was inefficient and if the *DataWriter* had a finite lifespan for its data and had written many instances, this action could take a long time, delaying the *DataReader* from otherwise continuing to process incoming data.

1.48 Monitoring Libraries and Distributed Logger now Part of Connext DDS Bundle

Libraries for Monitoring and Distributed Logger are now shipped as part of the Connext DDS bundle. Everyone who has access to the Connext DDS core libraries now also has access to the Monitoring and Distributed Logger libraries—you no longer need to install separate bundles to obtain these libraries.

1.49 Priority Inheritance used when Creating Semaphores on VxWorks Platforms

Starting with Connext DDS 5.2.0, semaphores are created with priority inheritance on VxWorks Platforms.

1.50 New Default Value for DiscoveryConfig Built-in Writer autopurge_unregistered_instances_delay

The default value for **autopurge_unregistered_instances_delay** in the DiscoveryConfigQosPolicy's .**publication_writer_data_lifecycle** and **subscription_writer_data_lifecycle** fields has been changed from an INFINITE duration to 0. This new default eliminates a possible unbounded memory growth in applications where *DataWriters* may enter and exit a system frequently.

1.51 New Default Value for DDS_DynamicProperty_t's buffer_max_size

The default setting for the DynamicData property **buffer_max_size** has changed from 65,536 to DDS_LENGTH_UNLIMITED in order to provide a better out-of-the-box experience. The old setting did not work with samples greater than 65 KB.

1.52 New Defaults for DataReaderResourceLimits' dynamically_allocate_fragmented_samples and max_fragments_per_sample

Starting with Connext DDS5.2.0, the default values for **dynamically_allocate_fragmented_samples** and **max_fragments_per_sample** in the DataReaderResourceLimitsQosPolicy and DDS_BuiltinTopicReaderResourceLimits_t have changed.

- The default value for **dynamically_allocate_fragmented_samples** has changed from FALSE to TRUE in the DataReaderResourceLimitsQosPolicy (it was already TRUE by default in DDS_BuiltinTopicReaderResourceLimits_t).
- The default value for **max_fragments_per_sample** has changed from 512 to unlimited in both the DataReaderResourceLimitsQosPolicy and the DDS_BuiltinTopicReaderResourceLimits_t.

These changes have been made in order to provide a better out-of-the-box experience. By default, the middleware will no longer allocate memory up-front for storing fragments; instead it will allocate memory from the heap when it receives the first fragment of a new sample.

1.53 Ability to Extend Internal CdrInputStream and CdrOutputStream Classes by Inheritance in Java

The internal classes **CdrInputStream** and **CdrOutputStream** no longer include the 'final' modifier. This change allows you to extend these classes by inheritance.

The type-plugin deserialization and serialization functions use these classes to read and write from/to the CDR buffer. You can extend their behavior by inheriting from them.

(This is an advanced, undocumented use case.)

1.54 Warning Logged when Setting Non-NULL Listener with STATUS_MASK_NONE

If a **create_<dds_entity>(), create_<dds_entity>_with_profile()**, or **set_listener()** operation is called with a non-NULL Listener and STATUS_MASK_NONE, Connext DDS should have logged a warning. In this release, a warning is logged with this message:

```
"Warning: setting a listener with STATUS_MASK_NONE will disable all callbacks"
```

1.55 Java Libraries Tested with Java 1.8

In this release, the Java libraries have been tested with JDK 1.8 (in addition to JDK 1.7).

1.56 Deprecated Platforms

Connext DDS 5.2.0 will be the last release that supports the platforms in Table 1.2 Deprecated Platforms. Please contact the RTI Sales team if you have any questions.

Table 1.2 Deprecated Platforms

Operating System	CPU	Compiler or SDK	RTI Architecture Abbreviation
Yellow Dog 4.0	PPC 74xx	gcc 3.3.3	ppc7400Linux2.6gcc3.3.3
Vista, 2003, XP Pro SP2	x86	VS 2008 SP1	i86Win32VS2008
Vista x64, 2003 x64, XP Pro SP2 x64	x64	VS 2008 SP1	x64Win64VS2008

1.57 Sparse Value Types Deprecated

Sparse value types have been deprecated in this release. Users are encouraged to use Mutable Types and Optional Members in place of sparse value types. Please see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types* for details on Mutable Types and Optional Members.

1.58 Separate Documentation for Code Generator

Details on RTI Code Generator, *rtiddsgen*, now appear in a separate *RTI Code Generator User's Manual*. There is also a separate *RTI Code Generator Release Notes* document.