RTI Connext DDS

Core Libraries

What's New in Version 6.0.1

Printed in U.S.A. First printing.
March 2020.

Trademarks

Copy and Use Restrictions

Technical Support

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

Contents

What's New in 6.0.1

RTI® Connext® DDS 6.0.1 is a maintenance release based on feature release 6.0.0. This document highlights new platforms and improvements in 6.0.1. These enhancements have been made since 6.0.0.

For what's fixed in 6.0.1 and 6.0.0, see the *RTI Connext DDS Core Libraries Release Notes*. For what's new in 6.0.0 and previous releases, see the *What's New* documents provided with those releases.

> **Note:** For backward compatibility information between 6.0.1 and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

1 Platforms

1.1 Added platforms

This release adds support for these platforms:

- Android™ 9 (Arm® v7, Arm v8 64-bit)
- macOS® 10.14 (x64)
- Red Hat® Enterprise Linux® 8 (x64)
- VxWorks® 6.9.4.11 (Arm v7) (Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.)
- VxWorks 7 SR0540 (x86) (Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.)
- Wind River® Linux 8 (PPC e6500) (Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.)
- Windows® 10 (x86, x64) with Visual Studio® 2019
- Windows Server 2016 (x86, x64) with Visual Studio 2019

- Yocto Project® 2.5 (Arm v7) (Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.)

1.2 Removed platforms

- macOS 10.11
- Windows 7
- Windows Server 2008 R2

2 Usability and Debuggability

2.1 Updates to the builtin QoS Profiles

Connext DDS 6.0.0 added support for QoS Profile composition (also referred to as QoS Profile multiple inheritance in Connext 6.0.0).

In *Connext DDS* 6.0.1, the builtin QoS Profiles have been modified to follow the QoS Profile composition pattern described in "18.3.3 QoS Profile Inheritance and Composition," in the *RTI Connext DDS Core Libraries User's Manual*. A new library, "BuiltinQosSnippetLib," has been included with a set of QoS Snippets:

- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.Common
- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.KeepAll
- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.KeepLast
- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.HighRate
- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.LowLatency
- BuiltinQosSnippetLib::Optimization.ReliabilityProtocol.LargeData
- BuiltinQosSnippetLib::Optimization.DataCache.LargeData.DynamicMemAlloc
- BuiltinQosSnippetLib::Optimization.Discovery.Common
- BuiltinQosSnippetLib::Optimization.Discovery.Participant.Compact
- BuiltinQosSnippetLib::Optimization.Discovery.Endpoint.Fast
- BuiltinQosSnippetLib::Optimization.Transport.LargeBuffers
- BuiltinQosSnippetLib::QosPolicy.Reliability.Reliable
- BuiltinQosSnippetLib::QosPolicy.Reliability.BestEffort
- BuiltinQosSnippetLib::QosPolicy.History.KeepLast_1
- BuiltinQosSnippetLib::QosPolicy.History.KeepAll
- BuiltinQosSnippetLib::QosPolicy.PublishMode.Asynchronous

- BuiltinQosSnippetLib::QosPolicy.Durability.TransientLocal
- BuiltinQosSnippetLib::QosPolicy.Durability.Transient
- BuiltinQosSnippetLib::QosPolicy.Durability.Persistent
- BuiltinQosSnippetLib::QosPolicy.Batching.Enable
- BuiltinQosSnippetLib::Feature.FlowController.838Mbps
- BuiltinQosSnippetLib::Feature.FlowController.209Mbps
- BuiltinQosSnippetLib::Feature.FlowController.52Mbps
- BuiltinQosSnippetLib::Feature.AutoTuning.Enable
- BuiltinQosSnippetLib::Feature.Monitoring.Enable
- BuiltinQosSnippetLib::Feature.Security.Enable
- BuiltinQosSnippetLib::Feature.TopicQuery.Enable
- BuiltinQosSnippetLib::Transport.TCP.LAN.Client
- BuiltinQosSnippetLib::Transport.TCP.WAN.Symmetric.Client
- BuiltinQosSnippetLib::Transport.TCP.WAN.Asymmetric.Server
- BuiltinQosSnippetLib::Transport.TCP.WAN.Asymmetric.Client
- BuiltinQosSnippetLib::Compatibility.ConnextMicro.Version243
- BuiltinQosSnippetLib::Compatibility.OtherDDSVendor.Enable
- BuiltinQosSnippetLib::Compatibility.510Transport.Enable

Furthermore, all of the existing QoS Profiles have been modified to use these QoS Snippets, achieving the same functionality they already have. Additionally, the builtin QoS Profiles from the experimental library "BuiltinQosLibExp" have been moved to the non-experimental library "BuiltinQosLib." The experimental library "BuiltinQosLibExp" still keeps these QoS Profiles for backward compatibility.

A documentation file of these builtin QoS Profiles can be found in **<NDDSHOME>/resource/xml/BuiltinProfiles.documentationONLY.xml**.

2.2 Ability to configure asynchronous publisher thread destruction timeout

In previous releases, the Asynchronous Publisher thread had a destruction timeout of 10 seconds. This timeout was exercised in two scenarios: as a result of a publisher destruction, or as a result of destroying a *DomainParticipant*. As such, this timeout had a direct impact on the potential maximum time it could take to destroy a *DomainParticipant*.

This release adds the ability to set this timeout, giving you more control over the maximum time it can take to destroy a *DomainParticipant*. The timeout can be set through the following new Property Qos policy for the *DomainParticipant*:

dds.domain_participant.asynchronous_publisher_thread_destruction_timeout: Maximum time the *DomainParticipant* will wait for the destruction of an asynchronous publisher thread. If this timeout expires before the asynchronous publisher thread is destroyed, the *DomainParticipant* cannot safely release the thread's resources, and it will skip their release. Default: 10 (seconds). Valid values: 1-60 (seconds).

2.3 Validate property names

Previously when you specified an incorrect property name, *Connext DDS* ignored it. You might not have known why the property's configuration wasn't being applied. Now, when you specify an incorrect property name, *Connext DDS* logs a warning similar to the following:

```
DDS_PropertyQosPolicy_validatePropertyNames:Unexpected property: dds.type_consistnecy.ignore_
sequence_bounds. Closest valid property: dds.type_consistency.ignore_sequence_bounds
```

The message contains the invalid property and the closest property name.

This validation can be configured by setting the property **dds.participant.property_validation_action**:

- 0: validate properties. Upon failure, log exceptions and fail.
- 1: skip validation.
- -1: validate properties. Upon failure, log warnings and do not fail.

The default validation behavior is -1.

2.4 Set thread names for Connext DDS threads on Linux, macOS, iOS, QNX, INTEGRITY, Windows, and VxWorks Platforms

Connext DDS now sets thread names at the operating system level for the following platforms: Linux, macOS, iOS, QNX, INTEGRITY, Windows, and VxWorks. (In the previous release, thread names were only set for VxWorks and INTEGRITY platforms.)

For information on how to see the thread names, see the new section, "24.5 Identifying Threads Used by *Connext DDS*," in the *RTI Connext DDS Core Libraries User's Manual*.

2.5 Logging a backtrace for failures

In some scenarios, it might be desirable to log the backtrace from the code. A backtrace is a list of the function calls that are currently active in a thread. You can usually inspect a backtrace by using debugging utilities like gdb, but sometimes these are not available.

Now, *Connext DDS* logs the backtrace when a precondition fails in debug mode and when a segmentation fault occurs, for Darwin, Windows, and Linux. The backtrace feature is automatically enabled upon creation of the first *DomainParticipant*. (That is, you will not see the backtrace log in a failure until the first *DomainParticipant* is created.)

- Normally when a precondition fails, the execution continues and there is no information about the problem. But now *Connext DDS* provides a backtrace with context about where the issue was.
- When a segmentation fault occurs, the processor or operating system does not always provide a core dump. Now *Connext DDS* provides a backtrace with context about where the issue was.

For Linux, the output of the backtrace will look like this:

```
#1 RTIOsapiProcessTester_testPrintBacktrace
/connextdds/osapi.1.0/srcC/process/test/processTester.c:638 [0x417371]
#2 RTITestSetting_runTestsExt /connextdds/test.1.0/srcC/setting/Setting.c:719 [0x4623B8]
#3 RTITestSetting_runTests /connextdds/test.1.0/srcC/setting/Setting.c:905 [0x462B85]
#4 RTIOsapiProcessTester_run /connextdds/osapi.1.0/srcC/process/test/processTester.c:683
[0x41750C]
#5 RTITestSetting_runTestsExt /connextdds/test.1.0/srcC/setting/Setting.c:719 [0x4623B8]
#6 RTITestSetting_runTests /connextdds/test.1.0/srcC/setting/Setting.c:905 [0x462B85]
#7 RTIOsapiTester_run /connextdds/osapi.1.0/srcC/test/Tester.c:128 [0x4039CB]
#8 main /connextdds/osapi.1.0/srcC/test/Tester.c:213 [0x403A65] #9 ?? ??:0 [0xE8434830] #10 _
start ??:? [0x403759]
```

For Linux and Darwin, this feature will install a signal handler for SIGSEGV, if there is not one already installed.

See the *RTI Connext DDS Core Libraries Platform Notes* for further details on enabling this feature in Windows and Linux.

2.6 Log a message when samples are not delivered due to time synchronization issue between machines

When a *DataReader* sets DDS_DestinationOrderQosPolicyKind to DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, the *DataReader* accepts a sample only if the source timestamp is no farther in the future from the reception timestamp than the specified tolerance. Otherwise, the sample is dropped.

In previous releases, the sample was dropped silently. In this release, a warning is now logged so that you can be aware of the issue. The message is similar to the following:

```
PRESPsReaderQueue_isNewerSample:[Topic: 'Topic', Type: 'Foo'] Dropped sample from DataWriter
(0X101BA02,0XB7568EE7,0X8B424723,0X80000003). The source timestamp (2019-07-12 11:18:30.304267)
is greater than the received timestamp (2019-07-12 11:35:10.304174) by more than the source_
timestamp_tolerance. The system clocks for the DataWriter and DataReader may not be
synchronized.
```

This message will help with the detection of time synchronization issues between machines.

2.7 New log message when DataWriters send samples

In previous releases, when a *DataReader* read a sample, a STATUS_ALL message was logged; however, there wasn't any information logged when a *DataWriter* wrote a sample.

Starting in 6.0.1, a new STATUS_ALL message is now logged when a *DataWriter* writes a sample. The message logged in the write would be similar to:

```
COMMENDBeWriterService_write:[1494884710,2049588453] writer oid 0x80000407 sends DATA to reader
(0x101BA02,0xB7568EE7,0x8B424723,0x80000003), sn [(0000000000,00000001)]"
```

2.8 More readable incompatible data representation QoS warnings

Log messages for incompatible DataRepresentationQosPolicy warnings are now more readable. In the following example, the DataRepresentationIds are clearly indicated:

```
PRESPsService_isLocalWriterRemoteReaderCompatible:incompatible data representation qos: writer
XCDR2; reader empty (XCDR)
```

2.9 Incompatible QoS warnings now include the topic and type names

Log messages for incompatible QoS warnings now include the topic and type names. For example:

```
PRESPsService_isLocalWriterRemoteReaderCompatible:endpoints of topic 'LongAndDoubleTopic',
local type 'LongAndDoubleType', and remote type 'LongAndDoubleType' have incompatible QoS
```

2.10 Improved logged message when discovering a ContentFilteredTopic that exceeds configured resource limits

When a ContentFilteredTopic could not be deserialized at the discovery level because it hit the maximum length configured in the resource limits, the following messages were logged:

```
DISCBuiltin_deserializeContentFilterProperty:content filter de-serialization error

DISCBuiltinTopicSubscriptionDataPlugin_deserializeParameterValue:content filter could not be
deserialized. Discovery will proceed but writer-side filtering will be disabled
```

This release updates the logged messages to include more information. The messages logged in this situation would be similar to:

```
DISCBuiltin_deserializeContentFilterProperty: !deserialize content filter "myFilter" field.
Reached maximum length 256 for content filter. Please consider increasing contentfilter_
property_max_length parameter under participant's resource limits.

DISCBuiltin_deserializeContentFilterProperty:content filter de-serialization error

DISCBuiltinTopicSubscriptionDataPlugin_deserializeParameterValue: [Topic: myTopic, Type:
myType] Content filter could not be deserialized. Discovery will proceed, but writer-side
filtering will be disabled
```

2.11 Warning messages shown when properties cannot be stored during discovery of remote entities now print TopicName

Connext DDS reports discovery warning messages when the properties (PropertyQosPolicy) from a remote entity (*DataWriter*, *DataReader*, *DomainParticipant*) cannot be stored in the local

DomainParticipant because resource limits are hit. These messages now print the TopicName.

2.12 Enhanced logging message when there are no destinations available to send ACKs/NACKs

If a *DataReader* had no destinations available to send ACKs/NACKs, the following message was logged:

```
COMMENDFacade_sendAck:!precondition
```

This release enhances this message; now this message will look something like the following:

```
COMMENDFacade_sendAck:[Local Participant: 101f9f9 733aad3 fe2bba69 | Local Endpoint 20087]
[Remote Participant: 101f9f9 733aad3 fe2bba69 | Remote Endpoint 20082] No destinations
available to send ACKs/NACKs.
This could prevent communication with the remote endpoint.

COMMENDSrReaderService_assertRemoteWriter:!send preemptive ACK
```

2.13 New warning message logged when a TopicQuery is not dispatched due to DataWriter's durability

In order for a TopicQuery to be dispatched, the *DataWriter* must be configured with a non-volatile durability. The exception to this rule is when the *DataWriter* is used as a proxy in *RTI Routing Service*.

Previously, there was no indication that a TopicQuery was not going to be dispatched in this scenario. In order to make it clearer that a TopicQuery will not be dispatched from a volatile *DataWriter*, the following log message has been added and is visible with a verbosity level of warning or higher:

```
PRESPsWriter_beginTopicQueryPublication:[Local Participant: 101702c d9cebd4 b4b81961 | Local
Endpoint: 80000003, Topic: 'Example MyType' Type: 'MyType', Writer Name: 'MyTypeDataWriter']
TopicQuery will not be dispatched since DataWriter has volatile durability
```

2.14 Report an error when IP Mobility thread cannot be created

The creation of the IP Mobility thread reported a warning if there was an error. But an error in the functioning of *Connext DDS* should report an exception, not a warning. Now an exception is reported if the IP Mobility thread creation fails.

2.15 Connext DDS thread execution no longer blocks SIGSEGV signal

Previously, if a SIGSEGV signal was raised in a thread created by *Connext DDS*, it was blocked in POSIX-compliant operating systems.

Now, if the SIGSEGV signal is raised in a *Connext DDS* thread, it will not be blocked.

2.16 Compressed TypeObject deserialization no longer depends on DDS_DomainParticipantResourceLimitsQosPolicy's type_object_max_serialized_length

In 6.0.0, compressed TypeObject (TypeObjectLb) deserialization depended on the DDS_DomainParticipantResourceLimitsQosPolicy's **type_object_max_serialized_length** to control the maximum size of the buffer *Connext DDS* uses to uncompress the received compressed TypeObject. Starting with 6.0.1, this uncompression buffer is no longer controlled by the DDS_DomainParticipantResourceLimitsQosPolicy's **type_object_max_serialized_length** (default value: 8192 bytes), and instead is controlled by the DDS_DomainParticipantResourceLimitsQosPolicy's **type_object_max_deserialized_length** (default value: DDS_LENGTH_UNLIMITED).

3 Transports

3.1 Interface name supported on Windows platforms for allow/deny interface properties

The transport properties **allow_interfaces_list**, **deny_interfaces_list**, **allow_multicast_interfaces_list**, and **deny_multicast_interfaces_list** now support the use of the interface name (not just the interface addresses) on Windows platforms.

3.2 New transport property to detect interface changes using polling

This release adds a new transport property, **force_interface_poll_detection**, that allows you to modify the way *Connext DDS* detects changes on the network interfaces in IP mobility scenarios. This property applies to the builtin UDPv4 and UDPv6 transports, and the TCP/TLS transports.

If the operating system supports asynchronous notification of changes on the interfaces, *Connext DDS* normally relies on these notifications to update the interface information. When the **force_interface_poll_detection** property is enabled, *Connext DDS* will instead use a polling mechanism to query the interfaces periodically. The default value for this property is FALSE.

4 APIs

4.1 Optional types are now more similar to std::optional (Modern C++ API)

The types **dds::core::optional<T>** and **rti::core::optional_value<T>** have similar semantics to **std::optional<T>**, but have different implementations.

To more closely match the API of **std::optional<T>**, **dds::core::optional<T>** and **rti::core::optional_value<T>** now include the following new member functions:

- **has_value()**, previously called **is_set()**, checks if the value exists.
- **value()**, previously called **get()**, returns the value if it exists or throws an exception if it doesn't.

- operator*, previously not available, returns the value without checking if it exists. If it doesn't exist, the behavior is undefined.

is_set() and **get()** are still available but have been deprecated.

With these changes, if a future release replaces **dds::core::optional**, **rti::core::optional_value**, or both, with **std::optional**, applications should require minimal changes or no changes at all.

4.2 Improved DataReader read/take API (.NET API)

This release includes new functions to read data from a *DataReader* that are easier to use and more .NET-friendly.

Three new functions are available in **TypedDataReader<T>**, the base class of all *DataReaders*:

- take() (new no-argument overload)
- read() (new no-argument overload)
- select() (new function)

These functions return a **LoanedSamples** collection, which implements **IDisposable** (to return the loan) and IEnumerable (to integrate with .NET features such as foreach loops and Linq queries).

The function **select()** is followed by a series of setters that specify a query and a call to **take()** or **read()**.

When the reader has no data samples, these functions return an empty **LoanedSamples** sequence—they do not throw **Retcode_NoData**. This change improves performance significantly for applications that read no data frequently.

The pre-existing variants of **read*** and **take*** taking two sequences as arguments are still available. The signatures and behavior of these functions haven't changed.

The following code uses the new **select()** function:

```
using (var samples = reader.select()
        .sample_state(SampleStateKind.NOT_READ_SAMPLE_STATE)
        .instance(ref myInstance)
        .max_samples(100)
        .read())
{
    foreach (var sample in samples)
    {
        if (sample.info.valid_data)
        {
            Console.WriteLine(sample.data.myField);
        }
    }
}
```

The code above produces the same result as the following, existing code, which uses the existing function **read_instance**:

```
var sampleSeq = new FooSeq();
var infoSeq = new SampleInfoSeq();
try
{
    reader.read_instance(
            sampleSeq,
            infoSeq,
            100,
            ref myInstance,
            SampleStateKind.NOT_READ_SAMPLE_STATE,
            ViewStateKind.ANY_VIEW_STATE,
            InstanceStateKind.ANY_INSTANCE_STATE);
    for (int i = 0; i < sampleSeq.length; i++)
    {
        if (infoSeq.get_at(i).valid_data)
        {
            var data = sampleSeq.get_at(i);
            Console.WriteLine(data.myField);
        }
    }
}
catch (Retcode_NoData)
{
    // nothing to do
}
finally
{
    reader.return_loan(sampleSeq, infoSeq);
}
```

Note: the **LoanedSamples** collection has been added to a new **DDS.Ext** namespace to avoid a collision with a pre-existing **RTI.Connext.Infrastructure.LoanedSamples** type used in the Request-Reply API. In a future release, **DDS.Ext.LoanedSamples** will be moved to DDS and will replace **RTI.Connext.Infrastructure.LoanedSamples**.

4.3 Bounded_sequence now can be initialized from an initializer_list (Modern C++ API)

The type **rti::core::bounded_sequence** now provides a constructor and an assignment operator taking a **std::initializer_list**.

This new constructor allows code like the following:

```
rti::core::bounded_sequence<int, 5> sequence {1, 2, 3};
```

It also allows *rtiddsgen*-generated topic-types to use uniform-initialization syntax. For example, given the following FooContainer IDL type:

```
struct Foo {
    long x, y;
};

struct FooContainer {
    string name;
    sequence<Foo, 10> foo_seq;
};
```

An instance of the type generated with **rtiddsgen -language C++11** can be created as follows:

```
FooContainer my_sample {
    "MyFooContainer",
    { Foo {1, 1}, Foo {2, 2}, Foo {3, 3} }
};
```

4.4 New API to store IDL representation of a TypeCode into a string

Previously, the **DDS_TypeCode_print_IDL** API printed a TypeCode in IDL format to standard output. This functionality has been extended such that it is now possible to print to a string using **DDS_TypeCode_to_string**. This API has been added in C, traditional C++, modern C++, and .NET.

5 Micro Application Generator (MAG)

5.1 Support for ignore_loopback_interface when configuring UDPv4 transport

This release adds support in the Micro Application Generator (MAG) for **ignore_loopback_interface** when configuring the UDPv4 transport.

MAG will add the loopback interface to the list of transports enabled to transmit and received user data traffic based on the **ignore_loopback_interface** value (see **NDDS_Transport_UDPv4_Property_t::ignore_loopback_interface** for more information).

5.2 Ability to configure max_samples_per_remote_builtin_endpoint_writer, builtin_endpoint_reader_nack_period and rtps_reliable_reader.nack_period through XML in MAG

This version introduces support in MAG for these QoS settings in XML:

- **rtps_reliable_reader.nack_period**
- **max_samples_per_remote_builtin_endpoint_writer**.

  This is the maximum value of **max_samples_per_remote_writer** within the **participant_reader_resource_limits**, **publication_reader_resource_limits**, and **subscription_reader_resource_limits**. It maps to **max_samples_per_remote_builtin_endpoint_writer** in *Connext DDS Micro*.
- **builtin_endpoint_reader_nack_period**.

This is the **nack_period** inside a **subscription_reader**. It maps to **builtin_endpoint_reader_nack_period** in *Connext DDS Micro*.

5.3 Ability to configure the DDS_PresentationQos policy through XML in MAG

This release introduces support in MAG for the PRESENTATION QosPolicy in XML. In *Connext DDS Micro*, this QoS policy is used by DDS_SubscriptionBuiltinTopicData when DPSE discovery is used.

6 Performance

6.1 Performance optimization on serialization/deserialization for sequences with complex elements

This release improves the serialization/deserialization performance for sequences containing complex elements when the element type has the following properties:

- It is marked as @final
- It only contains primitive members, or complex members with only primitive members

For example:

```
@final
struct Point {
    long x;
    long y;
};

@final
struct MyType {
    sequence<Point, 1000> points;
};
```

The optimization is applied to the code generation only when the optimization level (**-optimization**) is set to 2 (default value). The optimization always applies to DynamicData.

This improvement has been made for the following languages: C, C++, C++03, and C++11.