RTI Data Distribution Service

The Real-Time Publish-Subscribe Middleware

What's New in Version 4.5c

Printed in U.S.A. First printing.
June 2010.

Trademarks

Copy and Use Restrictions

Technical Support

Real-Time Innovations, Inc.
385 Moffett Park Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: http://www.rti.com/support

Contents

What's New

This document highlights new or changed features in *RTI® Data Distribution Service* 4.5c and 4.5b.

For more information, visit the RTI Knowledge Base, accessible from http://www.rti.com/support, to see sample code, general information on *RTI Data Distribution Service*, 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 What's New in 4.5c

This section describes what's new in 4.5c compared to 4.5b.

1.1 New Architectures

- ❏ Windows 7 (32/64-bit) and Windows Server 2008 R2 (64-bit) with Visual Studio® 2010 support
- ❏ Mac OS® X on x64
- ❏ VxWorks® 6.7
- ❏ Fedora™ 9 on Cell BE
- ❏ SELinux on PowerPC
- ❏ .Net 64-bit support

1.2 New Transports

- ❑ TCP/IPv4 transport on selected architectures (see the *Platform Notes*), with optional TLS support.

1.3 Support for Shared Memory on Windows CE Devices

The shared-memory transport is now supported on Windows® CE devices.

1.4 Database Updates for Durable Writer History and Durable Reader State

- ❑ Oracle® Database 11g
- ❑ TimesTen® 11
- ❑ MySQL 5.1

1.5 Send-Window Feature

The 'send window,' configurable via RtpsReliableWriterProtocol_t , sets the limit on the number of outstanding unacknowledged samples of a *DataWriter* and blocks the *DataWriter* whenever the limit is reached.

With this release, the size of the *DataWriter's* reliability window has been decoupled from its history queue. The *DataWriter's* queue size can now be set optimally for history and durability concerns, while the send window is configured to achieve the best reliable performance. The send window can be configured to be static or to change dynamically based on detected network congestion.

The current send-window size can be obtained via a new field called **send_window_size** in DataWriterProtocolStatus.

There are other related new fields in RtpsReliableWriterProtocol_t (used in DataWriterProtocolQosPolicy and DiscoveryConfigQosPolicy) for send-window configuration:

- ❑ **min_send_window_size**
- ❑ **max_send_window_size**
- ❑ **send_window_update_period**
- ❑ **send_window_increase_factor**
- ❑ **send_window_decrease_factor**

1.6 Instance Replacement and Auto-Registration of Replaced Instances

The new **instance_replacement** and **replace_empty_instances** fields in the DataWriterResourceLimitsQosPolicy configure the reuse of existing instances once a *DataWriter* has used all of its allotted instance resources. When the maximum instances have been used, instead of failing upon registration or writing a new instance, **instance_replacement** configures which kinds of instances the *DataWriter* may try to reclaim to use with the new instance.

Users can be notified via a new DataWriterListener callback, **on_instance_replaced()**. This callback is triggered when a replacement of an existing instance by a new instance.

To prevent a **write()** call from failing because an instance has been replaced (and thus unregistered), there is a new field in the DataWriterResourceLimitsQosPolicy called **autoregister_instances**. This can be used to robustly handle instances that have been replaced due to the DataWriterResourceLimitsQosPolicy's **instance_replacement** setting. Since a replaced instance is effectively unregistered, enabling **autoregister_instances** prevents writes of replaced instances from failing because of their prior replacement.

- ❏ Related new enum: DataWriterResourceLimitsInstanceReplacementKind
- ❏ Related new fields in DataWriterResourceLimitsQosPolicy:
  - **instance_replacement**
  - **replace_empty_instances**
  - **autoregister_instances**
- ❏ Related new method in DataWriterListener:
  - **on_instance_replaced**

1.7 Writer Liveliness Protocol Configuration

To support interoperability, all DDS implementations must support the Writer Liveliness Protocol. *RTI Data Distribution Service* implements this protocol using the built-in ParticipantMessage topic that uses reliable communications.

This release exposes the configuration of the *DataWriter* and *DataReader* involved in implementing the Writer Liveliness Protocol. You can set this QoS using the DiscoveryConfigQosPolicy's **participant_message_reader** and **participant_message_writer** fields.

1.8 New Protocol Statuses to Reveal Reliability State

New statuses have been added to DataWriterProtocolStatus and DataReaderProtocolStatus that reveal more detailed reliability state. For a *DataWriter*, its range of available samples and the acknowledgement level of its matched *DataReaders* are now exposed. These can be used to identify and isolate slow *DataReaders*. Likewise, for a *DataReader*, the available samples of each matched *DataWriter* have been added, as well as status on which samples it has committed and made available for a user to read or take. Status on the *DataWriter's* send-window size has also been added.

- ❏ Related new fields in DataReaderProtocolStatus:
  - **first_available_sample_sequence_number**
  - **last_available_sample_sequence_number**
  - **last_commited_sample_sequence_number**
  - **uncommitted_sample_count**
- ❏ Related new fields in DataWriterProtocolStatus:
  - **send_window_size**
  - **first_available_sequence_number**
  - **last_available_sequence_number**
  - **first_unacknowledged_sample_sequence_number,**
  - **first_available_sample_virtual_sequence_number**
  - **last_available_sample_virtual_sequence_number**
  - **first_unacknowledged_sample_virtual_sequence_number**
  - **first_unacknowledged_sample_subscription_handle**
  - **first_unelapsed_keep_duration_sample_sequence_number**

1.9 Support for Controlling Processor Affinity on Threads

While most thread-related QoS settings apply to a single thread, the ReceiverPool QoS policy's thread-settings control *every* receive thread created. In this case, there are several schemes to map *M* threads to *N* processors. With this release, you can control which scheme is used by setting the new **cpu_rotation** field in the DDS_ThreadSettings_t structure.

- ❏ Related new enum: ThreadSettingsCpuRotationKind
- ❏ Related new fields in ThreadSettings_t:
  - **cpu_list**

- **cpu_rotation**

(This feature is not supported on all architectures. See the *Platform Notes* for details.)

1.10 Ability to Choose Algorithm for Determining the GUID Prefix

By default, the GUID prefix is automatically generated from the IP address of the first up and running interface. The GUID prefix *must* be unique; however, the IP address may not be unique in all cases for a variety of reasons. To address this, a new mechanism for generating the GUID prefix has been implemented based on the MAC address, which is intended to be unique. The WireProtocol QoS policy's **rtps_auto_id_kind** selects the mechanism used to generate the GUID prefix.

The **rtps_auto_id_kind** allows you to choose the algorithm used to determine the GUID's 96-bit prefix. There are two possible values: DDS_RTPS_AUTO_ID_FROM_IP (the default, which uses an IPV4-based algorithm) and DDS_RTPS_AUTO_ID_FROM_MAC (which uses a MAC address-based algorithm).

This feature was introduced for a limited number of architectures in 4.5a, with more architectures added in 4.5b. Release 4.5c supports this feature on all architectures except INTEGRITY platforms without the ghnet2 IP stack (ppc7400Inty5.0.7.mvme5100-7400 and ppc7400Inty5.0.7.mvme5100-7400-ipk) and VxWorks 5.4.

[RTI Bug # 12325]

1.11 New Options for rtiddsgen

New command-line options have been added to *rtiddsgen*:

- **-convertToCcl**: Converts the input type description file into CCL format. This option creates a new file with the same name as the input file and a .ccl extension.
- **-convertToCcs**: Converts the input type description file into CCS format. This option creates a new file with the same name as the input file and a .ccs extension.
- **-enableEscapeChar**: Enables use of the escape character '_' in IDL identifiers.
- **-typeSequenceSuffix**: Assigns a suffix to the names of the implicit sequences defined for IDL types. It only applies if **-corba** is also specified. By default, the suffix is 'Seq'. Therefore, given the type 'Foo' the name of the implicit sequence will be 'FooSeq'.
- **-verbosity**: Controls the verbosity of messages from *rtiddsgen*. You can set it to report exceptions, warnings, and information. The default setting is for the highest verbosity.

1.12 New Options for rtiddsspy and rtiddsping

New command-line options have been added to *rtiddsspy* and *rtiddsping*:

- **-qosFile <file>**: Specifies an XML configuration file.
- **-qosProfile <lib::prof>**: Specifies the library name and profile name to be used.

1.13 Changes in Protocol Status and Cache Status Data Types

All the 'count' data types in DataReaderCacheStatus, DataReaderProtocolStatus, DataWriterCacheStatus and DataWriterProtocolStatus have been changed from 'long' to 'long long' in the C, C++ and .Net APIs in order to avoid a value over-run for DDS applications that run for very long periods of time.

1.14 long long Native Data Type Support

Starting with release 4.5c, we assume all platforms natively support the 'long long' data type. There is no longer a need to define RTI_CDR_SIZEOF_LONG_LONG to be 8 on some platforms in order to map the DDS 'long long' data type to a native 'long long' type.

1.15 XML QoS Libraries Can Be Reopened

Starting with release 4.5c, QoS libraries declared in an XML configuration file can be reopened within the same file and across different files. For example:

```
<dds>
    <qos_library name="RTILibrary">
        ...
     </qos_library>
     ...
    <qos_library name="RTILibrary">
        ...
     </qos_library>
</dds>
```

In previous releases, the parsing of the above XML code failed with a duplicate error message.

2 What's New in 4.5b

This section describes what was new in 4.5b compared to 4.4d.

2.1 New Platforms

Release 4.5b added support for the following platforms (see the *Release Notes* for details):

- Fedora 10 (x64Linux2.6gcc4.3.2)

2.2 QueryConditions

This release added support for QueryConditions, a special type of ReadCondition that allows you to filter data by specifying a query expression and parameters. This release also provides substantially better performance for ReadConditions.

The *DataReader* has a new API to create a QueryCondition called **create_querycondition()**.

The *DataReader's* **read_w_condition(), take_w_conditions(), and delete_readcondition()** APIs now support QueryConditions. Similarly, these ReadConditions APIs now also accept QueryConditions: **get_datareader()**, **get_sample_state_mask()**, **get_view_state_mask()**, and **get_instance_state_mask()**.

The QueryCondition class itself has these APIs: **get_query_expression ()**, **get_query_parameters ()**, and **set_query_parameters ()**.

There is a new field in the DomainParticipantResourceLimits QoS policy called **query_condition_allocation**, and a new field in the DataReaderResourceLimits QoS policy called **max_query_condition_filters**.

2.3 RTI Routing Service

This release introduced *RTI Routing Service,* a separate, optional solution for integrating and scaling DDS applications across domains, LANs and WANs, including firewall and NAT traversal. It can read data from one domain and write it to another domain; it also allows you to make transformations in the data along the way.

2.4 RTI TCP Transport

This release introduced *RTI TCP Transport*, a separate, optional package that enables participant discovery and data exchange using the TCP protocol (either on a local LAN, or over the public WAN).

RTI TCP Transport enables *RTI Data Distribution Service* to address the challenges of using TCP as a low-level communication mechanism between peers that limits the number of ports exposed to one. (When using the default UDP transport, an *RTI Data Distribution Service* application uses multiple UDP ports for communication, which may make it unsuitable for deployment across firewalled networks).

2.5 Reliability Improvements

The new *DataWriter* QoS field, **DDS_RtpsReliableWriterProtocol_t.inactivate_nonprogressing_readers**, allows reliable *DataWriters* to treat reliable *DataReaders* that send stale NACKs as inactive (in the same manner as *DataReaders* are treated as inactive when not sending any ACKNACKs for a consecutive **heartbeat_max_retries** number of heartbeat periods). In certain scenarios, you may prefer that *DataReaders* sending stale NACKs (NACKs requesting for the same samples) be treated as inactive. For example, suppose a reading application is so slow in taking samples from the reader that the reader's queue is full. Instead of waiting for the application to take the sample (thereby halting progression of reliable communication), if strict reliability is not required, the reader can now be treated as inactive after a configurable number of heartbeat periods has passed without receiving a progressing NACK.

2.6 Performance Improvements

This release added performance improvements in how samples are stored in the *DataReader's* queue when there are many instances and the samples are not removed from the queue. This performance improvement is applied only when there are no matching *DataWriters* using a finite lifespan.

2.7 Java Serializable Support

In this release the types generated by *rtiddsgen* implement the Java Serializable interface. The Java Serializable interface provides a standard way for the application to handle object serialization/deserialization beyond the scope of DDS communications.

It is important to note that the implementation of the Serializable interface does not replace the serialization/deserialization routines that are generated by *rtiddsgen* and used by *RTI Data Distribution Service* to send the samples on the wire.

2.8 Ability to Choose Algorithm for Determining the GUID Prefix

The WireProtocolQosPolicy has a new field called **rtps_auto_id_kind**. It allows you to choose the algorithm used to determine the GUID's 96-bit prefix. There are two possible values: DDS_RTPS_AUTO_ID_FROM_IP (the default, which uses an IPV4-based algorithm) and DDS_RTPS_AUTO_ID_FROM_MAC (which uses a MAC address-based algorithm).

2.9 Dependency on Iphlpapi.lib on Windows Systems

When building C/C++ applications on Windows systems, there is a new additional dependency on **Iphlpapi.lib**.

2.10 Simultaneous Creation of Participants in Multiple Threads is Now Thread Safe

Previously, it was not safe to create one participant while another thread was simultaneously creating another participant. This is now allowed.

2.11 DomainParticipantFactory's get_instance() is Thread Safe on Linux Systems

On Linux platforms, accessing the DomainParticipantFactory is now thread safe.

2.12 Changes in Type Support Interface

The type support interface has been changed to facilitate some new features. All code generated with previous versions of *rtiddsgen* must be regenerated with the version of *rtiddsgen* provided with *RTI Data Distribution Service* 4.5c.

2.13 Ability to Specify Add-on Transport Library Name in Platform-independent Format

Previously, to use add-on transports via DDS_PropertyQosPolicy, you had to specify the library name in a platform-dependent form. For example, to load the transport_wan library, you had to specify **libnddstransportwan.so** on UNIX-like platforms or **nddstransportwan.dll** on Windows platforms.

Now, you can specify **nddstransportwan** and it will load the correct library based on the operating system. This is useful when using XML based QoS configuration.

2.14 Accessing Topic from Entity in C API

The C API has a new function: **Topic_narrow_from_entity()**. As the name implies, it narrows a given DDS_Entity pointer to a DDS_Topic pointer.

2.15 Changes in Default Value for PublishMode QoS Policy

The default value of the PublishModeQosPolicy's **flow_controller_name** field has changed from NULL to DDS_DEFAULT_FLOW_CONTROLLER_NAME.

2.16 New Options in rtiddsspy to Control QoS of Subscriptions

New options have been added to *rtiddsspy* that allow you to specify the history depth of the subscription and to create a subscription based on the reliability and durability QoS of the first discovered publication of a topic.