RTI Monitoring Library

Getting Started Guide

Version 5.1.0

Your systems. Working as one.

Technical Support
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089

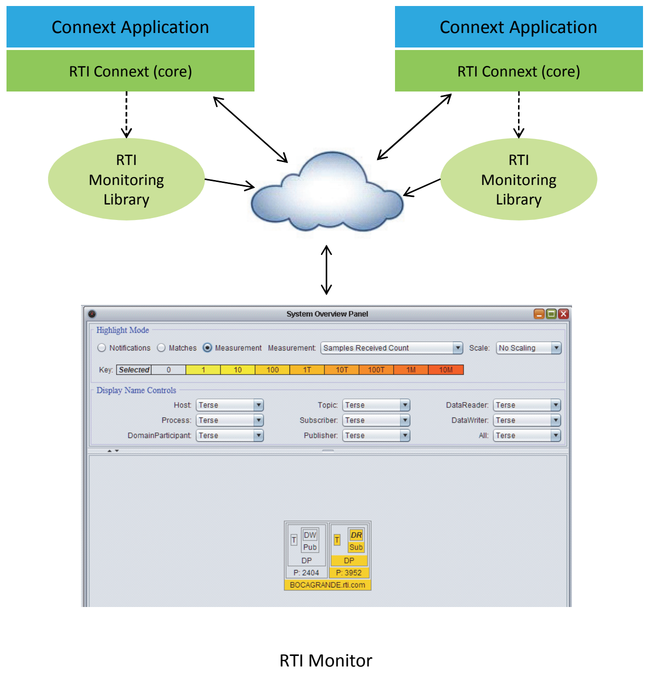
Phone:	(408) 990-7444
Email:	support@rti.com
Website:	https://support.rti.com/

Contents

Chapter 1 Welcome to RTI Monitoring Library

RTI® Monitoring Library is a plug-in that enables *RTI Connext*™ (formerly *RTI Data Distribution Service*) applications to provide monitoring data. The monitoring data can be visualized with *RTI Monitor*, a separate GUI application that can run on the same host as *Monitoring Library* or on a different host.

Connext notifies *Monitoring Library* every time an entity is created/deleted or a QoS is changed. *Monitoring Library* periodically queries the status of all *Connext* entities. You can enable/disable monitoring by setting values in the DomainParticipant's PropertyQosPolicy (programmatically or through an XML QoS profile).



Highlight Mode: ○ Notifications ○ Matches [✓ Measurement]
Measurement: Samples Received Count
Scale: No Scaling

Host:	Terse	Topic:	Terse	DataReader:	Terse
Process:	Terse	Subscriber:	Terse	DataWriter:	Terse
DomainParticipant:	Terse	Publisher:	Terse	All:	Terse

Chapter 2 Installing Monitoring Library

Monitoring Library is included with *RTI Connext Messaging* (formerly *RTI Data Distribution Service, Professional Edition*). Use the installation instructions in this chapter only if you are installing *RTI Monitoring Library* independently (not as part of *Connext Messaging*).

2.1 Instructions for Windows Systems

1. Make sure you have already installed a compatible version of *Connext*. See the *Release Notes* for compatible versions.
2. Extract the contents of the distribution file, **RTI_Monitoring_Library-<*version*>-<*architecture*>.zip**, into the *same* directory where you installed *Connext*.

   For instance, if you have **c:\Program Files\RTI\ndds.<*version*>**, then extract to **c:\Program Files\RTI**.

   You will see a message that the destination already contains a folder named **ndds.<*version*>** and be asked if you want to merge the folder from the .zip file with the existing one. Answer **Yes**. You will also be asked if you want to replace the RTI Software License Agreement file—select **Copy and Replace**.
3. *Optional*: Include the *Connext* and monitoring libraries in your Path. For example:

```
> set NDDSHOME=c:\Program Files\RTI\ndds.<version>
> set Path=%NDDSHOME%\lib\i86Win32VS2005;%Path%
```

4. *Monitoring Library* is used to turn on monitoring in a *Connext* application. Then you can see the monitored data with *Monitor*, a separate application that can run on the same host as *Monitoring Library* or on a different host. If you have not yet installed *Monitor*, you may want to do so now. Refer to the *Monitor* documentation in the *Monitor* bundle for further information. *Monitor* is available from the RTI Support Portal (accessible from https://support.rti.com).

2.2 Instructions for Other Operating Systems

1. Make sure you have already installed a compatible version of *Connext*. See the *Release Notes* for compatible versions.
2. Untar **RTI_Monitoring_Library-<*version*>-<*architecture*>.tar.gz** in the *same* directory as *Connext*.

   For example, if you have **/opt/rti/ndds.<*version*>**, then install in **/opt/rti**:

   ```
   > cd /opt/rti
   > gunzip RTI_Monitoring_Library-<version>-<architecture>.tar.gz
   > gtar xvf RTI_Monitoring_Library-<version>-<architecture>.tar
   ```

   Where <*architecture*> is your architecture, such as i86Linux2.6gcc4.4.5.
3. *Optional*: Include the *Connext* and Monitoring libraries in your LD_LIBRARY_PATH. For example:

   ```
   > setenv NDDSHOME /opt/rti/ndds.<version>
   > setenv LD_LIBRARY_PATH ${NDDSHOME}/lib/<architecture>
   ```
4. *Monitoring Library* is used to turn on monitoring in a *Connext* application. Then you can see the monitored data with *Monitor*, a separate application that can run on the same host as *Monitoring Library* or on a different host. If you have not yet installed *Monitor*, you may want to do so now. Refer to the documentation in the *Monitor* bundle for further information. *Monitor* is available from the RTI Support Portal (accessible from https://support.rti.com).

Chapter 3 Using Monitoring Library in Your Application

3.1 Enabling Monitoring in Your Application

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext* application is linked with the static release version of the *Connext* libraries, you will need to also use the static release version of the monitoring library. Do not link both static *and* dynamic libraries. Similarly, do not mix release *and* debug libraries.

Note: If you are using a non-Windows platform and plan to use *static* libraries, the RTI library from Table 3.0 must appear *first* in the list of libraries to be linked.

Table 3.0 **Required Libraries**

Platform	Static Release	Static Debug	Dynamic Release	Dynamic Debug
AIX®	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so	rtimonitoringd.so
INTEGRITY®	librtimonitoringz.a	librtimonitoringzd.a	*(Not supported)*	
Linux®	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so	rtimonitoringd.so
LynxOS®	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so	rtimonitoringd.so
Mac OS®	librtimonitoringz.a	librtimonitoringzd.a	librtimonitoring.dylib	librtimonitoringd.dylib
QNX®	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so[1]	rtimonitoringd.so[1]
Solaris™	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so	rtimonitoringd.so
VxWorks®	librtimonitoringz.a	librtimonitoringzd.a	rtimonitoring.so[2]	rtimonitoringd.so[2]
Windows®[3]	rtimonitoringz.lib Psapi.lib	rtimonitoringzd.lib Psapi.lib	rtimonitoring.lib rtimonitoring.dll	rtimonitoringd.lib rtimonitoringd.dll

1. To use dynamic libraries, make sure the permissions on the .so library files are readable by everyone.
2. Dynamic Libraries not supported for VxWorks platforms on PPC CPUs using RTP mode.
3. All supported Windows platforms as noted in the *Monitoring Library Release Notes*

There are two ways to enable monitoring in your application:

❑ Method 1—Change the Participant QoS to Automatically Load the Dynamic Monitoring Library (Section 3.1.1)

❑ Method 2—Change the Participant QoS to Specify the Monitoring Library Create Function Pointer and Explicitly Load the Monitoring Library (Section 3.1.2)

Notes:

❑ If your original application has made modifications to either the ParticipantQos resource_limits.type_code_max_serialized_length or any of the transport's default settings to enable large type code or large data, refer to What Monitoring Topics are Published? (Section 3.3) for additional QoS modifications that may be needed.

- ❑ *Monitoring Library* creates internal DataWriters to publish monitoring data by making modifications based on the default DataWriter QoS settings. If you have made changes to the default DataWriter QoS, especially if you have increased/decreased the initial or maximum sample/instance values, Monitoring Library may have trouble creating DataWriters to publish monitoring data, or it may limit the number of statistics that you can publish through the internal monitoring writers. If this is true for your case, you may want to specify the **qos_library** and **qos_profile** that will be used to create these internal writers for publishing monitoring data, to avoid being impacted by default DataWriter QoS settings. See Chapter 4 for details.

3.1.1 Method 1—Change the Participant QoS to Automatically Load the Dynamic Monitoring Library

If all of the following are true, you can enable monitoring simply by changing your participant QoS (otherwise, use Method 2—Change the Participant QoS to Specify the Monitoring Library Create Function Pointer and Explicitly Load the Monitoring Library (Section 3.1.2)):

1. Your application is linked to *dynamic Connext* libraries, or you are using Java or .Net, and
2. You will run your application on a Linux, Windows, Solaris, AIX or Mac OS platform, and
3. You are NOT linking in an additional monitoring library into your application at link time (you let the middleware load the monitoring library for you automatically as needed).

If you change the QoS in an XML file as shown below, you can enable/disable monitoring without recompiling. If you change the QoS in your source code, you may need to recompile every time you enable/disable monitoring.

The easiest way to enable monitoring is setting or inheriting from the built-in profile BuiltinQosLib::Generic.Monitoring.Common. See an example in **<NDDSHOME>/example/QoS/MONITORING_LIBRARY_QOS_PROFILES.xml**.

If you need to change the participant QoS by hand, refer to the definition of **BuiltinQosLib::Generic.Monitoring.Common** in **<NDDSHOME>/resource/qos_profiles_5.x.y/xml/BuiltinProfiles.documentationONLY.xml** for the values you should set.

Example XML to enable monitoring:

```
<participant_qos>
    <property>
        <value>
            <element>
               <name>rti.monitor.library</name>
               <value>rtimonitoring</value>
            </element>

            <element>
                <name>rti.monitor.create_function</name>
                <value>RTIDefaultMonitor_create</value>
            </element>
        </value>
    </property>
</participant_qos>
```

3.1.2 Method 2—Change the Participant QoS to Specify the Monitoring Library Create Function Pointer and Explicitly Load the Monitoring Library

If *any* of the following are true, you must change the Participant QoS to enable monitoring and explicitly load the correct version of *Monitoring Library* at compile time:

- ❑ Your application is linked to the static version of *Connext* libraries.
- ❑ You are NOT running your application on Linux, Windows, Solaris, AIX or Mac OS platforms.
- ❑ You want to explicitly link in the monitoring library (static or dynamic) into your application.

There are two ways to do this:

- ❑ Method 2-A: Change the Participant QoS by Specifying the Monitoring Library Create Function Pointer in Source Code (Section 3.1.2.1): Applies to most users who cannot use Method 1 and do not mind changing/recompiling source code every time you enable/disable monitoring, or whose system does not support setting environment variables programmatically. Participant QoS must be defined in source code with this approach.
- ❑ Method 2-B: Change the Participant QoS by Specifying the Monitoring Library Create Function Pointer in an Environment Variable (Section 3.1.2.2): Applies to users who cannot use Method 1 *and* want to specify the create function pointer via an environment variable. This approach allows the Participant QoS to be defined in an XML file or in source code.

3.1.2.1 Method 2-A: Change the Participant QoS by Specifying the Monitoring Library Create Function Pointer in Source Code

1. Modify your *Connext* application based on the following examples.

C++ Example:

```
#include "ndds/ndds_cpp.h"
#include "monitor/monitor_common.h"
extern "C" int publisher_main(int domainId, int sample_count)
{
    ...
    DDSDomainParticipant *participant = NULL;
    DDS_DomainParticipantQos participant_qos;
    DDSPropertyQosPolicyHelper *qos_policy_helper =
                          new DDSPropertyQosPolicyHelper();
    char valueBuffer[17];

    /*Get default QoS*/
    retcode = DDSTheParticipantFactory->get_default_participant_qos(
                                             participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        /*Error*/
    }

    /* This property indicates that the DomainParticipant has
       monitoring turned on. The property name MUST be
       "rti.monitor.library". The value can be anything.*/
    retcode = qos_policy_helper->add_property(
                          participant_qos.property,
                          "rti.monitor.library",
                          "rtimonitoring", DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        /*Error*/
    }
    /* The property name "rti.monitor.create_function"
       indicates the entry point for the monitoring library.
       The value MUST be the value of the function pointer of
       RTIDefaultMonitor_create */
    sprintf(valueBuffer, "%p", RTIDefaultMonitor_create);
    retcode = qos_policy_helper->add_property(
                             participant_qos.property,
                             "rti.monitor.create_function_ptr",
                             valueBuffer, DDS_BOOLEAN_FALSE);
    if (retcode!= DDS_RETCODE_OK) {
         /* Error */
    }
    /* Create DomainParticipant with participant_qos */
    participant = DDSTheParticipantFactory->create_participant(
                        domainId, participant_qos,
                        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
         /* Error */
    }
    ...
```

C Example:

```
#include "ndds/ndds_c.h"
#include "monitor/monitor_common.h"
...
extern "C" int publisher_main(int domainId, int sample_count)
{
    DDS_DomainParticipantFactory *factory = NULL;
    struct DDS_DomainParticipantQos participantQos =
                        DDS_DomainParticipantQos_INITIALIZER;
    char valueBuffer[17];
    DDS_DomainParticipant *participant = NULL;
    factory = DDS_DomainParticipantFactory_get_instance();
    if (factory == NULL) {
         /* error */
    }
    if (DDS_DomainParticipantFactory_get_default_participant_qos(
                factory, &participantQos) != DDS_RETCODE_OK) {
         /* error */
    }
    /* This property indicates that the DomainParticipant has
       monitoring turned on. The property name MUST be
      “rti.monitor.library”. The value can be anything.*/
    if (DDS_PropertyQosPolicyHelper_add_property(
                &participantQos.property,
                "rti.monitor.library", "rtimonitoring",
                DDS_BOOLEAN_FALSE) != DDS_RETCODE_OK) {
       /* error */
    }
    /* The property name "rti.monitor.create_function_ptr"
       indicates the entry point for the monitoring library.
       The value MUST be the value of the function pointer of
       RTIDefaultMonitor_create */
```

```
sprintf(valueBuffer, "%p", RTIDefaultMonitor_create);
if (DDS_PropertyQosPolicyHelper_add_property(
            &participantQos.property,
            "rti.monitor.create_function_ptr", valueBuffer,
            DDS_BOOLEAN_FALSE) != DDS_RETCODE_OK) {
    /* error */
}
/* create DomainParticipant with participantQos here */
participant= DDS_DomainParticipantFactory_create_participant(
            factory, domainId, &participantQos,
            NULL /* listener */, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    /* error */
}
DDS_DomainParticipantQos_finalize(&participantQos);
...
```

Note: In the above code, you may notice that valueBuffer is initialized to 17 characters. This is because a pointer (RTIDefaultMonitor_create) is at most 8 bytes (on a 64-bit system) and it takes two characters to represent a byte in hex. So the total size must be:

(2 * 8 characters) + 1 null-termination character = 17 characters.

2. Link the *Monitoring Library* for your platform into your application at compile time (see Table 3.0 on page 3-1).

   The kind of monitoring library that you link into your application at compile time must be consistent with the kind of *Connext* libraries that you are linking into your application (static/dynamic, release/debug version of the libraries).

   On Windows systems: As noted in Table 3.0 on page 3-1, if you are linking a static monitoring library, you will also need to link in **Psapi.lib** at compile time.

3.1.2.2 Method 2-B: Change the Participant QoS by Specifying the Monitoring Library Create Function Pointer in an Environment Variable

This is similar to Method 2-A, but if you specify the function pointer value for **rti.monitor.create_function_ptr** in an environment variable that is set programmatically, you can specify your QoS either in an XML file or in source code. If you specify the QoS in an XML file, you can enable/disable monitoring without recompiling. If you change the QoS in your source code, you may need to recompile every time you enable/disable monitoring.

1. In XML, enable monitoring by setting the **rti.monitor.create_function_ptr** property to an environment variable. In our example, the variable is named RTIMONITORFUNCPTR.

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>rti.monitor.library</name>
                <value>rtimonitoring</value>
            </element>
            <element>
                <name>rti.monitor.create_function_ptr</name>
                <value>$(RTIMONITORFUNCPTR)</value>
            </element>
        </value>
    </property>
</participant_qos>
```

2. In the DDS application that links in the monitoring library, get the function pointer of **RTIDefaultMonitor_create** and write it to the same environment variable you named in Step 1 and create a DomainParticipant by using the XML profile specified in Step 1. (Setting of the environment variable must appear in the application *before* it creates the *DomainParticipant* using the profile from Step 1.)

   Here is an example in C:

   ```
   #include <stdio.h>
   #include <stdlib.h>
   #include "monitor/monitor_common.h"
   ...
   char putenvBuffer[34];
   int putenvReturn;
   putenvBuffer[0] = '\0';
   sprintf(putenvBuffer, "RTIMONITORFUNCPTR=%p",
            RTIDefaultMonitor_create);
   putenvReturn = putenv(putenvBuffer);
   if (putenvReturn) {
      printf(
         "Error: couldn't set env variable for RTIMONITORFUNCPTR. "
         "error code: %d\n",  putenvReturn );
   }
   ...
   /* create DomainParticipant using XML profile from Step 1 */
   ...
   ```

   Note: In the above code, you may notice that **putenvBuffer** is initialized to 34 characters. This is because a pointer (RTIDefaultMonitor_create) is at most 8 bytes (on a 64-bit system) and it takes 2 characters to represent a byte in hex. So the total size must be: strlen(RTIMONITORFUNCPTR) + (2 * 8 characters) + 1 null-termination character = 17 + 16 + 1 = 34 characters

3. Link the *Monitoring Library* for your platform into your application at compile time (see Table 3.0).

   The kind of monitoring library that you link into your application at compile time must be consistent with the kind of *Connext* libraries that you are linking into your application (static/dynamic, release/debug version of the libraries).

   On Windows systems: As noted in Table 3.0, if you are linking a static monitoring library, you will also need to link in **Psapi.lib** at compile time.

3.2 How does Monitoring Library Work?

Monitoring Library works by creating DDS Topics that publish information about the other DDS entities contained in the same operating system process. The Topics can be created inside of the first DomainParticipant that enables the library (the default). Or they may be created in a separate DomainParticipant if the **rti.monitor.config.new_participant_domain_id** property is used. Use cases for this latter configuration include controlling the domain ID on which this information is exchanged (for example to ensure that this data does not interfere with production topics) as well as the ability to specify the QoS that is used for the DomainParticipant (through the **rti.monitor.config.qos_library** and **rti.monitor.config.qos_profile** properties). It may be desirable to specify the QoS for *RTI Distributed Logger's* DomainParticipant if the information will be consumed on a different transport or simply to enable the feature but keep it as isolated from the production system as possible.

3.3 What Monitoring Topics are Published?

Two categories of predefined monitoring topics are sent out:

- ❏ *Descriptions* are published when an entity is created or deleted, or there are QoS changes (see Table 3.1).
- ❏ *Entity Statistics* are published periodically (see Table 3.2).

Table 3.1 **Descriptions (QoS and Other Static System Information)**

Topic Name	Topic Contents
rti/dds/monitoring/domainParticipantDescription	DomainParticipant QoS and other static information
rti/dds/monitoring/topicDescription	Topic QoS and other static information
rti/dds/monitoring/publisherDescription	Publisher QoS and other static information
rti/dds/monitoring/subscriberDescription	Subscriber QoS and other static information
rti/dds/monitoring/dataReaderDescription	DataReader QoS and other static information
rti/dds/monitoring/dataWriterDescription	DataWriter QoS and other static information

Table 3.2 **Entity Statistics (Statuses, Aggregated Statuses, CPU and Memory Usage)**

Topic Name	Topic Contents
rti/dds/monitoring/domainParticipantEntityStatistics	Number of entities discovered in the system, CPU and memory usage of the process
rti/dds/monitoring/dataReaderEntityStatistics	DataReader statuses
rti/dds/monitoring/dataWriterEntityStatistics	DataWriter statuses
rti/dds/monitoring/topicEntityStatistics	Topic statuses
rti/dds/monitoring/ dataReaderEntityMatchedPublicationStatistics	DataReader statuses calculated on a per discovered matching writer basis
rti/dds/monitoring/ dataWriterEntityMatchedSubscriptionStatistics	DataWriter statuses calculated on a per discovered matching reader basis
rti/dds/monitoring/ dataWriterEntityMatchedSubscriptionWithLocatorStatistics	DataWriter statuses calculated on a per sending destination basis

All monitoring data are sent out using specially created DataWriters with the above topics.

You can configure some aspects of *Monitoring Library's* behavior, such as which monitoring topics to turn on, which user topics to monitor, how often to publish the statistics topics, and whether to publish monitoring data using (a) the participant created in the user's application that has monitoring turned on or (b) a separate participant created just for publishing monitoring data. See Chapter 4: Configuring Monitoring Library.

3.4 Enabling Support for Large Type-Code and Large Data (Optional)

Some monitoring topics have large type-code (larger than the default maximum type code serialized size setting). If you use *Monitor* to display all the monitoring data, it already has all the monitoring types built-in and therefore it uses the default maximum type-code serialized size in the *Connext* application and there is no problem. However, if you are using any other tools to display monitoring data (such as *RTI Spreadsheet Add-in for Microsoft Excel*, *rtiddsspy*, or writing your own application to subscribe to monitoring data), or if your user data-type has large type-code, you may need to increase the maximum type-code serialized size setting.

The description monitoring topics can potentially have large data sizes (larger than what the default transport settings can handle). By default, an asynchronous publisher is used in all the description topics in *Monitoring Library* to resolve this large-data issue. However, if your *Connext* application has a need to use large data (for example, due to large data in a user-defined data type), you may need to change the default QoS configuration to add support for large data in all transports.

If you use the default values for maximum type-code serialized size and transport settings, everything will work fine out of the box. However, if your original application has made changes to either type-code serialized size or transport settings, you will need to make sure that BOTH settings are changed in a consistent manner. If you are using *Monitor* to display the data, those changes will also need to be made in *Monitor*.

The following sample ParticipantQos configuration can be used to configure support for large type-code and large data usage for UDPv4 and shared-memory transports.

This participant configuration can be used either by your application's participant or in a new participant created just for publishing monitoring topics, depending on your monitoring library configuration (see "new_participant_domain_id" on page 4-2.).

To see a sample QoS profile containing these transport configurations, open **<NDDSHOME>/resource/monitor/xml/MONITORING_QOS_PROFILES.xml**. and look for the QoS library, **RTIMonitoringQosLibrary**, and QoS profile, **RTIMonitoringPublishingLargeDataQosProfile**.

```
<!-- ======================================= -->
<!-- Transport Configurations for Large Data -->
<!-- ======================================= -->
<participant_qos>
  <property>
   <value>
   <!-- UDPv4 -->
   <element>
     <name>dds.transport.UDPv4.builtin.parent.message_size_max</name>
     <value>65530</value>
     <propagate>false</propagate>
   </element>
   <element>
     <name>dds.transport.UDPv4.builtin.recv_socket_buffer_size</name>
     <value>65530</value>
     <propagate>false</propagate>
   </element>
   <element>
     <name>dds.transport.UDPv4.builtin.send_socket_buffer_size</name>
     <value>65530</value>
     <propagate>false</propagate>
   </element>
```

```
    <!-- Shared memory -->
    <element>
      <name>dds.transport.shmem.builtin.parent.message_size_max</name>
      <value>65530</value>
      <propagate>false</propagate>
    </element>
    <element>
      <name>dds.transport.shmem.builtin.receive_buffer_size</name>
      <value>65530</value>
      <propagate>false</propagate>
    </element>
    <element>
      <name>dds.transport.shmem.builtin.received_message_count_max</name>
      <value>32</value>
      <propagate>false</propagate>
    </element>
   </value>
   </property>
   <!-- monitoring types have large type code -->
   <resource_limits>
     <type_code_max_serialized_length>
          30000
     </type_code_max_serialized_length>
   </resource_limits>
   <!-- monitoring types can have large data -->
   <receiver_pool>
     <buffer_size>65530</buffer_size>
   </receiver_pool>
 </participant_qos>
```

Chapter 4 Configuring Monitoring Library

You can control some aspects of *Monitoring Library's* behavior by setting the PropertyQosPolicy of the DomainParticipant, either via an XML QoS profile or in your application's code.

Sample QoS profiles are provided in
<NDDSHOME>/example/QoS/MONITORING_LIBRARY_QOS_PROFILES.xml.

There are two QoS profiles in **MONITORING_LIBRARY_QOS_PROFILES.xml** file:

❑ **CustomerExampleMonitoringLibrary::CustomerExampleMonitoringProfile**

This is an example of how to enable *Monitoring Library* for your applications. It can be used as a guide to enabling *Monitoring Library* quickly in your applications.

❑ **RTIMonitoringQosLibrary::RTIMonitoringQosProfile**

This profile documents the QoS used by *Monitoring Library.* It can also be used as a starting point if you want to tune QoS for *Monitoring Library* (normally not necessary). Use cases for this include customizing DomainParticipant QoS (often the transports) to accommodate preferences or environment. This same profile can also be used to subscribe to the *Monitoring Library* Topics. This is useful in situations where the *Monitoring Library* information can be used directly by system components or it is not possible to use the *RTI Monitor* tool.

See the qos_library and qos_profile properties in Table 4.1 for further information on when to use the example profiles in **MONITORING_LIBRARY_QOS_PROFILES.xml**.

Table 4.1 lists the configuration properties that you can set for *Monitoring Library.*

Table 4.1 **Configuration Properties for Monitoring Library**

Property Name (all must be prepended with "rti.monitor.config.")	**Property Value**
get_process_statistics	This boolean value specifies whether or not *Monitoring Library* should collect CPU and memory usage statistics for the process in the topic **rti/dds/monitoring/domainParticipantDescription**. This property is only applicable to Linux and Windows systems—obtaining CPU and memory usage on other architectures is not supported. CPU usage is reported in terms of time spent since the process has been started. It can be longer than the actual running time of the process on a multi-core machine. Default: true if unspecified

Table 4.1 **Configuration Properties for Monitoring Library**

Property Name (all must be prepended with "rti.monitor.config.")	Property Value
new_participant_domain_id	To create a separate participant that will be used to publish monitoring information in the application, set this to the domain ID that you want to use for the newly created participant. This property can be used with the qos_library and qos_profile properties to specify the QoS that will be used to create a new participant. Default: Not set (means you want to reuse the participant in your application that has monitoring turned on to publish statistics information for that participant)
publish_period	Period of time to sample and publish all monitoring topics, in units of seconds. Default: 5 if unspecified
publish_thread_priority	Priority of the thread used to sample and publish monitoring data. This value is architecture dependent. Default if unspecified: same as the default used in *Connext* for the event thread: • Windows systems: -2 • Linux systems: -999999 (meaning use OS-default priority)
publish_thread_stacksize	Stack size used for the thread that samples and publishes monitoring data. This value is architecture dependent. Default if unspecified: same as the default used in *Connext* for the event thread: • Windows systems: 0 (meaning use the default size for the executable). • Linux systems: -1 (meaning use OS's default value).
publish_thread_options	Describes the type of thread. Supported values (may be combined with by OR'ing with '|' as seen in the default below): • **FLOATING_POINT**: Code executed within the thread may perform floating point operations • **STDIO**: Code executed within the thread may access standard • **I/O REALTIME_PRIORITY**: The thread will be scheduled on a real-time basis • **PRIORITY_ENFORCE**: Strictly enforce this thread's priority Default: **FLOATING_POINT | STDIO** (same as the default used in *Connext* for the event thread)
qos_library	Specifies the name of the QoS library that you want to use for creating entities in the monitoring library (if you do not want to use default QoS values as set by the monitoring library). The QoS values used for internally created entities can be found in the library **RTIMonitoringQosLibrary** in **<NDDSHOME>/example/QoS/MONITORING_LIBRARY_QOS_PROFILES.xml**. Default: Not set (means you want to use default *Monitoring Library* QoS values)

Table 4.1 **Configuration Properties for Monitoring Library**

Property Name (all must be prepended with "rti.monitor.config.")	**Property Value**
qos_profile	Specifies the name of the QoS profile that you want to use for creating entities in the monitoring library (if you do not want to use the default QoS values). The QoS values used for internally created entities can be found in the profile **RTIMonitoringPublishingQosProfile** in **<NDDSHOME>/example/QoS/ MONITORING_LIBRARY_QOS_PROFILES.xm**l. Default: Not set (means you want to use default *Monitoring Library* QoS values)
reset_status_change_counts	*Monitoring Library* obtains all statuses of all entities in the *Connext* application. This boolean value controls whether or not the change counts in those statuses are reset by *Monitoring Library.* If set to true, the change counts are reset each time *Monitoring Library* is done accessing them. If set to false, the change counts truly reflect what users will see in their application and are unaffected by the access of the monitoring library. Default: false
skip_monitor_entities	This boolean value controls whether or not the entities created internally by *Monitoring Library* should be included in the entity counts published by the participant entity statistics topic. If set to true, the internal monitoring entities will not be included in the count. (Thirteen internal writers are created by the monitoring library by default.) Default: true
skip_participant_properties	If set to true, DomainParticipant PropertyQosPolicy name and value pairs will not be sent out through the domainParticipantDescriptionTopic. This is necessary if you are linking with *Monitoring Library* and any of these conditions occur: • The PropertyQosPolicy of a DomainParticipant has more than 32 properties. • Any of the properties in PropertyQosPolicy of a DomainParticipant has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified
skip_reader_properties	If set to true, DataReader PropertyQosPolicy name and value pairs will not be sent out through the dataReaderDescriptionTopic. This is necessary if you are linking with *Monitoring Library* and any of these conditions occur: • The PropertyQosPolicy of a DataReader has more than 32 properties. • Any of the properties in PropertyQosPolicy of a DataReader has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified

Table 4.1 **Configuration Properties for Monitoring Library**

Property Name (all must be prepended with "rti.monitor.config.")	**Property Value**
skip_writer_properties	If set to true, DataWriter PropertyQosPolicy name and value pairs will not be sent out through the dataWriterDescriptionTopic. This is necessary if you are linking with *Monitoring Library* and any of these conditions occur: • The PropertyQosPolicy of a DataWriter has more than 32 properties. • Any of the properties in PropertyQosPolicy of a DataWriter has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified
topics	Filter for monitoring topics, with regular expression matching syntax as specified in the *Connext* documentation (similar to the POSIX fnmatch syntax). For example, if you only want to send description topics and the entity statistics topics, but NOT the matching statistics topics, you can specify **"*Description,*EntityStatistics"**. Default: * if unspecified
usertopics	Filter for user topics, with regular expression matching syntax as specified in the *Connext* documentation (similar to the POSIX fnmatch syntax). For example, if you only want to send monitoring information for reader/writer/topic entities for topics that start with Foo or Bar, you can specify **"Foo*,Bar*"**. Default: * if unspecified
verbosity	Sets the verbosity on the monitoring library for debugging purposes (does not affect the topic/data that is sent out). • -1: Silent • 0: Exceptions only • 1: Warnings • 2 and up: Higher verbosity level Default: 1 if unspecified
writer_pool_buffer_max_size	Controls the threshold at which dynamic memory allocation is used, expressed as a number of bytes. If the serialized size of the data to be sent is smaller than this size, a pre-allocated writer buffer pool is used to obtain the memory. If the serialized size of the data is larger than this value, the memory is allocated dynamically. This setting can be used to control memory consumption of the monitoring library, at the cost of performance, when the maximum serialized size of the data type is large (which is the case for some description topics' data types) or if you have several participants on the same machine. The default setting is -1, meaning memory is always obtained from the writer buffer pool, whose size is determined by the maximum serialized size.

Chapter 5 Troubleshooting

5.1 Buffer Allocation Error

Monitoring Library obtains the default DataWriter QoS from the *Connext* application's DomainParticipant. If the application has changed the default QoS Profile, either through application code or in an XML file, *Monitoring Library* will use this new default QoS. In specific scenarios, the new default QoS may cause your *Connext* application to run out of memory and report error messages similar to these:

```
REDAFastBufferPool_growEmptyPoolEA: !allocate buffer of 1210632000 bytes
[D0012|ENABLE]REDAFastBufferPool_newWithNotification:!create fast buffer pool
buffers
[D0012|ENABLE]PRESTypePluginDefaultEndpointData_createWriterPool:!create
writer buffer pool
[D0012|ENABLE]WriterHistorySessionManager_new:!create newAllocator
[D0012|ENABLE]WriterHistoryMemoryPlugin_createHistory:!create sessionManager
[D0012|ENABLE]PRESWriterHistoryDriver_new:!create _whHnd
[D0012|ENABLE]PRESPsService_enableLocalEndpointWithCursor:!create WriterHisto-
ryDriver
[D0012|ENABLE]PRESPsService_enableAllLocalEndpointsInGroupWithCursor:!enable
endpoint
[D0012|ENABLE]PRESPsService_enableGroupWithCursor:!enableAllLocalEndpointsIn-
GroupWithCursor
[D0012|ENABLE]PRESPsService_enableGroup:!enableGroupWithCursor
[D0012|ENABLE]RTIDefaultMonitorPublisher_enableEntitiesAndStartThreadI:!cre-
ate enable publisher
[D0012|ENABLE]RTIDefaultMonitorPublisher_onEventNotify:!create enable entities
```

To resolve this problem, either:

- ❏ Configure *Monitoring Library* to use a non-default QoS Profile. For details, please see Chapter 4: Configuring Monitoring Library
- ❏ Change the default QoS to have a lower value for DataWriter's **initial_samples**; this field is part of the ResourceLimitsQosPolicy.

[RTI Bug # 13771]