RTI Connext

Core Libraries and Utilities

What's New

in Version 5.0.0

Your systems. Working as one.

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

This document highlights new or changed features in *RTI*® *Connext*™ (formerly *RTI Data Distribution Service*) 5.0.0 compared to the previous release, 4.5f. (For details on fixed bugs, please see the *Release Notes*.)

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

1 New Platforms

This release adds support for the platforms described in Table 1.1. Please see the updated *Platform Notes* for details on using these platforms.

Table 1.1 **New Platforms**

Operating System		CPU	Compiler	RTI Architecture Abbreviation
INTEGRITY®	INTEGRITY 10.0.2	x86	multi 5.0.6	pentiumInty10.0.2.pcx86
Linux®	CentOS 6.1, 6.2	x86	gcc 4.4.5	i86Linux2.6gcc4.4.5
		x64	gcc 4.4.5	x64Linux2.6gcc4.4.5
	Red Hat® Enterprise Linux 6.2	x86	gcc 4.4.5	i86Linux2.6gcc4.4.5
		x64	gcc 4.4.5	x64Linux2.6gcc4.4.5
	Wind River Linux 3.0.3 (2.6 kernel) Note: This platform is only available as a Custom Target Library (CTL).	Pentium class	gcc 4.3.2	i86WRLinux2.6gcc4.3.2
	Wind River Linux 4 (2.6 kernel)	Pentium class	gcc 4.4.1	x64WRLinux2.6gcc4.4.1
VxWorks®	VxWorks MILS 2.1.1 with vThreads 2.2.3	ppc85xx	gcc 3.3.2	ppc85xxVxT2.2.3gcc3.3.2

The platforms in Table 1.2 were supported in 4.5f but are not supported in this release.

Table 1.2 **Removed Platforms**

Operating System		CPU	Compiler	RTI Architecture Abbreviation
INTEGRITY	INTEGRITY 10.0.0	x86	multi 5.0.6	pentiumInty10.0.0.pcx86
LynxOS	All Java platforms			
Solaris	Solaris 2.9	Ultra-SPARC	gcc 3.2	sparcSol2.9gcc3.2
	Solaris 2.10	AMD64	gcc 3.4.3	x64Sol2.10gcc3.4.3
			Sun Java Platform Standard Edition JDK 1.5 or 1.6	x64Sol2.10jdk
		x86	gcc 3.4.4	i86Sol2.10gcc3.4.4
			Sun Java Platform Standard Edition JDK 1.5 or 1.6	i86Sol2.10jdk
Windows	Windows 2000	x86	VS 2005 SP 1	i86Win32VS2005

2 New Language Support

This release adds support for Java 7. Java 5 is no longer supported.

3 Application-Level Acknowledgments

Application-level acknowledgments extend reliability so that samples are not considered acknowledged—and therefore are available for redelivery—until they have been successfully *processed*, not just successfully *received*.

Depending on whether samples are processed synchronously or asynchronously with respect to their reception, samples may require acknowledgment out of order. For example, suppose A sends three samples to B, which processes them in parallel. B will not necessarily finish processing the samples in the order received, so an acknowledgment of a later sample does not imply acknowledgment of an earlier one. The acknowledgments are independent and may be sent in any order.

Connext extends the existing reliability model to support application-level acknowledgments, so individual samples can be acknowledged independent of order. Multiple acknowledgment modes allow you to decide the appropriate level of reliability for each topic.

Note that application-level acknowledgment does not guarantee delivery of a sample; the feature still operates in the context of how the reliability protocol is configured. For example, if you have a *DataWriter* with a HistoryQosPolicy setting of KEEP_LAST, sample availability will be based on memory constraints, regardless of confirmed receipt by subscribers.

For additional information about this feature, see Section 6.3.12, *Application Acknowledgment*, in the *Core Libraries and Utilities User's Manual*.

4 Collaborative DataWriters

The Collaborative DataWriters feature allows you to have multiple *DataWriters* that publish samples from a common data source. *DataReaders* will combine samples from these *DataWriters* to reconstruct the order to match the source.

The combination of samples in the *DataReader* is configured using the AvailabilityQosPolicy.

There are two main uses cases for the Collaborative DataWriters feature:

- ❏ **Redundancy and fault tolerance:** Multiple *DataWriters* can be used to publish the same samples coming from a single data source.

  For example, when persistence service is used in peer-to-peer mode, the same samples are published by multiple *DataWriters*: the original *DataWriter* and one or more persistence service *DataWriters*. If the original *DataWriter* goes away, late joiners can receive the samples from the Persistence Service DataWriters.

- ❏ **Load balancing and scalability:** Multiple *DataWriters* can be used to publish different sets of samples from a single data source.

  For example, multiple persistence service *DataWriters* can be used to publish different sets of samples from the original *DataWriter* using ContentFilteredTopics.

To reconstruct the sample order of the data source, the *DataReaders* need to correlate the incoming samples. Each sample published in a DDS domain is uniquely identified by a pair of numbers (virtual GUID, virtual sequence number):

- ❏ The virtual GUID (Global Unique Identifier) is a 16-byte character identifier associated with the logical data source. With Collaborative DataWriters, a *DataWriter* can be used to publish samples coming from a single data source or from multiple data sources. In the latter case, the virtual GUID can be configured on a per sample basis using the new API, **write_w_params()**.
- ❏ The sequence number is a 64-bit identifier that identifies changes within a data source.

For additional information about this feature, see Chapter 11, *Collaborative DataWriters,* in the *Core Libraries and Utilities User's Manual*.

5 Required And Durable Subscriptions

5.1 Required Subscriptions

The DurabilityQosPolicy specifies whether acknowledged samples need to be kept in the *DataWriter's* queue and made available to late-joining applications. When a late joining application is discovered, available samples will be sent to the late joiner. With the DurabilityQosPolicy alone, there is no way to specify or characterize the intended consumers of the information and the user does not have control over which samples will be kept for late-joining applications. If while waiting for late-joining applications, the middleware needs to free up samples, it will reclaim samples if they have been previously acknowledged by active/matching *DataReaders*.

With the Required Subscriptions feature, you can configure a *DataWriter* using the Availability-QosPolicy to keep samples in its queue until they are received and acknowledged by a set of named *DataReaders*.

The Required Subscriptions feature only works with *DataWriters* configure as RELIABLE reliability and NON-VOLATILE durability.

For additional information about this feature see Section 6.3.13, *Required Subscriptions*, in the *Core Libraries and Utilities User's Manual*.

5.2 Durable Subscriptions

The Durable Subscriptions feature allows you to configure *RTI Persistence Service* to persist samples intended for Required Subscriptions such that they are delivered even if the originating *DataWriter* is not available.

An application can configure the list of Durable Subscriptions in *RTI Persistence Service* by providing this list as part of the service's XML configuration or by invoking the *DomainParticipant's* new **register_durable_subscription()** operation.

Note: The Durable Subscription feature is only available with *RTI Connext Messaging* as it requires *RTI Persistence Service*.

For additional information about this feature see Section 27.9, *Configuring Durable Subscriptions,* and Chapter 13, *Guaranteed Delivery of Data*, in the *Core Libraries and Utilities User's Manual*.

6 Extensible Types

This release includes partial support for the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification[1] from the Object Management Group (OMG). This support allows systems to define data types in a more flexible way, and to evolve data types over time without giving up portability, interoperability, or the expressiveness of the DDS type system.

Specifically, the following are now supported:

- ❑ Type definitions are now checked as part of the *Connext* discovery process to ensure that *DataReaders* will not deserialize the data sent to them incorrectly.
- ❑ Type definitions need not match exactly between a *DataWriter* and its matching *DataReaders*. For example, a *DataWriter* may publish a subclass while a *DataReader* subscribes to a superclass, or a new version of a component may add a field to a preexisting data type.
- ❑ Data-type designers can annotate type definitions to indicate the degree of flexibility allowed when the middleware enforces type consistency.
- ❑ The above features are supported in the RTI core middleware in the C and C++ programming languages, as well as in the *RTI Routing Service* component.

For details on using this new feature and for the list of features that are not supported, please see the new *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types* (**RTI_CoreLibrariesAndUtilities_GettingStarted_ExtensibleTypesAddendum.pdf**).

6.1 Extensible Types Support in rtiddsspy

rtiddsspy includes limited support for Extensible Types. Please see the new *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types* (**RTI_CoreLibrariesAndUtilities_GettingStarted_ExtensibleTypesAddendum.pdf**) for additional details.

1. http://www.omg.org/spec/DDS-XTypes/

7 Support for Sending 'Large Data' During Discovery

Now you can configure *Connext* so that 'large data' can be sent during endpoint discovery. The size of an endpoint discovery message is no longer limited by the underlying transport's maximum message size because the discovery data packets can be sent in fragments.

With this new feature, you can enable asynchronous publishing for the built-in Publication and Subscription DataWriters using the new fields **publication_writer_publish_mode**, **subscription_writer_publish_mode**, and **asynchronous_publisher** in the DiscoveryConfigQosPolicy for the *DomainParticipant*.

By configuring the built-in *DataWriters* to use asynchronous publishing, you can create *DataWriters* and *DataReaders* with large UserDataQosPolicy and TypeObjects values.

8 Support for Sending Large TypeObjects

Previous versions of *Connext* (prior to 5.0) used TypeCodes as the wire representation to communicate types over the network.

One limitation of using TypeCodes as the wire representation is that their serialized size is limited to 65 KB. This was a problem for services and tools that depend on the discovered types, such as *RTI Routing Service* and *RTI Spreadsheet Add-in for Microsoft Excel*.

The Extensible Types specification and *Connext* 5.0 provide an alternative wire representation for types called TypeObject. One advantage of using TypeObjects to propagate type information is that they are not subject to the 65 KB limitation of TypeCodes.

To learn about TypeObjects and how to send them on the wire, please see the new *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types* (**RTI_CoreLibrariesAndUtilities_GettingStarted_ExtensibleTypesAddendum.pdf**).

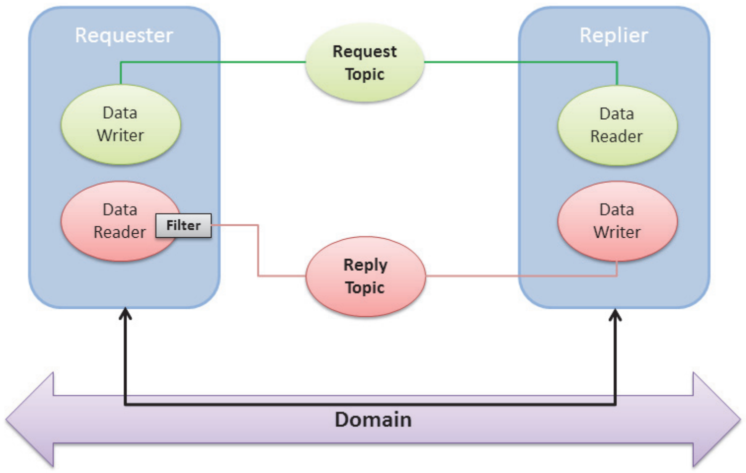
Note that sending large TypeObjects with discovery information will also require enabling asynchronous publishing in the built-in topic DataWriters (see Section 7).

9 Request-Reply Communication

This release introduces support for a Request-Reply communication pattern; this is only available with *RTI Connext Messaging*.

As applications become more complex, it often becomes necessary to use other communication patterns in addition to publish-subscribe. Sometimes an application needs to get a one-time snapshot of information; for example, to make a query into a database or retrieve configuration parameters that never change. Other times an application needs to ask a remote application to perform an action on its behalf; for example, to invoke a remote procedure call or a service. To support these scenarios, *Connext Messaging* includes support for the request-reply communication pattern.

An application uses a *Requester* to send requests to a *Replier*; another application using a *Replier* receives a request and can send one or more replies for that request. The *Requester* that sent the request (and only that one) will receive the reply (or replies).



A *Requester* uses an existing *DomainParticipant* to communicate through a domain. It owns a *DataWriter* for writing requests and a *DataReader* for receiving replies. Similarly, a *Replier* uses an existing *DomainParticipant* to communicate through a domain and owns a *DataReader* for receiving requests and a *DataWriter* for writing replies.

Requesters and *Repliers* are built on top of DDS. There are APIs in C, C++, Java and C#; these APIs have been designed to be closer to the programming language. In particular, the C++ and Java Request-Reply APIs include significant improvements with respect to the traditional DDS API, such as the use of generic code (templates in C++ and generics in Java) or close integration with standard language features and utilities (like the C++ STL, and Java Collections).

New Request-Reply examples are provided for C, C++, C#, and Java in their respective **<*install dir*>/example/<*language*>** directories. For more information on using the examples, see the *RTI Core Libraries and Utilities Getting Started Guide*. The *User's Manual* also includes two new chapters in *Part 4: Request-Reply Communication Pattern*.

10 Multicast Support for VxWorks 653 Platforms

Multicast is now supported on VxWorks 653 platforms. For details on using this platform, see the *RTI Core Libraries and Utilities Platform Notes*.

11 New Command-Line Option for rtiddsspy: -partition

This release introduces a new command-line option for *rtiddsspy* that provides a list of partitions. For example:

```
rtiddsspy -domain 56 -partition P1
```

To add multiple partitions, use -partition multiple times. For example:

```
rtiddsspy -domain 56 -partition P1 -partition P2
```

The default behavior of *rtiddsspy* has been changed so that it now subscribes to the "*" partition instead of the empty partition.

12 Support for related_sample_identity

There is one new field in the **DDS_WriteParams_t** used by the **write_w_params** API: **related_sample_identity**.

The value of the **related_sample_identity** field identifies another sample that is logically related to the one that is written. For example, the *DataWriter* created by a Replier (see Section 9) uses this field to associate the identity of the request sample to response sample. A *DataReader* can inspect the related sample identity of a received sample by inspecting the content of the fields **related_original_publication_virtual_guid** and **related_original_publication_virtual_sequence_number** in the SampleInfo Structure.

13 Ability to Filter on Meta Data

The ContentFilter's 'evaluate' function now receives a new 'struct DDS_FilterSampleInfo *' parameter. Because the DDS_FilterSampleInfo structure contains the meta-data associated with a sample, now you can perform filtering not just on the sample's content, but also on sample's meta-data.

Note: Currently the struct DDS_FilterSampleInfo only supports filtering on **related_sample_identity** (see Section 12).

Future releases will provide the option of filtering on additional meta-data such as the **source_timestamp**.

14 Increased Resource Limit: max_remote_reader_filters

The valid range of values for **max_remote_reader_filters** in the DataWriterResourceLimitsQo-sPolicy has been increased from 32 to $2^{(31-2)}$; it may also be set to DDS_LENGTH_UNLIMITED. The default remains 32.

This field controls the maximum number of remote *DataReaders* for which the *DataWriter* will perform content-based filtering.

The values have the following meaning:

- ❏ 0: The *DataWriter* will not perform filtering for any *DataReader*.
- ❏ 1 to $2^{(31-2)}$: The *DataWriter* will filter for up to the specified number of *DataReaders*. In addition, the *DataWriter* will store the result of the filtering per sample per *DataReader*.
- ❏ DDS_LENGTH_UNLIMITED: The *DataWriter* will filter for up to $2^{(31-2)}$: *DataReaders*. However, in this case, the *DataWriter* will not store the filtering result per sample per *DataReader*. Thus, if a sample is resent (such as due to a loss of reliable communication), the sample will be filtered again.

15 Support for Writer-side Filtering no Longer Requires DataReader to Propagate TypeCode

In previous releases, writer-side filtering required the *DataReader* to transmit the typecode. This is no longer required because now writer-side filtering uses the typecode registered with the *DataWriter*.

16 Limited Support for Writer-Side Filtering when Asynchronous Publishing Enabled

Previous releases did not support writer-side filtering when using a ContentFilteredTopic in combination with asynchronous publishing. All filtering was done on the reader side.

This release introduces limited support for writer-side filtering with asynchronous publishing in order to save bandwidth.

Connext will not send any sample to a destination if the sample is filtered out by all the *DataReaders* at that destination. However, if there is one *DataReader* to which the sample must be sent, then all the *DataReaders* at the destination will perform reader-side filtering for the incoming sample.

17 New APIs for Writer-Side Filtering

There is a new set of APIs for Custom Filters that facilitate writer-side filtering. These new APIs enable the filter implementation to cache and reuse the filtering results. See Section 5.4.8.1, *Filtering on the Writer Side with Custom Filters*, in the *Core Libraries and Utilities User's Manual* for more details.

18 Enhanced Scalability

18.1 CPU Usage Scalability Enhancements With Writer-Side Filtering

In previous releases, a *DataWriter* performing writer-side filtering had to iterate over all matching *DataReaders* that used a ContentFilteredTopic each time a new sample was published. This O(*n*) iteration was needed to decide whether the sample had to be sent to the *DataReader* or not.

To remove this linear scalability component from the write path, the *DataWriter* logic has been updated so that an iteration is not needed over *DataReaders* using ContentFilteredTopics with a filter expression that is either using key-only fields or using only the meta-data field **related_sample_identity.**

If all the *DataReaders* that use ContentFilteredTopics use filter expressions as described above, the execution order on the write path is near O(1).

18.2 Bandwidth Usage Scalability Enhancements

In previous releases, when a sample was filtered out for a *DataReader*, the *DataWriter* would send a GAP on the wire for that *DataReader* to indicate that the sample was filtered out. This resulted in inefficient utilization of the network bandwidth since for every sample published, the *DataWriter* had to send a GAP or DATA message to every matching *DataReader*.

In this release, the *DataWriter* logic has been modified to coalesce consecutive GAPs into a single GAP that is sent on the wire the next time the *DataWriter* has to send a sample to the *DataReader*.

For example, assume that a *DataWriter* publishes samples S1, S2, S3 and S4 and a *DataReader* is only interested in S4. In previous releases, the *DataWriter* sent four RTPS messages: a GAP for S1, a GAP for S2, a GAP for S3, and a DATA for S4. In this release, the *DataWriter* only sends two RTPS messages: a GAP for S1-S3, and a DATA for S4.

Coalescing GAP messages not only reduces bandwidth consumption, it also optimizes CPU usage since the amount of reliability traffic is smaller.

19 Support for Asynchronous DataWriters push_on_write Set to False

Previously, setting the DataWriterProtocolQosPolicy's **push_on_write** field to false for *DataWriters* using asynchronous publishing would cause samples to never be sent. Now, this configuration is supported and samples will be sent.

20 Support for System Properties

Connext will now initialize the *DomainParticipant's* PropertyQosPolicy with a set of properties that provide system information such as the hostname. The system properties supported in this release are:

- ❏ **dds.sys_info.hostname**
- ❏ **dds.sys_info.process_id**

These properties are not intended to be set by the user; instead they are set by *Connext* to communicate system information.

These properties are automatically assigned to their correct values using system calls where supported. If a property is not supported on a specific architecture, it will not be set. In this release, the **hostname** is only supported on Linux and Windows platforms. The **process_id** is supported on all platforms.

If a property has not been set and you try to retrieve it using functions such as **DDS_PropertyQosPolicyHelper_lookup_property()** or **DDS_PropertyQosPolicyHelper_get_properties()**, a NULL value or an empty sequence is returned.

By default, these properties are propagated during the discovery process. You can disable propagation by setting the property's **propagate** field to FALSE.

For more details on System Properties, see the new *Systems Properties* section (Section 8.7) in the *RTI Core Libraries and Utilities User's Manual* or consult the API Reference HTML documentation (under **Modules, RTI Connext API Reference, System Properties**).

21 Support for Environment Variables in XML Configuration Files

This new feature allows you to refer to an environment variable within an XML tag. When the *Connext* XML parser parses the configuration file, it will expand the environment variable. To refer to an environment variable, use the format $(MY_VARIABLE).

For example:

```
<element>
    <name>The name is $(MY_NAME)</name>
    <value>The value is $(MY_VALUE)</value>
</element>
```

Being able to refer to an environment variable within an XML file increases XML reusability. For example, the new feature could be used to specify the initial peers, so you do not need to use multiple XML files or XML profiles per application.

22 Support for Property Inheritance in XML Qos Profiles

In previous releases, sequences defined in a derived QoS profile replaced the corresponding sequences in the base QoS profile. In this release, the <property> tag accepts a new attribute, **inherit**, which allows you to choose the inheritance behavior for the sequence defined in PropertyQosPolicy.

By default, the value of the attribute **inherit** is TRUE. Therefore, the <property> defined within a derived XML QoS profile will inherit its values from the property defined within a base XML QoS profile.

Important Change in Behavior: The default value of the **inherit** attribute (TRUE) changes the inheritance behavior with respect to previous releases, where the behavior was to overwrite all the properties defined in the base profile.

For more information, see Section 17.8.2, *Sequences,* in the *Core Libraries and Utilities User's Manual*.

23 Support for User-Defined Logging Devices to Customize Handling of Log Messages

In previous releases, log messages from *Connext* were redirected to the standard output. Users could optionally redirect the log messages to an output file using the operation **ConfigLogger::set_output_file()**.

This release introduces the concept of *logging devices*, which allow you to customize how the log messages generated by *Connext* are handled.

For example, you can send the log messages over the network by implementing and installing a logging device with the logger.

For more details on logging devices, see Section 20.2, Controlling Messages from Connext, in the *Core Libraries and Utilities User's Manual* or consult the API Reference HTML documentation (under **Modules, RTI Connext API Reference, Configuration Utilities**).

24 Ability to Configure Memory Allocation Schemas for Samples in DataWriter and DataReader Queues

This release introduces several configuration settings that allow more flexible memory allocation schemas for the samples in *DataWriter* and *DataReader* queues.

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

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

For more details on these configuration parameters, see Chapter 19, *Sample-Data Memory Management*, in the *Core Libraries and Utilities User's Manual.*

25 Ability to Specify Buffers where DynamicData Objects Store Values—C and C++ APIs Only

This release introduces two new DynamicData APIs in C/C++:

- ❏ **set_buffer()** allows you to specify the underlying buffer in which a DynamicData object will store its values. If you do not use this operation, then by default, the DynamicData object will manage the memory allocated for storing its values.
- ❏ **get_estimated_max_buffer_size()** is a helper operation that returns an estimate of the maximum size for the buffer provided to **set_buffer()**.

For more details, see the *Core Library and Utilities* API Reference HTML documentation, which is available for all supported programming languages (select **Modules, DDS API Reference, Topic Module, Dynamic Data**).

26 Experimental Features

Experimental features are used to evaluate potential new features and obtain customer feedback. They are not guaranteed to be consistent or supported and they should not be used in production.

Experimental features are clearly documented as such in the *Core Libraries and Utilities What's New* document or the *Release Notes* document of the component in which they are included, as well as in the component's *User's Manual*.

For more information on the concept of experimental features, see the *Core Libraries and Utilities Release Notes*.

26.1 Experimental Feature: XML-Based Application Creation

In this release, you can continue evaluating the XML-Based Application Creation experimental feature introduced in *Connext* 4.5f. This feature simplifies the development and programming of *RTI Connext* applications by allowing you to define your *entire* system using XML.

With previous releases, you could define data types and Quality of Service settings in XML. With this feature, you can also use XML to define your system's *Topics*, *DomainParticipants*, and all the *Entities* they contain (*Publishers*, *Subscribers*, *DataWriters* and *DataReaders*).

Connext 5.0 adds support for two new languages: Java and .NET.

For details on using this feature, see the *XML-Based Application Creation Getting Started Guide (***RTI_CoreLibrariesAndUtilities_XML_AppCreation_GettingStarted.pdf***)*.

Note: Experimental features cannot be used in a production system. See the *Core Libraries and Utilities Release Notes* for more information.

26.2 Prototyper Application to Determine Key Performance Indicators

Connext Prototyper is a tool to accelerate *Connext* application development and scenario testing. It gives you an easy way to try out realistic scenarios on your own computer systems and networks, and get immediate information on the expected performance, resource usage, and behavior on your system.

With the traditional approach, if you want to test a specific *Connext* distributed application design and determine Key Performance Indicators (KPIs), you would have to spend significant time and effort to develop a custom prototype that could determine KPIs such as:

- ❑ The memory a particular application is likely to use.
- ❑ The time it will take for discovery to complete.
- ❑ The system bandwidth the running application will consume.
- ❑ The CPU usage it will take for a particular application to publish its data at a certain rate, or to receive a certain set of Topics.
- ❑ The impact of changing data-types, Topics, Quality of Service, and other design parameters.

Using the Prototyper, this process is significantly simplified. Instead of writing custom code, you can:

1. Describe the system in an XML file (or files),
2. Run *Prototyper* on each computer, specifying the particular configuration for that computer, and
3. Observe the behavior of the running system and read the KPIs using *RTI Monitor.*

Prototyper is included with *RTI Connext DDS* and *RTI Connext Messaging.* See its *Getting Started Guide* for more information (in **<*Connext installation directory*>/ndds.<*version*>/doc/pdf**).

27 Other New Features

27.1 New Arithmetic Methods for Duration_t in Java API

The class Duration_t includes three new convenient arithmetic methods: **add()**, **subtract()**, and **from_nanos()**.

27.2 New 'print_IDL()' in TypeCode Class Sends Results to Output Stream—Java API Only

The class TypeCode in Java includes a new method, **print_IDL()**, which sends the pseudo-IDL result to an output stream.

```
public void print_IDL(int indent, java.io.Writer out) throws IOException
```

The old **print_IDL()** method that sends the result to the output console has been kept for backward compatibility.

27.3 Support for DataWriterCacheStatus in DataWriters using Durable Writer History

This release adds the ability to get the DataWriterCacheStatus from a *DataWriter* using Durable Writer History.

28 Documentation Changes

28.1 New Filenames for Core Libraries and Utilities PDF Documentation

The PDF files for the *Core Libraries and Utilities* were named **RTI_Connext_<name>.pdf**. In this release, these files have been renamed to **RTI_CoreLibrariesAndUtilities_<name>.pdf** (for example, **RTI_CoreLibrariesAndUtilities_UsersManual.pdf**).

28.2 Improved Instructions for Working with VxWorks 653 and VxWorks MILS Platforms

The *Getting Started Guide Addendum for Embedded Systems* (**RTI_CoreLibrariesAndUtilities_GettingStarted_EmbeddedSystemsAddendum.pdf**) has been updated to make it easier to set up a *Connext* application for a VxWorks 653 platform with a SimPC CPU (simpcVx653-2.3gcc3.3.2). The document also has a new chapter with instructions for using VxWorks MILS 2.1.1 systems.