RTI Connext DDS

Core Libraries

What's New in Version 6.0.0

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/

Table of Contents

What's New in 6.0.0

Connext DDS 6.0.0 is a general access release. This document highlights new platforms and improvements in the Core Libraries for 6.0.0. For what's *fixed* in the Core Libraries for 6.0.0, see the RTI Connext DDS Core Libraries Release Notes.

For what's new and fixed in other products included in the *Connext* suite, see those products' release notes.

1 Platforms

1.1 New platforms

This release adds support for the following platforms.

Operating System		CPU	Compiler	RTI Architecture Abbreviation
INTEGRITY ®	INTEGRITY 11.4.4	x64	multi 7.1.6	pentiumInty11.pcx64
INtime® for Windows	INtime 6.3 (custom target platform)	x86[a]	Visual Studio® 2017	i86Win32INtime6.3VS2017
Linux®	Red Hat® Enterprise Linux 7.5	x86	gcc 4.8.2	i86Linux3gcc4.8.2
		x64	gcc 4.8.2	x64Linux3gcc4.8.2
	SUSE® Linux Enterprise Server 12	x64	gcc 4.3.4	x64Linux2.6gcc4.3.4
	Ubuntu® 18.04 LTS	x64	gcc 7.3.0	x64Linux4gcc7.3.0
	Wind River® Linux 8 (custom target platform)	ARM v7	gcc 5.2.0	armv7aWRLinux8gcc5.2.0
QNX®	QNX 7.0.0 (custom target platform)	ARM v7	qcc 5.2.0	armv7QNX7.0.0qcc5.2.0

[a]Tested on 64-bit Windows 10 operating system.

Operating System		CPU	Compiler	RTI Architecture Abbreviation
VxWorks®	VxWorks 6.9.4.6	PPC (e6500)	diab 5.9.1	ppce6500Vx6.9.4.6diab5.9.1
	VxWorks 653 2.5.0.2 (custom target platform)	ARMv7	gcc 4.3.3	ppce500v2Vx653-2.5gcc4.3.3
Windows®	Windows 7 Windows 10 Windows Server 2016 Windows IoT[a]	x86, x64	Visual Studio 2017	i86Win32VS2017, x64Win64VS2017

1.2 Removed platforms

This release removes support for the following platforms.

Operating System	
Android	Android 2.3-4.4
INTEGRITY	INTEGRITY 5.0.11
Linux	CentOS 5.x
	Red Hat Enterprise Linux 5.x (except 5.2 is supported with gcc 4.2.1 on x86 as a custom platform)
	Wind River Linux 4
LynxOS	LynxOS 4.0, 4,2
OS X	OS X 10.10
VxWorks	VxWorks 6.3, 6,4, 6,6, 6,7, 6.8

2 Large data streaming using RTI FlatData™ language binding and Zero Copy transfer over shared memory

To meet strict latency requirements, you can reduce the default number of copies made by the middleware when publishing and receiving large samples (on the order of MBs) by using two new features: FlatData language binding and Zero Copy transfer over shared memory.

These features can be used standalone or in combination.

Using FlatData language binding, you can reduce the number of copies from the default of four copies to two copies, for both UDP and shared memory communications. FlatData is a language binding in which

[a]Per Microsoft, this should be compatible with Windows 10 IoT Enterprise with Windows native app.

the in-memory representation of a sample matches the wire representation, reducing the cost of serialization/deserialization to zero. You can directly access the serialized data without deserializing it first. To select FlatData as the language binding of a type, annotate it with the new @languange_binding(FLAT_DATA) annotation.

Zero Copy transfer over shared memory allows reducing the number of copies to zero for communications within the same host. This feature accomplishes zero copies by using the shared memory builtin transport to send references to samples within a shared memory segment owned by the *DataWriter*, instead of using the shared memory builtin transport to send the serialized sample content by making a copy. With Zero Copy transfer over shared memory, there is no need for the *DataWriter* to serialize a sample, and there is no need for the *DataReader* to deserialize an incoming sample since the sample is accessed directly on the shared memory segment created by the *DataWriter*. The new TransferModeQosPolicy specifies the properties of a Zero Copy *DataWriter*.

For more information on setting up and using one or both of these features, see the chapter "Sending Large Data" in the *RTI Connext DDS Core Libraries User's Manual*.

3 XML multiple inheritance

Connext DDS now supports multiple inheritance for QoS Profiles.

Previously, you could inherit an individual QoS or Profile only from one other QoS or Profile (then you could inherit another profile from that profile, and so on, in a single chain of inheritance). Now, independent QoSes or Profiles can be combined with multiple inheritance by using the new <base_name> tag. In this tag, you can specify multiple profiles, which will combine to produce the new desired behavior.

The base_name attribute is still available for single inheritance. You can use both single and multiple inheritance.

For more information, see section 18.3.3 "QoS Profile Inheritance" in the *RTI Connext DDS Core Libraries User's Manual*.

4 Extensible Types

4.1 Support for XCDR encoding version 2

This release adds support for the standard XCDR encoding version 2 data representation described in the "Extensible and Dynamic Topic Types for DDS" specification. This encoding version is more efficient in terms of bandwidth than the predecessor XCDR encoding version 1 supported in previous *Connext DDS* releases (and still supported in this release).

To select between XCDR and XCDR2 data representations, you can use the DataRepresentationQosPolicy for *DataReaders* and *DataWriters*. *Connext DDS* now supports this policy. You may specify XCDR, XCDR2, or AUTO to indicate which versions of the Extended Common Data Representation (CDR) are offered and requested. AUTO is the default.

A *DataWriter* offers a single representation, which indicates the CDR version the *DataWriter* uses to serialize its data. A *DataReader* requests one or more representations, which indicate the CDR versions the *DataReader* accepts. If a *DataWriter*'s offered representation is contained within a reader's sequence of requested representations, then the offer satisfies the request, and the policies are compatible. Otherwise, they are incompatible.

In support of this feature, a new QoS, DATA_REPRESENTATION, has been added for the *DataWriter* and *DataReader*. There is also a new annotation, **@allowed_data_representation**, that allows selecting the supported data representations for a type.

For more information, see the "Extensible and Dynamic Topic Types for DDS" specification from the Object Management Group (OMG): https://www.omg.org/spec/DDS-XTypes/. Also see "DATA_REPRESENTATION QosPolicy" in the *RTI Connext DDS Core Libraries User's Manual* and the "Data Representation" chapter of the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

4.2 Type-Consistency Enforcement Enhancements

The TypeConsistencyEnforcementQosPolicy can be set on a *DataReader* to control the rules that determine whether the type used to publish a given topic is consistent with that used to subscribe to it. This QoS policy is part of the The "Extensible and Dynamic Topic Types for DDS" (XTypes) specification (https://www.omg.org/spec/DDS-XTypes/).

In previous releases, the only field present in this QoS policy was **kind**, which determines whether or not type coercion is allowed.

This release introduces five additional fields in the QoS policy that provide additional control over the type matching rules:

- **ignore_string_bounds** controls whether string bounds are taken into consideration for type assignability.
- **ignore_sequence_bounds** controls whether sequence bounds are taken into consideration for type assignability.
- **ignore_member_names** controls whether member names are taken into consideration for type assignability.
- **prevent_type_widening** controls whether type widening is allowed.
- **force_type_validation** controls whether type information must be available in order to complete matching between a *DataWriter* and this *DataReader*.
- **ignore_enum_literal_names** controls whether enumeration constant names are taken into consideration for type assignability.

It is recommended to use the fields **ignore_string_bounds** and **ignore_sequence_bounds** instead of the QoS property **dds.type_consistency.ignore_sequence_bounds** supported in previous releases.

It is recommended to use the fields **ignore_member_names** and **ignore_enum_literal_names** instead of the QoS property **dds.type_consistency.ignore_member_names** supported in previous releases.

For more information, see the "Type-Consistency Enforcement" specification from the Object Management Group (OMG): https://www.omg.org/spec/DDS-XTypes/. Also see the "Type-Consistency Enforcement" section of the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

5 DynamicData

5.1 DynamicData support for accessing members without explicit binding

It is possible to refer to a nested member in a type without first having to bind to (or loan in the Modern C++ API) the type in which the member is defined. You can do this by using a hierarchical name. A hierarchical member name is a concatenation of member names separated by the '.' character. The hierarchical name describes the complete path from a top-level type to the nested member.

For example, consider the following type:

```
struct MyNestedType {
    char theChar;
};

struct MyType {
    MyNestedType theNestedType;
};
```

In this example, any DynamicData API that receives a member name will accept "theNestedType.theChar" to refer to the char member in MyNestedType:

```
char myChar = myDynamicData.get_char("theNestedType.theChar", DDS_DYNAMIC_DATA_MEMBER_ID_
UNSPECIFIED);
```

In order to access the value of theChar without using a hierarchical name, you would have to first bind to theNestedType and then get the value:

```
myDynamicData.bind_complex_member(myBoundData, "theNestedType",
DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
DDS_Char myChar = myBoundData.get_char("theChar", DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

For more details, please refer to the "Hierarchical Member Names" section in the DynamicData API Reference HTML documentation: RTI Connext DDS API Reference > Topic Module > Dynamic Data > DDS_DynamicData. (In the Modern C++ API, refer to RTI Connext DDS API Reference > Infrastructure Module > Dynamic Data.)

5.2 New DynamicData::clear_member API

The DynamicData API has been extended to include a **DynamicData::clear_member()** API that can be used to clear the value of any member in the DynamicData object. The member will be set to its default value.

5.3 Performance improvements across the DynamicData API

There have been general performance improvements made across the entire DynamicData API. You should expect more predictable performance when using the DynamicData API, regardless of the access pattern of the members in the object. In previous releases, it was recommended to set members in the order in which they were declared in the type definition and to avoid resizing variable sized members such as strings and sequences. Following these types of recommendations is no longer required. Setting members in a specific order should no longer have a significant impact on the API's performance.

6 Discovery

6.1 Support Domain ID and Domain Tag in Simple Discovery

This release adds new mechanisms to isolate a particular group of *DomainParticipants* from the rest of the *DomainParticipants* in the network. In particular, it adds two new OMG Real-Time Publish-Subscribe (RTPS) 2.3 standard mechanisms:

- **Domain ID** propagation in Simple Participant Discovery: Domain ID propagation allows *DomainParticipants* to drop Participant Discovery messages not belonging to the same Domain ID they are using. Note that this capability was already supported in previous versions of *Connext DDS* as an RTI-specific discovery parameter (0x800F). This release now adds support to the OMG RTPS 2.3 standardized parameter (0x000F). To ensure backwards compatibility, *Connext DDS* will serialize both RTI and OMG parameters.
- **Domain Tag** propagation in Simple Participant Discovery: Domain Tag propagation is a new RTPS 2.3 concept that allows for an intuitive way of subdividing domains. It consists of a string value (with a maximum of 255 characters). It allows *DomainParticipants* to drop Participant Discovery messages not belonging to the same Domain Tag they are using.

The IDs and contents for these parameters are as follows:

Member name	Member type	Parameter ID name	Parameter ID value
domainId	DomainId_t	PID_DOMAIN_ID	0x000f
domainTag	string<256>	PID_DOMAIN_TAG	0x4014

Domain ID propagation is enabled by default. Domain Tag propagation can be enabled by using the following *DomainParticipant* PropertyQos property:

dds.domain_participant.domain_tag: A string (with a maximum of 255 characters) defining the Domain Tag the *DomainParticipant* will propagate through Participant Discovery. Participants will drop any Participant discovery message that contains a Domain Tag that does not match the local Domain Tag. This parameter is only propagated if it is set to a value different than the default. Default: "" (empty, zero-length string).

Note: While Domain ID is fully supported across the whole *Connext DDS* ecosystem, Domain Tag support is currently limited to Core Libraries and infrastructure Services (by setting the aforementioned *DomainParticipant* PropertyQos property). Domain Tags are not well supported in *Connext DDS* tools (such as *Admin Console*). *Connext DDS* tools do not provide a tool-specific mechanism to configure Domain Tags. Consequently, if you configure an application to use Domain Tags, that application will not be able to communicate with *Connext DDS* tools, unless you edit the tool's QoS configuration (if it has one—for instance, see *Admin Console's* Preferences dialog) to use Domain Tags.

For more information about the Domain ID and Domain Tag, see the following sections in the *RTI Connext DDS Core Libraries User's Manual*: Section 8.3.4 "Choosing a Domain ID and Creating Multiple DDS Domains" and Section 8.3.5 "Choosing a Domain Tag."

6.2 Decreased discovery traffic

Saving bandwidth by not propagating default values of QoS policies during discovery

During the discovery process, default QoS Policies were propagated in the endpoint discovery builtin topic data. Deserialization, however, already sets any policies that are not in the serialized data to their default value. Therefore, default QoS policy values are now no longer propagated during discovery, saving discovery bandwidth.

TypeCode not propagated by default

Endpoint Discovery traffic is now reduced by not propagating the TypeCode by default.

Increase default type_object_max_serialized_length to 8KB

Now that TypeCode is no longer propagated by default in Endpoint Discovery, the default value of **type_object_max_serialized_length** has been increased from 3KB to 8KB.

Option to reduce size required to propagate TypeObject in Simple Endpoint Discovery

This release reduces the size needed to propagate a TypeObject as part of Simple Endpoint Discovery. With this feature, a compressed version of the serialized TypeObject (TypeObjectLb) is now sent as a Simple Endpoint Discovery parameter. The ID and contents for this parameter are as follows:

Member name	Member type	Parameter ID name	Parameter ID value
type_object_lb	TypeObjectLb	PID_TYPE_OBJECT_LB	0x8021

where TypeObjectLb is represented with the following IDL:

```
enum RTIOsapiCompressionPluginClassId {
    @value(0) RTI_OSAPI_COMPRESSION_CLASS_ID_NONE,
    @value(1) RTI_OSAPI_COMPRESSION_CLASS_ID_ZLIB,
    @value(2) RTI_OSAPI_COMPRESSION_CLASS_ID_BZIP,
    @value(-1) RTI_OSAPI_COMPRESSION_CLASS_ID_AUTO,
};

struct TypeObjectLb {
    RTIOsapiCompressionPluginClassId classId;
    unsigned long uncompressedSerializedLength;
    OctetSeq compressedSerializedTypeObject;
};
```

This feature is enabled by default. It can be configured in the *DomainParticipant*'s DDS_DiscoveryConfigQosPolicy field:

- **endpoint_type_object_lb_serialization_threshold:** Minimum size (in bytes) of the serialized TypeObject that will trigger the serialization of a TypeObjectLb instead of the regular TypeObject. For example, setting this policy to 1000 will trigger the serialization of the TypeObjectLb for TypeObjects whose serialized size is greater than 1000 Bytes. Range: [-1, 2147483647]. The sentinel value -1 disables TypeObject compression (by never sending TypeObjectLb). Any non-valid values will behave as 0. Default: 0

Note: In 5.3.0.17, this feature was provided but not enabled by default.

Because this feature is enabled by default, previous versions of *Connext DDS* will not be able to receive TypeObject out-of-the-box. This may affect backward compatibility with *Connext DDS* applications from previous releases. For details about how to interoperate *Connext DDS* 6.0.0 with older versions, please see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

7 Transports

7.1 Added shared memory transport UDP debugging configuration properties

When applications communicate through the shared memory transport, it is not possible to capture the traffic they exchange. There is a new feature that allows you to specify that all shared memory traffic should be published to a configurable IP address and port. This way, the traffic can be captured from a network interface.

You can enable this feature (it is disabled by default) in XML as follows:

```
<transport_builtin>
    <shmem>
        <enable_udp_debugging>true</enable_udp_debugging>
        <!-- Set the following tag to the IP address to which the traffic will be published -->
        <udp_debugging_address>127.0.0.1</udp_debugging_address>
        <!-- Set the following tag to port to which the traffic will be sent -->
        <udp_debugging_port>7900</udp_debugging_port>
    </shmem>
```

```
</transport_builtin>
```

Alternatively, you can set the following participant properties programmatically or in XML: **dds.transport.shmem.builtin.enable_udp_debugging**, **dds.transport.shmem.builtin.udp_debugging_address**, and **dds.transport.shmem.builtin.udp_debugging_port**.

This feature is only meant for debugging purposes; it is not recommended for use in production.

7.2 Improved shared memory transport compatibility detection

This release changes the way *Connext DDS* detects if two different *DomainParticipants* can communicate over shared memory. In previous releases, the compatibility detection was based on both *DomainParticipants* having the same shared memory locator. In this release, the compatibility detection is based on checking if it is possible to attach to a shared memory segment compatible with *Connext DDS*.

This method allows *Connext DDS* to establish a communication over shared memory between two *DomainParticipants* that do not generate the same shared memory locator but are running on the same host. This method produces more robust behavior when *Connext DDS* is used in hosts where network interfaces change while *DomainParticipants* are being created, or when *Connext DDS* is used in combination with other technologies like Docker®. This improvement also allows communication between *Connext DDS Professional* and *Connext DDS Micro* DomainParticipants over shared memory.

This new method of detecting shared memory transport compatibility can generate shared memory locators that are compatible with previous versions of *Connext DDS*, based on the Wire Protocol QoS policy. Due to changes in the default values of some of the fields of this QoS policy, however, a *DomainParticipant* in this release will not communicate out-of-the-box with a *DomainParticipant* from an older *Connext DDS* version. For details about how to interoperate *Connext DDS* 6.0.0 with older versions, please see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

7.3 Reintroduced support for ignore_nonrunning_interfaces for TCP and UDPv4/v6 transports

In 5.3.0, the property **ignore_nonrunning_interfaces** for the TCP and UDPv4/v6 transports was marked as deprecated and not supported. As a result, it was no longer possible to enable a TCPv4_LAN/TLSv4_LAN-only or UDPv4/v6-only *DomainParticipant* in a machine with interfaces that were up but not running, without setting up the DDS_WireProtocolQosPolicy's **participant_id**.

This release reintroduces the **ignore_nonrunning_interfaces** property, with a default of 1. When set to 0, non-running interfaces will not be ignored by the local *DomainParticipant*; therefore, automatic participant ID assignment will still work even if the system has up, but not running, interfaces.

7.4 Support for strongly-typed XML elements to configure builtin transports

You can configure builtin transports from an XML QoS Profile, using the new <transport_builtin> tag. For example:

```
<participant_qos>
    <transport_builtin>
        <udpv4>
            <message_size_max>1024</message_size_max>
        </udpv4>
    </transport_builtin>
</participant_qos>
```

The <transport_builtin> tags are an easy alternative to the regular XML tags. Because they are part of the XSD, the tag names will be auto-completed when you create the file. The XSD ensures you are not specifying invalid values. See section 18.4.6 "Transport Properties" in the *RTI Connext DDS Core Libraries User's Manual* for more information.

7.5 New values accepted by the builtin transport properties that represent boolean values

Previously, the builtin transport properties that represent boolean values only accepted "0" or "1" as valid values. This behavior has been improved, and the properties now also accept the following values: "true", "false", "yes" and "no".

8 Topic Queries

8.1 Ability to select only alive instances with TopicQuery

A new TopicQuery filter expression provides a way to restrict the selection to samples belonging to alive instances. The new special sub-expression is "@instance_state = ALIVE" followed by the rest of a normal content-filter expression. For example:

```
"@instance_state = ALIVE AND (id = 3 OR id = 10)"
```

Notes:

- "@instance_state = ALIVE" may be used as the only condition in the filter, in which case it selects all samples for all alive instances.
- Otherwise, this condition must appear at the beginning of the expression, followed by "AND." For example, the following are invalid topic query filters: "@instance_state OR id = 3" (invalid), "id = 1 AND instance_state = ALIVE" (invalid).

8.2 Support for continuous TopicQuery

In general, a TopicQuery selects a subset of the *DataWriter* history, up to the moment the *DataWriter* discovers it. After those samples are published, each *DataWriter* will forget that TopicQuery.

This release adds a new way to create a TopicQuery in which it may select samples indefinitely, even those written after its discovery. To enable this mode, a new field **kind** has been added to **DDS::TopicQuerySelection**, which can be set to DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT (default behavior) or DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS.

A continuous TopicQuery has to be explicitly deleted in the *DataReader* to stop it from selecting new samples.

See the API Reference HTML documentation for a specific programming language for more information.

8.3 Improved CPU usage in publishing applications using TopicQueries

A *DataWriter* using TopicQueries may have been subject to increasing CPU usage. This occurred when the publication rate was high and there was a high number of TopicQueries (historical and continuous) being served. This problem has been resolved.

8.4 Reduced logging verbosity when a TopicQuery is received

The verbosity of log messages similar to the following have been reduced from local to debug:

```
"PRESPsService_dispatchReceivedTopicQuerySample:participant received a Topic Query with filter
expression x=3"
```

This change was made to allow you to more easily prevent potentially sensitive information contained in the TopicQuery filter expression from being printed in log messages.

9 Language Bindings and APIs

9.1 New APIs to get version number

New APIs are provided to get the library version number of the following components:

- Connext Messaging
- Monitoring Library
- Distributed Logger
- RTI TCP Transport

The new APIs are as follows:

Connext Messaging (C)

- RTI_Connext_Messaging_get_api_version()
- RTI_Connext_Messaging_Library_get_api_version_string()
- RTI_Connext_Messaging_get_api_build_number_string()

Connext Messaging (C++)

- MessagingVersion::get_api_version()
- MessagingVersion::get_api_version_string()
- MessagingVersion::get_api_build_version()

Connext Messaging (Modern C++)

- rti::config::request_reply_api_version()
- rti::config::request_reply_build_number()

Connext Messaging (Java)

- com.rti.connext.infrastructure.Version.request_reply_api_version()

Connext Messaging (C#)

- RTI.Connext.RequestReply.ProductVersion
- RTI.Connext.RequestReply.BuildNumber

Monitoring Library

- RTIMonitor_get_version()
- RTIMonitor_get_api_build_version_string()

Distributed Logger

- RTI_DL_DistLogger_get_version()
- RTI_DL_DistLogger_get_api_build_version()
- RTI_DL_DistLogger_get_api_version_string()

RTI TCP Transport

- NDDS_Transport_TCPv4_get_library_version()
- NDDS_Transport_TCPv4_get_build_version_string()

9.2 C API: New typedef of DDS_PropertyQosPolicy

In the C API, there is a new type definition for DDS_PropertyQosPolicy, which allows you to omit "struct" when you write "DDS_PropertyQosPolicy":

```
typedef struct DDS_PropertyQosPolicy {
    /*e \dref_PropertyQosPolicy_value
    */
    struct DDS_PropertySeq value;

    DDSCPP_VARIABLE_LENGTH_VALUE_TYPE_SUPPORT(DDS_PropertyQosPolicy)
} DDS_PropertyQosPolicy;
```

9.3 Modern C++ API: new IDE-friendly way to access extension functions

The Modern C++ API separates standard and extension functions in different headers, and the way to invoke them is different, as explained in the API Reference HTML documentation under Modules > Conventions.

Until this release, invoking an extension function in a standard type required using an overloaded arrow operator. For example:

```
// Standard class (in dds namespace)
dds::domain::DomainParticipant participant(MY_DOMAIN_ID);
// Call a standard method
participant.assert_liveliness();
// Call an extension method:
participant->register_durable_subscription(...);
```

The use of the arrow operator may have been counterintuitive (the variable is not a pointer) and didn't allow an IDE to show the available extensions if the regular dot operator was used.

This release adds a new **extensions()** function to all standard types that can be used as follows:

```
// Call an extension method:
participant.extensions().register_durable_subscription(...);
```

9.4 Modern C++ API: new function allows accessing only valid-data samples

A new standalone function, **rti::sub::valid_data()**, allows accessing only valid-data samples. The function simplifies code like the following.

Before:

```
auto samples = reader.take();
for (const auto& sample : samples) {
    if (sample.info().valid()) {
        std::cout << sample.data() << std::endl;
    }
}
```

Now:

```
auto samples = rti::sub::valid_data(reader.take());
for (const auto& sample : samples) {
    std::cout << sample.data() << std::endl;
}
```

9.5 Modern C++ API: new data-access pattern returns samples in vector of shared pointers

This release adds a new function (with several overloads) that allows unpacking a collection of data samples (LoanedSamples or SharedSamples) into a vector with individual shared pointers to each sample.

LoanedSamples returns the loan at once, when the collection goes out of scope. By unpacking into individual shared_ptrs, the loan is returned when all these shared_ptrs are destroyed. This allows sharing the samples with more flexibility. For example, it is possible to combine the results of multiple calls to **read ()/take()** into a single vector.

The following code shows how **rti::sub::unpack()** can be called:

```
// Overload 1: directly create a vector from the result of any variant of read() or take()
std::vector<std::shared_ptr<const Foo>> sample_ptrs = rti::sub::unpack(reader.take());

// Overload 2: add to the vector:
dds::sub::SharedSamples<Foo> shared_samples = reader.take();
rti::sub::unpack(shared_samples, sample_ptrs);

// Freely share the samples
std::shared_ptr<const Foo> sample = sample_ptrs[0];

// All loans are returned automatically when all shared_ptrs lose all references
```

9.6 Modern C++ API: ListenerBinder now allows retrieving the entity in addition to the listener

ListenerBinder now adds a function to retrieve the associated entity (for example, a *DataReader*). Retrieving the listener was already possible.

Because of this improvement, it is now possible to keep or return a single object: the ListenerBinder. Before, applications were forced to keep both the ListenerBinder and the entity.

For an example, see the Modern C++ API reference, under Programming How-To's > DataWriter Use Cases > DataWriter Listeners.

9.7 Modern C++ API: Reference types now provide comparison operators and can be key of a map

All reference types (such as Condition or DomainParticipant) now support the <, <=, >, >= operators (== and != were already available). They compare the underlying shared_ptr.

The addition of operator < is especially useful because it allows making any reference type the key of a **std::map** with no extra code.

[RTI Issue ID CORE-8345]

9.8 New API to convert Cookie value into a pointer

This release provides a new API to convert a Cookie value into a pointer. This API is only supported in the following languages:

- **C:** DDS_Cookie_to_pointer
- **Traditional C++:** DDS::Cookie::to_pointer
- **Modern C++:** rti::core::Cookie::to_pointer

9.9 New API to get the maximum serialized size of a type sample

This releases adds a new API to get the maximum serialized size of a type sample. This API is only supported in the following languages:

- **C:** DDS_TypeCode_get_cdr_serialized_sample_max_size
- **Traditional C++:** DDS::TypeCode::get_cdr_serialized_sample_max_size
- **Modern C++:** dds::core::xtypes::cdr_serialized_sample_max_size

9.10 Added DDS_RTPS_GUID_t to C API

This release adds the DDS_RTPS_GUID_t type to the C API. The definition of this type matches the one from the RTPS specification:

```
#define DDS_RTPS_GUID_PREFIX_LENGTH 12
#define DDS_RTPS_ENTITY_KEY_LENGTH 3

/*
 * From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.
 */
typedef DDS_Octet DDS_RTPS_GuidPrefix_t[DDS_RTPS_GUID_PREFIX_LENGTH];

/*
 * From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.
 */
typedef struct DDS_RTPS_EntityId_t {
    DDS_Octet entityKey[DDS_RTPS_ENTITY_KEY_LENGTH];
    DDS_Octet entityKind;
} DDS_RTPS_EntityId_t;

/*
 * From DDS-RTPS Specification, clauses 8.4.2.1 and 9.3.1.
 */
typedef struct DDS_RTPS_GUID_t {
    DDS_RTPS_GuidPrefix_t prefix;
    DDS_RTPS_EntityId_t   entityId;
} DDS_RTPS_GUID_t;
```

Note that *RTI Connext DDS* APIs still use the DDS_GUID_t type defined as:

```
#define DDS_GUID_LENGTH 16
/*
 * Alternative representation of DDS_RTPS_GUID_t. Memory and wire representation
 * for this type is the same as the one for DDS_RTPS_GUID_t.
 */
typedef struct DDS_GUID_t {
DDS_Octet value[DDS_GUID_LENGTH];
} DDS_GUID_t;
```

To convert to/from DDS_GUID_t and DDS_RTPS_GUID_t, the following two APIs have been added to the C API:

```
/*
 *  @brief Gets a GUID from an RTPS GUID.
 */
DDS_GUID_t* DDS_RTPS_GUID_as_guid(DDS_RTPS_GUID_t*);

/*
 *  @brief Gets an RTPS GUID from a GUID.
 */
DDS_RTPS_GUID_t* DDS_GUID_as_rtps_guid(DDS_GUID_t*);
```

Note that the wire representation and memory mapping of these two types is the same.

10 Performance

10.1 Improved CPU usage when setting autopurge_disposed_instances_delay or autopurge_unregistered_instances_delay

When the number of instances published by the *DataWriter* was large, high CPU usage may have occurred when **writer_qos.writer_data_lifecycle.autopurge_disposed_instances_delay** or **writer_qos.writer_data_lifecycle.autopurge_unregister_instances_delay** was set to a finite value in combination with setting the property **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay** to 1. This problem has been resolved.

10.2 Performance improvements to algorithm that purges unregistered/disposed instances on DataWriter configured with finite purging delay

This release introduces performance improvements to the algorithm that purges unregistered/disposed instances on a *DataWriter* when the QoS value **writer_data_lifecycle.autopurge_disposed_instances_delay** or **writer_data_lifecycle.autopurge_unregistered_instances_delay** (or both) is set to a finite value other than zero.

10.3 Performance improvements for content filters

The performance impact of a SQL content filter (for example, in a ContentFilteredTopic) was significant, especially as the data size increased. The content filter algorithm deserialized each sample completely before evaluating the filter expression.

In this release, this algorithm has been improved to deserialize the least possible data required to evaluate the filter expression.

This improvement applies to all the language APIs except the C API, which didn't require the deserialization step in the first place and was already efficient.

11 Logging

11.1 Error messages on the write operation now print the TopicName

The error messages on the **DataWriter::write** operation that are generated because the sample cannot be added to the *DataWriter* queue now print the TopicName.

11.2 Enhanced timestamp format of logged messages

The format of the timestamp reported by the NDDSConfigLogger has been updated to the UTC format. In previous releases, timestamps were logged in the form "ssssss.mmmmmm" where <ssssss> is a number of seconds, and <mmmmm> is a fraction of a second expressed in microseconds. The format of the timestamp will now be YYYY-MM-DD HH:MM::SS.<microseconds>, where SS is the number of seconds and <microseconds> is a fraction of that second expressed in microseconds.

12 Changes to Default QoS Values

12.1 Set default DDS_DomainParticipantQos::wire_protocol::rtps_auto_id_kind to DDS_RTPS_AUTO_ID_FROM_UUID

The default value of **DDS_DomainParticipantQos::wire_protocol::rtps_auto_id_kind** has changed from DDS_RTPS_AUTO_ID_FROM_IP (which caused the **rtps_host_id** to be the IP address by default) to DDS_RTPS_AUTO_ID_FROM_UUID (which causes the **rtps_host_id** to be a unique, randomly-generated value). This change was done to comply with the RTPS specification and reduce the possibility of non-unique GUIDs due to process ID collision.

There are backward compatibility issues as a result of this change. See the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

12.2 Set DataReader's reader_resource_limits.max_app_ack_response_length to 1

The default value for **reader_qos.reader_resource_limits.max_app_ack_response_length** has been changed from 0 to 1 to allow the use of the *DataReader* **acknowledge_sample()** API.

12.3 Set default DDS_DomainParticipantQos::resource_limits::type_code_max_serialized_length to 0

This change disables sending TypeCode by default.

There are backward compatibility issues as a result of this change. See the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

12.4 Increase default DDS_DomainParticipantQos::resource_limits::type_object_max_serialized_length to 8KB

The default value of **type_object_max_serialized_length** has been increased from 3KB to 8KB.

12.5 Set default DDS_TypeConsistencyEnforcementQosPolicy::kind to DDS_AUTO_TYPE_COERCION

The default value of **kind** has been changed from DDS_ALLOW_TYPE_COERCION to DDS_AUTO_TYPE_COERCION. For a regular *DataReader*, AUTO translates to DDS_ALLOW_TYPE_COERCION. For a Zero Copy *DataReader*, this default value is translated to DISALLOW_TYPE_COERCION.

13 Packaging and Installer: Find Package CMake® script

CMake allows you to find third-party libraries using the FIND_PACKAGE macro. This macro will search for a Find<Package Name>.cmake script, which will set all the CMake variables you need to link against the third-party dependency (in this case, the *RTI Connext DDS* libraries).

Connext DDS 6.0.0 now includes a FindRTIConnextDDS.cmake CMake script to make it easier to link your application against the *Connext DDS* libraries. Mainly, this script searches for the *RTI Connext DDS* libraries and provides them in some CMake variables.

The CMake script contains all the documentation related to the output variables, including what CMake components are available. It is compatible with the following platforms listed in the *RTI Connext DDS Core Libraries Platform Notes*:

- All the Linux i86 and x64 platforms
- Raspbian Wheezy 7.0 (3.x kernel) on ARMv6 (armv6vfphLinux3.xgcc4.7.2)

- Android 5.0 and 5.1 (armv7aAndroid5.0gcc4.9ndkr10e)
- All the Windows i86 and x64 platforms
- All the Darwin platforms (OS X 10.11-10.13)

The following fixes and improvements were made in 6.0.0 to the FindRTIConnextDDS.cmake originally shipped in 5.3.0.8. These fixes were made after 5.3.0.17.

13.1 Added a way to disable the module version check in the FindRTIConnextDDS script

The **FindRTIConnextDDS** script checks if the version of all the installed modules is consistent. Now, it is possible to disable this behavior by setting the CONNEXTDDS_DISABLE_VERSION_CHECK variable to true in your CMake scripts before calling the **FindRTIConnextDDS** script.

[RTI Issue ID BUILD-870]

13.2 Added FOUND CMake variable for each component in the FindRTIConnextDDS script

Now, the FOUND CMake variable is set for all the CMake components defined in the **FindRTIConnextDDS** script. The definition of these variables helps you check if the desired components were found by CMake.

[RTI Issue ID BUILD-872]

13.3 Added imported targets for the RTI ConnextDDS libraries to the FindRTIConnextDDS script

Some imported targets were added to the **FindRTIConnextDDS** script to make it easier to use the RTI Connext DDS libraries. The list of the new imported targets was added to the header of the **FindRTIConnextDDS** script.

[RTI Issue ID BUILD-874]

14 Other

14.1 Integration with RTI Connext DDS Micro: unified XSD schema for QoS policies

A unified schema has been created to support the QoS policies from both *RTI Connext DDS Professional* and *RTI Connext DDS Micro*. As part of this integration, the following improvements have been addressed in the *Connext DDS* XML parser:

- Deleted the default values from the XSD schema.
- Added support for *Connext DDS Micro*-specific fields and values. *Micro*-specific fields/values will be ignored by *Connext DDS Professional*, and a warning will be logged to show that they were ignored.
- Added ability to configure builtin transports from an XML QoS Profile. The available transports are udpv4, udpv6, and shmem. See 7.4 Support for strongly-typed XML elements to configure builtin transports on page 14.
- Added ability to choose Static Discovery (LBED) as a builtin discovery plugin. The LBED discovery plugin still has to be configured as described in the *RTI Connext DDS Core Libraries User's Manual*.

14.2 Dynamically link Spy, Ping, and Prototyper

The utilities DDS Spy, DDS Ping, and RTI Prototyper are now linked dynamically. Previously, running utilities (linked statically) with libraries such as security plugins (linked dynamically) could fail, and was not safe. Now, these utilities can use *Connext DDS* dynamic libraries such as security or low-bandwidth plugins. The utilities are linked dynamically for all architectures except INTEGRITY, iOS, and VxWorks RTP.

14.3 Support for finite values in writer_data_lifecycle.autopurge_disposed_instances_delay

This release adds support for finite values in **writer_data_lifecycle.autopurge_disposed_instances_delay** for in-memory *DataWriter* queues.

The purging of the disposed instances can be done based on the dispose sample source timestamp or the time when the dispose sample was added to the *DataWriter* queue, by setting the following property to 1 or 0 respectively: **dds.data_writer.history.source_timestamp_based_autopurge_instances_delay**. Default value is 0.

This feature is not supported yet with durable *DataWriter* queues.

14.4 New field messageId added to com::rti::dl::LogMessage in RTI Distributed Logger

This release adds a new field called **messageId** to the **com::rti::dl::LogMessage** Type Name used by *RTI Distributed Logger* to publish log messages.

This field uniquely identifies each one of the messages published by the *DistributedLogger* instance running a process. The *DistributedLogger* instance itself is identified using the existing field **hostAndAppId**.

The first message published by *Distributed Logger* has a **messageId** set to 1, and the number increases by one with each message.

The **messageId** field allows applications to detect losses in log messages. Log messages can be lost when the internal *Distributed Logger* queue is full. (The queue's capacity can be configured by using the API **RTI_DL_Options_setQueueSize**.) The messages can also be lost when they are published by the *Distributed Logger's* LogMessage *DataWriter*. For more information, see the new Troubleshooting section in "Chapter 41 Using Distributed Logger in a Connext DDS Application" in the *RTI Connext DDS Core Libraries User's Manual*.

14.5 Heap monitoring improvements to fragment assembly code path when receiving large data

This release introduces heap monitoring improvements to the fragment assembly code path when receiving large data (data that cannot be sent as a single packet by a transport). Specifically, the memory allocations in this code path in the snapshot file will have the topic name associated with them.

14.6 Support for configuring initial_virtual_sequence_number on DataWriter

By default, the first sample published by a *DataWriter* will have the virtual sequence number 1. This release adds a new QoS parameter, **DataWriterProtocolQosPolicy::initial_virtual_sequence_number**, that allows configuring the virtual sequence number of the first sample published by a *DataWriter* to a different value.

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

14.7 New file extension for modern C++ libraries for VxWorks systems

For VxWorks systems, the modern C++ libraries now use the **.so** extension (instead of **.lo**), for consistency with the other libraries.