RTI Connext DDS

Core Libraries

What's New in Version 5.3.0

Printed in U.S.A. First printing.
June 2017.

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/

Table of Contents

What's New in 5.3.0

Connext DDS 5.3.0 is a general access release. This document highlights new platforms and improvements in 5.3.0. For what's *fixed* in 5.3.0, see the RTI Connext DDS Core Libraries Release Notes.

1 Support for RTI Security Plugins

Connext DDS 5.3.0 provides a General Access Release (GAR) of RTI Security Plugins.

Security Plugins introduces a robust set of security capabilities, including authentication, encryption, access control and logging. Secure multicast support enables efficient and scalable distribution of data to many subscribers. Performance is also optimized by fine-grained control over the level of security applied to each data flow, such as whether encryption or just data integrity is required.

Security Plugins is available in a separate package from the RTI Support Portal, https://support.rti.com/. See the *RTI Security Plugins Release Notes,* as well as the *RTI Security Plugins Getting Started Guide*.

2 Support for Locator Change At Run-Time

In Connext DDS 5.2.3 and earlier, the set of IP addresses associated with an RTPS endpoint (*DataWriter* or *DataReader*) could not be changed after the RTPS endpoint was enabled. Therefore, Connext DDS was not prepared to deal with IP addresses changes at run-time, including the following use cases:

1. Starting without network connectivity and connecting to the network at run time.
2. Switching network interfaces. For example, going from Wired to WIFI.
3. Acquiring a new IP address after DHCP lease expiration.
4. Mobile devices roaming across network segments.

This release introduces support for IP-address changes at run-time for the following transports:

- UDPv4 and DTLSv4
- UDPv6
- TCPv4 and TLSv4
- LBRTPS
- ZRTPS

The functionality is enabled out-of-the-box, except for the case where systems can be started without enabled network interfaces. For this case, you must set **participant_qos.wire_protocol.rtps_auto_id_kind** to DDS_RTPS_AUTO_ID_FROM_UUID.

When possible, the detection of IP address changes is done asynchronously using the APIs offered by the underlying OS. If there is no mechanism to do that, the detection will use a polling strategy where the polling period is 500 millisec by default.

The polling period can be configured using the following transport properties in the *DomainParticipant's* PropertyQosPolicy: **<<transport prefix>>.interface_poll_period**.

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

Starting without Enabled Network Interfaces

This new feature supports the use case where an application that uses DDS is started with no network interfaces enabled or connected and subsequently connects to a network while the application is running.

If this is your case, you must change the GUID prefix generation algorithm to not be based on the IPv4 address of the first enabled interface, and use a UUID algorithm instead This is important to avoid collisions on the GUID, which needs to be unique on the network. To enable the use of a UUID algorithm to generate the GUID, you need to modify the *DomainParticipant's* WireProtocol QoS policy. Specifically, set **participant_qos.wire_protocol.rtps_auto_id_kind** to DDS_RTPS_AUTO_ID_FROM_UUID.

Disabling IP Mobility Support

Notice that Connext DDS 5.2.3 and earlier versions of the product will report errors if they detect locator changes in an RTPS endpoint. You can disable the notification and propagation of these changes. This way, an interface change in a 5.3.0 application will not trigger errors in an application running 5.2. Of course, this will prevent the 5.3.0 application from being able to detect network interface changes.

To disable IP Mobility, you can set the following transport property in the *DomainParticipant's* PropertyQosPolicy: **<<transport prefix>>.disable_interface_tracking**.

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

3 Ability to Detect Unreachable Locators

It is possible that a destination RTPS Endpoint (*DataWriter* or *DataReader*) announces locators that are temporarily or permanently unreachable.

For example, a destination RTPS endpoint may announce an IP address that is only valid within the LAN where the endpoint is running. In such case, RTPS endpoints running outside the LAN should not use that address to send information to the destination endpoint.

In previous releases, the middleware did not have the ability to detect unreachable locators. This had two main consequences:

1. The middleware could waste CPU cycles and bandwidth sending RTPS messages to unreachable locators.
2. If the unreachable locator was a multicast locator, the destination endpoint would never receive live samples from the sender's endpoints. For best-effort communication, this would have resulted in never receiving samples. For reliable communication, this would have resulted in sending samples as repair traffic.

This release introduces a new locator REACHABILITY PING mechanism, which the middleware can use to detect when an RTPS endpoint is not reachable at a locator; then it can stop using the locator to send data to the endpoint. For temporary disconnections, the middleware will be able to detect and use an RTPS endpoint’s locator that becomes reachable again. While data is not sent to an unreachable locator, the middleware still sends periodic REACHABILITY PING messages to check if it is still unreachable.

The configuration of the REACHABILITY mechanism is done using the following *DomainParticipant's* QosPolicy values:

- **participant_qos.discovery_config.locator_reachability_assert_period**

  This value configures the period at which a *DomainParticipant* will ping all the locators that it has discovered. This period should be strictly less than **locator_reachability_lease_duration**. Default: 20 seconds.

- **participant_qos.discovery_config.reachability_lease_duration**

  This value configures a timeout announced to remote *DomainParticipants*. This timeout is used by remote *DomainParticipants* as the maximum period by which a remote *DomainParticipant* locator must be asserted (through a REACHABILITY PING message sent from the local *DomainParticipant* to that locator), or that locator will be considered "unreachable" from the local *DomainParticipant*. If the value is set to infinite, the feature is disabled. Default: infinite (disabled).

- **participant_qos.discovery_config.locator_reachability_change_detection_period**

This value determines the maximum period at which Connext DDS will check to see if remote *DomainParticipant* locators are still alive. Default: 60 seconds.

4 Automatic Way to Configure Host ID of SHMEM Transport Instance Friendly to IP-Mobility Scenarios

In Connext DDS, all the *DomainParticipants* within the same host must share a common shared-memory (SHMEM) transport address. In previous releases, this address was automatically generated using the host ID that was the IP address of the first available network interface.

This way of generating the transport address was not friendly to IP-mobility scenarios in which:

- A host may start without any network interface available.
- A host may change the IP address of its first enabled interface at run time.

This release introduces a better way to automatically generate the host ID used to build SHMEM addresses. This new way does not depend on the IP address of an interface. This mechanism is based on the MAC address of the first available interface.

This new host ID generation is only activated when you set **participant_qos.wire_protocol.rtps_auto_id_kind** to DDS_RTPS_AUTO_ID_FROM_UUID.

Compatibility Note: Because the host ID generation mode for SHMEM changes depending on the value of **participant_qos.wire_protocol.rtps_auto_id_kind**, to communicate over shared memory within a node, all the *DomainParticipants* running within that node must use a consistent value for **participant_qos.wire_protocol.rtps_auto_id_kind**.

5 New GUID Prefix Generation Mode Based on UUID

This release introduces a new GUID prefix generation mode for a *DomainParticipant* based on UUID. This mode does not require having a network interface and is friendly to IP mobility scenarios in which a Connext DDS application may start on a node that does not have a physical network interface enabled.

In addition, this new GUID generation makes the first two bytes equals to the RTPS VendorId to be compliant with the latest RTPS specification.

To configure this generation mode set **participant_qos.wire_protocol.rtps_auto_id_kind** to DDS_RTPS_AUTO_ID_FROM_UUID.

6 Increased Maximum Locators a Participant can Announce

Starting with this release, the maximum number of locators a participant can handle and announce has been increased from four to 16.

For more information about this change, please see the Discovery Wire Compatibility section in the *RTI Connext DDS Core Libraries Release Notes* and the sections on these QoS policies in the *RTI Connext DDS Core Libraries User's Manual*: MULTI_CHANNEL, TRANSPORT_UNICAST, and TRANSPORT_MULTICAST.

7 Support for Querying DataWriter Sample Cache Based on Selection Criteria

This release provides a mechanism called TopicQuery, which allows a *DataReader* to query the sample cache of its matching *DataWriters*.

To create a TopicQuery, use the following *DataReader* operation:

```
struct TopicQuerySelection {
....char * filter_class_name; /* SQL if NULL */
....char * filter_expression;
....StringSeq filter_parameters;
};
TopicQuery * DataReader::create_topic_query(const TopicQuerySelection &
selection);
```

Each TopicQuery is identified by a GUID, which can be accessed using the method:

```
void TopicQuery::get_guid(GUID_t & query_guid);
```

Connext DDS will propagate TopicQueries to other *DomainParticipants* and their *DataWriters*. When a *DataWriter* receives the TopicQuery that was created by a matching *DataReader*, the *DataWriter* will send the cached samples that pass the filter.

To configure how to dispatch a TopicQuery, there is a new QoS policy for *DataWriters*: TopicQueryDispatchQosPolicy. By default, a *DataWriter* ignores TopicQueries unless they are explicitly enabled using this policy.

TopicQuery samples are delivered in a separate RTPS channel. This allows *DataReaders* to receive TopicQuery samples and live samples in parallel. This is a key difference with respect to the Durability QoS policy.

Late-joining *DataWriters* will also discover existing TopicQueries. To delete a TopicQuery, use the following *DataReader* operation:

```
void DataReader::delete_topic_query(TopicQuery * query);
```

After deleting a TopicQuery, new *DataWriters* won't discover it and existing *DataWriters* currently publishing cached samples may stop before delivering all of them.

The samples received in response to a TopicQuery are stored in the associated *DataReader's* cache. The existing read/take operations can retrieve TopicQuery samples. The field **SampleInfo::topic_query_guid** associates each sample to its TopicQuery.

To read or take only TopicQuery samples, ReadConditions and QueryConditions provide two new operations:

```
typedef enum {
    LIVE_STREAM,
    TOPIC_QUERY_STREAM
} StreamKind;
struct ReadConditionParams {
    SampleStateMask sample_states;
    ViewStateMask view_states;
    InstanceStateMask instance_states;
    StreamKindMask stream_kinds;
};

struct QueryConditionParams : ReadConditionParams {
    char *query_expression;
DDS_StringSeq query_parameters
};

ReadCondition * DataReader::create_readcondition_w_params(const ReadConditionParams &
params);
QueryCondition* DataReader::create_querycondition_w_params(const QueryConditionParams &
params);
```

For more information on TopicQueries, see the API Reference HTML documentation and the TopicQuery examples (distributed separately).

8 Ability to Configure Wire IDL String Encoding and Support for UTF-8 Encoding

In previous releases, there was no a way to configure the wire encoding for IDL strings, string sequences, and string arrays.

- In C, C++, and modern C++, the serialization routines put the bytes of the string on the wire with no transformation.
- In Java, the serialization routines converted Java Strings characters to ISO-8859-1 encoding for code generated TypePlugins and UTF-8 for DynamicData. There was no way to change this behavior.
- In .NET, the serialization routines converted .NET Strings characters to ISO-8859-1 encoding for code generated TypePlugins and ANSI using the local code page for DynamicData.

This release allows you to configure the IDL wire string encoding and adds official support for UTF-8 encoding.

For generated code TypePlugins and builtin types, the wire encoding can be set per endpoint by setting the following properties:

- **dds.data_reader.type_support.cdr_string_encoding_kind**
- **dds.data_writer.type_support.cdr_string_encoding_kind**

These properties can be set at the endpoint level or the participant level. The only values currently supported are UTF-8 and ISO-8859-1. By default, the wire character encoding is assumed to be UTF-8.

Notice that in in C, C++, and modern C++, it is the user's responsibility to use the right character encoding when populating the string values independently of the value of the properties. There is no automatic transformation performed by the middleware.

In Java and .NET, the serialization routines will automatically convert between wire encoding and the VM default character encoding based on the value of the properties.

For DynamicData TypePlugins, this release adds a new property, **string_character_encoding**, to DynamicDataProperty_t for Java and .NET. This new property allows you to select the wire encoding for IDL strings in a DynamicData object. The following values are supported:

- For Java: StandardCharsets.ISO_8859_1 and StandardCharsets.UTF_8
- For .NET: StringEncodingKind::ISO_8859_1 and StringEncodingKind::UTF_8

In C, C++, and modern C++, it is the user's responsibility to use the right character encoding when populating the string values of the DynamicData object. There is no automatic conversion done by the middleware.

In Java and .NET, the set routines will automatically convert between wire encoding and the VM default character encoding based on the value of **string_character_encoding**.

9 Identification of Corrupted RTPS Messages by Using CRC

Connext DDS 5.3.0 supports a new DomainParticipantProtocolStatus to identify and collect statistics about corrupted RTPS messages. A four-byte CRC is computed over the DDS RTPS message, including the RTPS Header. This CRC is sent as a new RTPS CRC32 submessage. The subscribing application detects this new submessage and validates the contained CRC. When a corrupted message is detected, the protocol status is updated and the message is dropped.

To enable the use of CRC in a *DomainParticipant*, there are two new fields in the WireProtocolQosPolicy: **compute_crc** and **check_crc**. To send the CRC, enable **compute_crc** at the sending application. To get the DomainParticipantProtocolStatus, enable **check_crc** at the receiving application and use the new API **DomainParticipant::get_participant_protocol_status()**.

10 Identification of Corrupted RTI TCP Messages by Using CRC

This release introduces a new CRC validation mechanism for TCP Transport messages. When this feature is enabled, a four-byte CRC is computed over the RTI TCP control and data messages, excluding the RTI TCP header. This CRC is sent as part of an extended RTI TCP header. The receiving TCP Transport uses this CRC to validate the received messages. If a corrupted message is detected, the message is dropped.

To enable this feature, two new TCP Transport properties have been added:

- **send_crc**: When set to 1, enables the computation of the CRC for sent RTI TCP messages.

  Default: 0

- **force_crc_check**: When set to 1, forces the checking of the CRC for received RTI TCP messages. By default, the TCP Transport plugin will only validate the CRC if the CRC is present in the received message. If this property is set to 1, TCP Transport will drop messages not including the CRC.

  Default: 0

Note: Enabling **send_crc** or f**orce_crc_check** breaks backward-compatibility with previous versions of TCP Transport Plugin.

11 AsyncWaitSet: Specialized WaitSet for Multi-Threaded Processing

An *AsyncWaitSet* is a new specialization of *WaitSet* that provides a mechanism to perform the wait asynchronously and dispatch the attached active *Conditions* using a thread pool.

The new component is released as an experimental feature and available for the C and C++ (traditional and modern) APIs. For details, see the API Reference HTML documentation (under the Infrastructure module, select Conditions and WaitSets, AsyncWaitSet).

12 Ability to Resize Sequences in C/C++ Beyond Defined Maximum

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

To provide a behavior that is more aligned with the IDL-to-C++ mapping, the **length()** method now allows the sequence to be resized and grow beyond the current maximum if necessary. For unbounded sequences, the semantic is equivalent to the one provided by the **ensure_length()** operation.

For bounded sequences, setting the length of the sequence to a value larger than the bound specified in IDL will still return an error (false).

13 New QoS Policy to Identify DomainParticipants of RTI Infrastructure Services

This release adds a new QoS policy named ServiceQosPolicy to *DomainParticipants*. Also available for *DataWriters* and *DataReaders*, the ServiceQosPolicy is used to mark an entity as part of an infrastructure service.

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
- DDS_WEB_INTEGRATION_SERVICE_QOS

User applications should not modify this policy's value.

An application can determine the kind of service associated with a discovered *DomainParticipant*, *DataWriter*, and *DataReader* by looking at a new field called **service** in the ParticipantBuiltinTopicData, PublicationBuiltinTopicData, and SubscriptionBuiltinTopicData, respectively.

14 Higher Precision in Print Function for Sample Members of type float and double in C/C++ Generated Examples

In earlier releases, the print function used in generated C and C++ examples printed sample members of type float and double with 6 decimal places of precision. This release prints float members up to 9 decimal places, and double members up to 17 decimal places, on all platforms except QNX, INTEGRITY and VxWorks. On these excluded platforms, the double members are printed up to 15 decimal places.

15 Increased Character Limit for Property Values in XML Configuration

In previous releases, property values in XML configuration files could not be longer than 2,047 characters. This release increases the limit to 32,767 characters. For example:

```
<property>
    <element>
        <name>MyPropertyName</name>
        <value>xxxxxxxxxx ...</value> <!-- New limit is 32,767 characters -->
```

```
    </element>
</property>
```

16 Builtin Profile for Minimum Memory Footprint (Generic.MinimalMemoryFootprint)

The new Minimum Memory Footprint profile establishes the QoS needed to reduce the memory footprint of a Connext DDS application to its minimum. Since this profile drastically reduces several resource limits, it is expected that some changes may need to be made when using this profile in a real environment to account for system and application requirements.

To make your QoS profile extend this builtin profile, you just need to use the following line:

```
<qos_profile name="My_Profile" base_name="BuiltinQosLibExp::Generic.MinimalMemoryFootprint">
```

17 Support for Deserializing Union with Unknown Discriminator Value

By default, a *DataReader* cannot receive an unknown union discriminator value from a *DataWriter*. If this situation does occur, the sample containing the unknown discriminator fails to be deserialized. For example, the following two types are assignable:

Publisher Type:

```
union MyUnion switch(long) {
    case 0:
        long m1;
    case 1:
        short m2;
    case 2:
        double m3;
}; //@Extensibility (MUTABLE_EXTENSIBILITY)
```

Subscriber Type:

```
union MyUnion switch(long) {
    case 0:
        long m1;
    case 1:
    short m2;
}; //@Extensibility (MUTABLE_EXTENSIBILITY)
```

However, if the *DataWriter* sends a union with the discriminator set to 2, the *DataReader* will not be able to deserialize the sample.

As of this release, there is a new property, **dds.sample_assignability.accept_unknown_union_discriminator**, which allows samples containing an unknown discriminator to be successfully deserialized to the default discriminator value. The default discriminator value is defined as the default element if one is

specified, otherwise the lowest value associated with any discriminator value. The member identified by the default discriminator is also initialized to its default value.

The property can be set as part of the PropertyQos in either the DomainParticipantQos or the DataReaderQos. If it is set in both the *DomainParticipant* and *DataReader*, the value in the *DataReader's* QoS will be applied.

This functionality is supported both in generated code as well as when using the DynamicData API.

18 Support for Deserializing Enum Member with Unknown Enumerator

By default, a *DataReader* cannot receive an unknown enumeration value from a *DataWriter*. If this situation does occur, the sample containing the unknown enumerator fails to be deserialized. For example, the following two types are assignable:

Publisher Type:

```
enum MyEnum {
    ONE = 1,
    TWO = 2,
    THREE = 3
};
struct MyType {
    MyEnum m1;
};
```

Subscriber Type:

```
enum MyEnum {
    ONE = 1,
    TWO = 2
};
struct MyType {
    MyEnum m1;
};
```

However, if the *DataWriter* sends m1 = THREE, the *DataReader* cannot deserialize the sample.

This release adds a new property, **dds.sample_assignability.accept_unknown_enum_value**, which allows samples containing an unknown enumerator to be successfully deserialized to the default enumeration value. The default enumeration value is defined as the first declared member of the enumeration.

You can set **dds.sample_assignability.accept_unknown_enum_value** as part of the Property QoS for either the *DomainParticipant* or the *DataReader*. If it is set in both the *DomainParticipant* and *DataReader*, the value in the *DataReader's* QoS will be applied.

This functionality is supported both in generated code as well as when using the DynamicData API.

19 Higher Precision in DynamicData print Method for Sample Members of Type float and double

In earlier releases, the DynamicData print method printed sample members of type float and double with 6 decimal places of precision. This release prints float members up to 9 decimal places; double members up to 17 decimal places on all platforms except QNX, INTEGRITY and VxWorks. On these excluded platforms, the double members are printed up to 15 decimal places.

20 Faster Creation of DomainParticipants

This release introduces improvements that make the creation of a *DomainParticipant* using the operation **DDS::DomainParticipantFactory::create_participant()** faster. Specifically, it can decrease the creation time by 250 ms.

21 Properties to Configure Verbosity of Security-Related Log Output from Secure Transports

This release provides new configuration properties that allow you to configure the verbosity of the security-related log output generated by the Secure Transports

- RTI Secure TCP Transport (nddstransporttcp)

  There is a new property called **security_logging_verbosity_bitmap**. This property is a bitmap that specifies the verbosity of security-related log messages generated by the transport.

- RTI Secure WAN Transport (nddstransportwan)

  There is a new property called **security_verbosity**. This property specifies the verbosity of security-related log messages generated by the transport.

- RTI Secure Transport (nddstransporttls)

  There is a new property called **security_verbosity**. This property specifies the verbosity of security-related log messages generated by the transport.

For more information, see *Connext DDS Core Libraries User's Manual.*

22 Support for Configuring Distributed Logger Option echo_to_stdout via XML

RTI services that use Distributed Logger can use the new option, **echo_to_stdout**, to control if log messages should be echoed to standard output. The default value of this option is TRUE.

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

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

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

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

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

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

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

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

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

25 Improved TCP Transport Logging

This release introduces improvements to the TCP Transport logging. In particular, **connect()** and **send()** related errors and relevant events now include the involved IP addresses and ports.

Example:

```
NDDS_Transport_TCPv4_Plugin_serverProcessConnect:accepted connection
from 127.0.0.1:41251
NDDS_Transport_TCPv4_Plugin_clientProcessControlConnect:connected to peer
```

```
at 127.0.0.1:36131)
NDDS_Transport_TCPv4_Plugin_clientOpenDataConnectionSR:connect to RR at
127.0.0.1:36131 (destination: [S] 127.0.0.1:36131:23660) in progress
NDDS_Transport_TCPv4_Plugin_clientProcessDataConnect:connected to RR at
127.0.0.1:36131 (destination: [S] 127.0.0.1:36131:23660)
NDDS_Transport_TCPv4_Plugin_serverProcessConnect:accepted connection from
127.0.0.1:41252
NDDS_Transport_TCPv4_Plugin_serverProcessConnectionBindRequest:bound
connection from SR at 127.0.0.1:41252 (logical port: 23660)
NDDS_Transport_TCPv4_Plugin_clientOpenDataConnectionRR:connect to SR at
127.0.0.1:36131 (logical port: 23660) in progress
NDDS_Transport_TCPv4_Plugin_clientProcessDataConnect:connected to SR at
127.0.0.1:36131 (logical port: 23660)
NDDS_Transport_TCPv4_Plugin_serverProcessConnect:accepted connection from
127.0.0.1:41253
NDDS_Transport_TCPv4_Plugin_serverProcessConnectionBindRequest:bound
connection from RR at 127.0.0.1:41253 (destination:
[C] 0000:0000:0000:0000:25BB:5EC5:B9EB:610A:23660)
```

26 Platforms

26.1 New Platforms

This release adds support for the platforms in Table 1.1 New Platforms.

Table 1.1 New Platforms

Operating System	Version
Linux ®	Red Hat® Enterprise Linux 6.8
	Ubuntu® 16.04 LTS
	NI Linux 3 (was a custom support platform, now is standard)
	Freescale™ Linux 3.8.13 on QorIQ or P4040/P4080/P4081 CPU (custom support)
	Debian™ Linux 3.12 on ARMv7a Cortex-A9 CPU (custom support)
	Wind River® Linux 7 on ARMv7 CPU (custom support)
Mac® OS X®	OS X 10.12
VxWorks®	VxWorks 7.0 on x64 CPU
Windows®	Windows Server 2016
QNX®	QNX 6.5 on PPC E500 v2 CPU (custom support)

26.2 Platforms on Legacy Operating Systems

The following legacy operating systems have reached end-of-life from their corresponding vendors. Please contact RTI support or your account manager if you require version 5.3 to run on these platforms:

- CentOS 5.x
- Red Hat Enterprise Linux 5.x
- VxWorks 6.3, 6.4, 6.6, 6.7, 6.8
- Wind River Linux 4

26.3 Removed Platforms

Platforms on the following operating systems are no longer supported:

- Freescale™ P2020RDB
- OS X 10.8
- Red Hat Enterprise Linux 4
- Red Hat Enterprise Linux 5.2 with Real-Time Extensions
- Solaris 2.9
- VxWorks 5.5, 6.5
- Wind River Linux 3
- Windows Vista, Windows XP Pro, Windows 2003
- Yellow Dog Linux 4.0

26.4 Linking with Dynamic Windows C Run-Time (CRT)

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

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

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

You may also have to ignore the static run-time libraries in your static configurations. In Visual Studio, select **Linker, Input** in the project properties and add libcmtd;libcmt to the 'Ignore Specific Default Libraries' entry. For command-line linking, add **/NODEFAULTLIB:"libcmtd" /NODEFAULTLIB:"libcmt"** to the linker options.

26.5 Warning-Free Compilation with GCC 4.1.1 and GCC 4.8.2

All warnings have been fixed when compiling both the Connext DDS libraries and the *rtiddsgen* generated code using GCC 4.1.1 or GCC 4.8.2 with the following warning flags enabled:

- -Wall
- -Wno-unknown-pragmas
- -Werror=implicit-function-declaration

This applies only to C and C++ source code.

26.6 Use of Real Time Clock instead of tickGet–VxWorks 6.3 and Higher Platforms

This release uses the Real Time Clock (instead of tickGet) to get the time from the System Clock on VxWorks 6.3 and higher platforms.

26.7 Changes to Thread-Priority Definitions for QNX and INTEGRITY Platforms

The thread-priority definitions for all QNX and INTEGRITY platforms have changed. The new definitions are:

- QNX:
  - High: 14
  - Above normal: 12
  - Normal: 10
  - Below normal: 8
  - Low: 6
- INTEGRITY
  - High: 120
  - Above normal: 100
  - Normal: 90
  - Below normal: 80
  - Low: 60

26.8 Priority Inheritance used when Creating Semaphores

Starting with this release, semaphores are created with priority inheritance on non-Linux POSIX platforms that support it (e.g., QNX and Solaris). On VxWorks platforms, semaphores are created with priority inheritance, inversion-safe and delete-safe parameters (SEM_Q_PRIORITY | SEM_INVERSION_SAFE | SEM_DELETE_SAFE).

26.9 Changes to Default Stack Size for INTEGRITY, LynxOS, and QNX Platforms

The default stack size for middleware-created threads has changed for INTEGRITY, LynxOS, and QNX platforms:

OS	Thread	New Value
INTEGRITY	Asynchronous Publisher, Asynchronous flushing thread	32*1024
	Database thread	32*1024
	Event thread	4*32*1024
	ReceiverPool threads	4*32*1024
QNX and LynxOS	Asynchronous Publisher, Asynchronous flushing thread	64*1024
	Database thread	64*1024
	Event thread	4*64*1024
	ReceiverPool threads	4*64*1024

See the *RTI Connext DDS Core Libraries Platform Notes* for more details.

27 Language Bindings and APIs

27.1 Connext DDS Java Classes now Compiled to Target Java 1.5

Connext DDS Java classes are now compiled to target java 1.5. It should therefore run on a java 1.5 or later VM.

27.2 New Operations to Convert BuiltinTopicKey to/from GUID

This release adds the following new operations to convert from BuiltinTopicKey to GUID and vice versa:

- **DDS::BuiltinTopicKey_to_guid()**
- **DDS::BuiltinTopicKey_from_guid()**

For details, see the API Reference HTML documentation.

27.3 Condition Handler now Accepts Functions that Receive Condition as Parameter–Modern C++ API Only

Before this release, the handler of a *Condition* could only receive a no-argument functor. For example:

```
condition.handler([]() { /* do something */ });
```

The handler setter function has been overloaded to also accept a functor that receives the *Condition* as a parameter. For example:

```
condition.handler([](dds::core::cond::Condition c) { /* condition == c */ });
```

27.4 TypeCode.print_complete_idl() API Added–Java API Only

A new API, **TypeCode.print_complete_idl()** has been added for the Java language. This new API will print out the IDL that represents the TypeCode. This method will take into account dependent types (sub-class, non-primitive fields) as well as include modules in the output.

27.5 New API to Compare DDS_InstanceHandle_t

A new method to compare instance handles has been added. This method will return a zero, positive, or negative number depending on the contents of the instance handles. This new API makes it easier to store instance handles in sorted lists.

Please see the API Reference HTML documentation for more details about this method.

27.6 Ability to Subtract Two Times to Calculate Duration –Modern C++ Only

This release includes a new subtraction operator that, given two Time instances, returns a Duration representing the time elapsed.

For example:

```
using namespace dds::core;
Time initial_time = my_participant->current_time();
// ...
Time final_time = my_participant->current_time();
Duration elapsed = final_time - initial_time;
```

27.7 Strong-Name Signing of Request-Reply .NET DLLs

The Request-Reply .NET DLLs, which contain "rticonnextmsgdotnet" in their names, are now signed with a strong name using RTI's private key. For more information, visit https://msdn.microsoft.com/en-us/library/wd40t7ad(v=vs.110).aspx.

28 Changes to Default QoS Values

28.1 Default Value for writer_qos.writer_resource_limits.max_remote_reader_filters Changed to DDS_LENGTH_UNLIMITED

The default value of **writer_qos.writer_resource_limits.max_remote_reader_filters** has been changed from 32 to DDS_LENGTH_UNLIMITED.

This change was done for scalability reasons to favor writer-side filtering independently of the number of matched *DataReaders* using Content Filters.

28.2 Default Value for participant_property_string_max_length Changed to 4096

The default value for the resource limit **participant_property_string_max_length** has been increased to 4096 to handle the requirements of RTI Security Plugins.

28.3 Default Value for participant_qos.resource_limits.max_gather_destinations Changed to 16

The default value for **participant_qos.resource_limits.max_gather_destinations** has been includes from 8 to 16.

The minimum value for this QoS value has been changed from 4 to 16.

29 Debuggability

29.1 Ability to Monitor Heap Memory Usage

Connext DDS includes a new feature that allows you to monitor the memory allocations done by the middleware on native heap. This feature can be used to analyze and debug unexpected memory growth.

This feature includes the following APIs (available in all languages):

- **NDDSUtility:enable_heap_monitoring()**
- **NDDSUtility::disable_heap_monitoring()**
- **NDDSUtility::pause_heap_monitoring()**
- **NDDSUtility::resume_heap_monitoring()**
- **NDDSUtility::take_heap_snapshot()**

After **NDDS_Utility_enable_heap_monitoring()** is called, you may invoke **NDDS_Utility_take_heap_snapshot()** to save the current heap memory usage to a file. By comparing two snapshots, you can tell if new memory has been allocated and, in many cases, where.

For more information, see the API Reference HTML documentation.

29.2 New Flag in LogMessage Structure Indicates Security-Related Message

This release provides a new boolean member of the LogMessage structure, **is_security_message**. This flag indicates if a log message is a transport security-related message (e.g., SSL handshake failures or certificate validation failures). This member is exposed in C, C++, Java and .NET.

C/C++:

```
struct NDDS_Config_LogMessage {
        /* ... */
       DDS_Boolean is_security_message;
}
```

Java:

```
package com.rti.ndds.config;

class LogMessage {
  // ...
  public boolean is_security_message = false;
}
```

C++/CLI:

```
#include <managed_config_dotnet.h>

namespace NDDS {
  public ref class LogMessage {
    System::Boolean is_security_message;
  }
}
```

29.3 Improved Consistency of Logging Messages

In previous releases, the format for printing the GUID of a participant was inconsistent for logging messages starting with the prefix "PRES". This release normalizes the format so all the participant GUIDs are printed using a "%x %x %x" format.

29.4 Improved Logging of Serialize and Deserialize Errors

Starting with this release, serialization/deserialization errors now include the Topic and Type name.

29.5 Additional Log Information Provided during Disruption of Reliability Protocol

To aid in debugging large systems, additional log information is now provided at the remote status verbosity level. This information is provided when there is a disruption in the reliability protocol due to a writer inactivating a reader, or when a writer times out. The information includes the involved local writer's information such as its GUID, name if available, first available sequence number, and unacknowledged sample count. Similar information about the remote reader that caused the disruption is also provided including the remote reader GUID, name if available, lowest unacknowledged sequence number, max samples, and locator information.

29.6 Support for Logging Infrastructure to Write into Multiple Files

The logging infrastructure now supports writing into multiple files via a new API, **set_output_file_set()**.

C API:

```
DDS_Boolean NDDS_Config_Logger_set_ouput_file_set(
NDDS_Config_Logger *self,
const char *file_prefix,
const char *file_suffix,
int max_capacity,
int max_files)
```

Traditional C++ API:

```
bool NDDSConfigLogger::set_ouput_file_set(
const char *file_prefix,
const char *file_suffix,
int max_capacity,
int max_files)
```

Modern C++ API:

```
void rti::config::Logger::output_file_set(
const char * file_prefix,
const char * file_suffix,
int max_bytes,
int max_files)
```

Java API:

```
void set_ouput_file_set(
String file_prefix,
String file_suffix,
int max_capacity,
int maxFiles) throws IOException
```

.NET API:

```
System::Boolean NDDS::ConfigLogger::set_ouput_file_set(
System::String ^file_prefix,
```

```
System::String ^file_suffix,
System::Int32 max_capacity,
System::Int32 max_files)
```

The logged output will be redirected to a set of files whose names are configured with a prefix and a suffix. The maximum number of bytes configures how many bytes to write into a file before opening the next file. After reaching the maximum number of files, the first one is overwritten.

For more information, see the API Reference HTML documentation.

30 XML-Based Application Creation

30.1 Support for EntityNameQosPolicy User Settings for Entities Created with XML-Based Application Creation

When using XML-Based Application Creation, now you can explicitly configure the entity name through the EntityNameQosPolicy settings. When explicitly specified, entities created through XML-Based Application Creation will get their names from those settings. For example, see the following XML-snippet:

```
<data_writer name="MyWriter" topic_ref="MyTopic">
    <datawriter_qos>
        <publication_name>
            <name>WriterNameFromQoS</name>
        </publication_name>
    </datawriter_qos>
</data_writer>
```

A *DataWriter* created from the above configuration will have an entity name of **WriterNameFromQoS**.

This new functionality changes the previous behavior in which the default name overrode the EntityNameQosPolicy settings. For more information, see the *XML-Based Application Creation Getting Started Guide*.

30.2 XML Tag <register_type> is now Optional and its Attribute 'kind' is Deprecated

The XML tag <register_type> is an optional element. Consequently, its attribute kind is no longer required and its use is deprecated. This change does not affect the way XML-Application Creation registers types.

Connext DDS may still load configurations using the attribute, **kind**, and will log a warning similar to this:

```
DDS_XMLRegisterType_initialize:XML attribute 'kind' in tag <register_type> is deprecated and
will be ignored
```

See the *XML-Application Creation Getting Started Guide* for more details about type registration.

30.3 Renamed DomainParticipant Library XML Tag to <domain_participant_library>

The XML tag for the DomainParticipant library has been renamed to **<domain_participant_library>** so it is compliant with the DDS-WEB OMG standard.

This version of Connext DDS can still load configurations containing the old value (**<participant_library>**), but its use has been deprecated and may not be supported in future versions.

30.4 Global Constant DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT Initialization did not Match Documentation

The global constant DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT was initialized with values the led to a behavior different that the one specified. Namely, the fields **participant_qos_library_name**, **participant_qos_profile_name**, **domain_entity_qos_library_name**, and **domain_entity_qos_profile_name** were set to NULL, which specified to use the default QoS profiles for the entities created from configuration. The constant initialization has changed so these fields have the value DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG, which specifies to use the QoS that the entity configuration specifies. This matches the default behavior described in the documentation.

30.5 XML-Based Application Creation Supports Different ContentFilteredTopic for DataReader Configurations with Filter and Multiplicity

When using XML-Based Application Creation, a *DataReader* configuration that indicates a multiplicity greater than one and a filter will cause the creation of each *DataReader* from an independent ContentFilteredTopic whose name is composed to be unique and contains the topic and filter names.

For instance, consider the following XML snippet:

```
<data_reader name="reader" topic_ref="topic" multiplicity="3">
    <filter name="filteredTopic2" kind="builtin.sql">
        ...
    </filter>
</data_reader>
```

This XML will cause the creation of three *DataReaders*, each from a different ContentFilteredTopic instance. This behavior provides a way to independently change the filter parameters for each individual *DataReader*.

This is an improvement over the previous behavior, in which all the *DataReaders* shared only one ContentFilteredTopic and thus filter parameter changes affected all the *DataReaders*.

31 Packaging and Installer

31.1 Optional RTI Package for OpenSSL Run-Time Libraries

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

31.2 Renamed Host-Side OpenSSL Package

The OpenSSL **.rtipkg** filename now includes both the version of OpenSSL contained within the package and the version of Connext DDS it should be installed onto.

31.3 Add-ons Such as Secure WAN Transport have New Bundles that Install for Tools

Add-on libraries, such as the Limited-Bandwidth Plugins and Secure WAN Transport, have new **.rtipkg** files that automatically install the libraries to be used by RTI tools and infrastructure services.

32 Performance and Resource Consumption Improvements

32.1 Decreased Stack Size Requirement for Creating and Enabling a Participant

With respect to Connext DDS 5.2.7, this release reduces the stack size required for creating and enabling a Participant.

32.2 Decreased Heap Consumption

With respect to 5.2.7, this release reduces the heap consumption. For a detailed memory report, please refer to https://www.rti.com/products/dds/benchmarks.

33 Experimental Features

33.1 RTI Connext DDS Professional now includes New Experimental Cloud Discovery Service

RTI Connext DDS Professional now comes with an experimental new infrastructure service called Cloud Discovery Service. Cloud Discovery Service enables discovery in cloud-based environments where multicast may not be available.