RTI RTSJ Extension Kit

Release Notes

Version 5.1.0

Your systems. Working as one.

Printed in U.S.A. First printing.
December 2013.

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/

Release Notes

1 Supported Platforms

RTI® RTSJ Extension Kit has been tested on:

- ❏ CentOS 5.4 and 5.5[1] on an x64 CPU (x64Linux2.6gcc4.1.2)
- ❏ Red Hat® Enterprise Linux 5.1, 5.2, 5.4, and 5.5[1] on an x64 CPU (x64Linux2.6gcc4.1.2)

2 Compatibility

RTSJ Extension Kit is built on top of, and intended for use with, *RTI Connext* with the same version number.

3 What's New in 5.1.0

- ❏ This release provides compatibility with *RTI Connext*™ 5.1.0.
- ❏ Support for LynxOS® 5.0 is no longer available

4 Known Issues

This section describes known limitations when using *Connext* with RTSJ.

4.1 Strings Require Memory Allocation

This limitation applies to any application that uses strings within the context of a scoped or immortal memory area.

The OMG standard IDL-to-Java mapping for strings, which *Connext* respects, represents the IDL string type as a Java object of type **java.lang.String**. This type is immutable. Therefore, the middleware must reallocate a String object every time it changes. Any time an application obtains a String from *Connext*—including those embedded in application-specific data types as well as those obtained from *Connext* APIs that use them directly (such as PartitionQosPolicy, Content-

1. Tested on IBM Websphere Real Time 32-bit JVM with *Connext* 32-bit libraries running on a 64-bit kernel

FilteredTopics, etc.)—the application should consider that String to be newly allocated. If the memory area of the thread in which that String is allocated is not garbage-collected, the String's memory will effectively be leaked.

4.2 Resized Sequences Require Memory Allocation

This limitation applies to the reading or taking of any custom data-type that directly or indirectly contains a sequence when the size of that sequence changes from sample to sample.

When *Connext* reads data from the network, it deserializes that data stream into a sample that has already been allocated in the middleware's internal queue. If the data sample contains a sequence (either directly or indirectly), deserializing it will require that sequence to be resized if it contains a different number of elements than it did in the previous sample. This act of resizing involves the orphaning or allocation of objects. If the receive threads have been configured to use a memory area that is not garbage-collected, this memory will not be reclaimed for the life of the corresponding *DomainParticipant*.

4.3 Types Must be Registered in Immortal Memory

This limitation applies to the registration of all application data types.

Because of the way that *Connext* manages the data-type registry, all types must be registered (that is, **<MyType>TypeSupport.register_type** must be called) in immortal memory.

For example:

```
ImmortalMemory.instance().executeInArea(
    new Runnable() {
        public void run() {
            <MyType>TypeSupport.register_type(
                  participant, typeName);
        }
    }
);
```

This restriction applies to the built-in types as well (String, KeyedString, Bytes, and KeyedBytes). By default, these types are registered automatically when you create a *DomainParticipant*. If you do not create your *DomainParticipant* in immortal memory, you must disable automatic registration of these types. You can do so by setting the property **dds.builtin_type.auto_register** to "0" or "false" in the *DomainParticipant's* PropertyQosPolicy.

The String, KeyedString, and KeyedBytes built-in types are of limited utility when using real-time memory areas (see Strings Require Memory Allocation (Section 4.1)). However, you may want to use the Bytes type. In that case, you can register this type manually in immortal memory, just as you would one of your own types. See the online documentation for **BytesTypeSupport** for more information.

4.4 Connext Threads' Memory Areas Must Be Accessible to Entity Creation Thread

This limitation applies to the creation of entities (i.e., calling a **create_*** method) that own threads: *DomainParticipants* and asynchronous *Publishers*.

The **RtsjThreadProperty_t** structure allows you to specify a memory area that will be used by threads created by *Connext*. These threads are started when their owning entity (a *Publisher* in the case of an asynchronous publishing thread, a *DomainParticipant* in the case of all other threads) is created.

As part of the process for spawning these threads, the thread object itself and the **java.lang.Runnable** object that encapsulates the internal *Connext* implementation must be created. These objects are created within the calling thread (the thread that is calling the relevant entity **create_***

method) and in the user-specified memory area. Therefore, the entity creation thread *must* be able to (temporarily) enter the specified memory area.

For example, suppose that a *DomainParticipant* is being created from a thread using heap memory. It is permissible to select immortal memory for the participant's receive threads, because immortal memory can be temporarily entered by a thread running with heap memory. However, it would not be permissible to select scoped memory for any of the participant's threads, since it would be impossible for *Connext* to allocate the necessary objects in that scoped memory area in the context of the calling thread.

4.5 Entity's Memory Area Must Be Accessible to Database Thread

This limitation applies to the destruction of entities (that is, the calling of a **delete_*** method and the subsequent internal finalization).

Because of the way that *Connext* manages the destruction and finalization of entities in a highly concurrent application, a portion of an entity's internal destruction and finalization occurs in the context of the thread that calls the corresponding **delete_*** method, and another portion occurs at a later time in the context of the corresponding participant's database thread. Therefore, the memory area in which a participant's contained entities are allocated must be accessible to the memory area chosen for the database thread.

For example, suppose that a *DomainParticipant* and one of its subscribers were created in scoped memory. The participant's database thread must use the same scoped memory area, or the entities' destruction will fail.