Contents

1 Welcome to RTI Monitoring Library ................................................................. 1-1

2 Installing RTI Monitoring Library ................................................................. 2-1
   2.1 Instructions for Windows Systems ............................................................. 2-1
   2.2 Instructions for Other Operating Systems ............................................... 2-2

3 Using RTI Monitoring Library in Your Application ................................. 3-1
   3.1 Enabling Monitoring in Your Application.................................................. 3-1
      3.1.1 Method 1—Change the Participant QoS and Automatically Link the Library .... 3-2
      3.1.2 Method 2—Change Your Source Code and Explicitly Link the Library ......... 3-3
   3.2 What Monitoring Topics are Published? ..................................................... 3-5
   3.3 Enabling Support for Large Type-Code and Large Data (Optional) ............... 3-7

4 Configuring RTI Monitoring Library ............................................................ 4-1
Chapter 1  Welcome to RTI Monitoring Library

RTI Monitoring Library is a plug-in that enables DDS 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 RTI Monitoring Library or on a different host.

RTI Data Distribution Service notifies RTI Monitoring Library every time an entity is created/deleted or a QoS is changed. RTI Monitoring Library periodically queries the status of all DDS entities.

You can enable/disable monitoring by setting values in the DomainParticipant’s PropertyQosPolicy (programmatically or through an XML QoS profile).
Chapter 2 Installing RTI Monitoring Library

2.1 Instructions for Windows Systems

1. Make sure you have already installed a compatible version of RTI Data Distribution Service. 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 RTI Data Distribution Service.

   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.4.5x 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 DDS and monitoring libraries in your Path. For example:

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

4. RTI Monitoring Library is used to turn on monitoring in a DDS application. Then you can see the monitored data with RTI Monitor, a separate application that can run on the same host as RTI Monitoring Library or on a different host. If you have not yet installed RTI Monitor, you may want to do so now. Refer to the RTI Monitor documentation in the RTI Monitor bundle for further information. RTI Monitor is available from the RTI Customer 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 RTI Data Distribution Service. See the Release Notes for compatible versions.

2. Untar RTI_Monitoring_Library-<version>-<architecture>.tar.gz in the same directory as RTI Data Distribution Service.
   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 i86Linux2.6gcc4.1.1.

3. Optional: Include the DDS and monitoring libraries in your LD_LIBRARY_PATH. For example:
   
   ```
   > setenv NDDSHOME /opt/rti/ndds.<version>
   > setenv LD_LIBRARY_PATH ${NDDSHOME}/lib/<architecture>
   ```

4. RTI Monitoring Library is used to turn on monitoring in a DDS application. Then you can see the monitored data with RTI Monitor, a separate application that can run on the same host as RTI Monitoring Library or on a different host. If you have not yet installed RTI Monitor, you may want to do so now. Refer to the RTI Monitor documentation in the RTI Monitor bundle for further information. RTI Monitor is available from the RTI Customer Portal (accessible from https://support.rti.com).
Chapter 3  Using RTI 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—if your DDS application is linked with the static release version of the DDS 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.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Static Release</th>
<th>Static Debug</th>
<th>Dynamic Release</th>
<th>Dynamic Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>rtimonitoring.so</td>
<td>rtimonitoringd.so</td>
</tr>
<tr>
<td>INTEGRITY®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>(Not supported)</td>
<td></td>
</tr>
<tr>
<td>Linux®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>rtimonitoring.so</td>
<td>rtimonitoringd.so</td>
</tr>
<tr>
<td>LynxOS®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>rtimonitoring.so</td>
<td>rtimonitoringd.so</td>
</tr>
<tr>
<td>Mac OS®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>librtimonitoring.dylib</td>
<td>librtimonitoringd.dylib</td>
</tr>
<tr>
<td>Solaris™</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>rtimonitoring.so</td>
<td>rtimonitoringd.so</td>
</tr>
<tr>
<td>VxWorks®</td>
<td>librtimonitoringz.a</td>
<td>librtimonitoringzd.a</td>
<td>rtimonitoring.so¹</td>
<td>rtimonitoringd.so¹</td>
</tr>
<tr>
<td>Windows®²</td>
<td>rtimonitoringz.lib</td>
<td>rtimonitoringzd.lib</td>
<td>rtimonitoring.lib</td>
<td>rtimonitoringd.lib</td>
</tr>
<tr>
<td>Windows CE</td>
<td>rtimonitoringz.lib</td>
<td>rtimonitoringzd.lib</td>
<td>rtimonitoring.lib</td>
<td>rtimonitoringd.lib</td>
</tr>
</tbody>
</table>

1. Dynamic Libraries not supported for VxWorks platforms on PPC CPUs using RTP mode.
2. All supported Windows platforms as noted in the RTI Monitoring Library Release Notes (except Windows CE, which is described in a separate row in this table)
3.1.1 Method 1—Change the Participant QoS and Automatically Link the Library

If **all of the following are true**, you can enable monitoring simply by changing your participant QoS (otherwise, use Method 2—Change Your Source Code and Explicitly Link the Library (Section 3.1.2)):

1. Your application is linked to *dynamic* DDS 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.

**Example XML to enable monitoring**:

```xml
<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>
```

**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 Section 3.3 for additional QoS modifications that may be needed.
- *RTI 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, the RTI 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.

At run time, the dynamic DDS libraries and the RTI Monitoring Library must all be in your Path (for Windows systems), LD_LIBRARY_PATH (for Linux systems), or DYLD_LIBRARY_PATH (for Mac OS systems). The dynamic Monitoring library will be loaded automatically when the first participant that has monitoring turned on is created.

### 3.1.2 Method 2—Change Your Source Code and Explicitly Link the Library

If any of the following are true, you must change your source code to enable monitoring and explicitly link in the correct version of the monitoring library at compile time:

- Your application is linked to static version of DDS 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.

1. Modify your DDS application based on the following example C code:

```c
#include "n.dds/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, 0 /* domainId */ , &participantQos,
    NULL /* listener */ , DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    /* error */
}
DDS_DomainParticipantQos_finalize(&participantQos);
...

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 Section 3.3 for additional QoS modifications that may be needed.

- RTI 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, RTI 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.

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 DDS libraries that you are linking into your application (static/dynamic, release/debug version of the libraries).

On Windows systems (except Windows CE): 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. If you are linking to a dynamic version of the monitoring library, that library must be in your Path (for Windows systems), LD_LIBRARY_PATH (for Linux systems), or DYLD_LIBRARY_PATH (for Mac OS systems) at run time.

### 3.2 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)

<table>
<thead>
<tr>
<th>Topic Name</th>
<th>Topic Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti/dds/monitoring/domainParticipantDescription</td>
<td>DomainParticipant QoS and other static information</td>
</tr>
<tr>
<td>rti/dds/monitoring/topicDescription</td>
<td>Topic QoS and other static information</td>
</tr>
<tr>
<td>rti/dds/monitoring/publisherDescription</td>
<td>Publisher QoS and other static information</td>
</tr>
</tbody>
</table>
All monitoring data are sent out using specially created DDS DataWriters with the above topics.

You can configure some aspects of RTI 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 RTI Monitoring Library.

<table>
<thead>
<tr>
<th>Table 3.1</th>
<th>Descriptions (QoS and Other Static System Information)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Name</td>
<td>Topic Contents</td>
</tr>
<tr>
<td>rti/dds/monitoring/subscriberDescription</td>
<td>Subscriber QoS and other static information</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataReaderDescription</td>
<td>DataReader QoS and other static information</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataWriterDescription</td>
<td>DataWriter QoS and other static information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.2</th>
<th>Entity Statistics (Statuses, Aggregated Statuses, CPU and Memory Usage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Name</td>
<td>Topic Contents</td>
</tr>
<tr>
<td>rti/dds/monitoring/domainParticipantEntityStatistics</td>
<td>Number of entities discovered in the system, CPU and memory usage of the process</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataReaderEntityStatistics</td>
<td>DataReader statuses</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataWriterEntityStatistics</td>
<td>DataWriter statuses</td>
</tr>
<tr>
<td>rti/dds/monitoring/topicEntityStatistics</td>
<td>Topic statuses</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataReaderEntityMatchedPublicationStatistics</td>
<td>DataReader statuses calculated on a per discovered matching writer basis</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataWriterEntityMatchedSubscriptionStatistics</td>
<td>DataWriter statuses calculated on a per discovered matching reader basis</td>
</tr>
<tr>
<td>rti/dds/monitoring/dataWriterEntityMatchedSubscriptionWithLocatorStatistics</td>
<td>DataWriter statuses calculated on a per sending destination basis</td>
</tr>
</tbody>
</table>
3.3 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 RTI 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 DDS 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, rtiddspy, 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 RTI Monitoring Library to resolve this large-data issue. However, if your DDS 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 RTI Monitor to display the data, those changes will also need to be made in RTI 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, RTIMonitoring-PublishingLargeDataQosProfile.

<!-- Transport Configurations for Large Data -->

<participant_qos>
  <property>
    <value>
<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>

<!-- 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 RTI Monitoring Library

You can control some aspects of RTI Monitoring Library’s behavior by setting the Property QosPolicy of the DomainParticipant, either via an XML QoS profile or in your application’s code. Sample QoS profiles are provided in `<NDDSHOME>/resource/monitor/xml/MONITORING_QOS_PROFILES.xml`. See qos_library and qos_profile properties in Table 4.1 for further information on when to use the example profiles in MONITORING_QOS_PROFILES.xml.

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

Table 4.1  Configuration Properties for RTI Monitoring Library

<table>
<thead>
<tr>
<th>Property Name (all must be prepended with “rti.monitor.config.”)</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_process_statistics</td>
<td>This boolean value specifies whether or not RTI Monitoring Library should collect CPU and memory usage statistics for the process in the topic <code>rti/dds/monitoring/domainParticipantDescription</code>. 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</td>
</tr>
</tbody>
</table>
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 DDS 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 DDS for the event thread:
- Windows systems: 0 (meaning use the default size for the executable).
- Linux systems: -1 (meaning use OS’s default value).
### Table 4.1 Configuration Properties for RTI Monitoring Library

<table>
<thead>
<tr>
<th>Property Name (all must be prepended with “rti.monitor.config.”)</th>
<th>Property Value</th>
</tr>
</thead>
</table>
| 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 DDS 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>/resource/monitor/xml/MONITORING_QOS_PROFILES.xml`. Default: Not set (means you want to use default RTI Monitoring Library QoS values) |
| 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>/resource/monitor/xml/MONITORING_QOS_PROFILES.xml`. Default: Not set (means use default RTI Monitoring Library QoS values). |
Table 4.1  Configuration Properties for RTI Monitoring Library

<table>
<thead>
<tr>
<th>Property Name (all must be prepended with “rti.monitor.config.”)</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset_status_change_counts</td>
<td>RTI Monitoring Library obtains all statuses of all entities in the DDS application. This boolean value controls whether or not the change counts in those statuses are reset by RTI Monitoring Library. If set to true, the change counts are reset each time RTI 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</td>
</tr>
<tr>
<td>skip_monitor_entities</td>
<td>This boolean value controls whether or not the entities created internally by RTI 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 readers are created by the monitoring library by default.) Default: true</td>
</tr>
<tr>
<td>skip_participant_properties</td>
<td>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 RTI Monitoring Library and the PropertyQosPolicy of a DomainParticipant has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified</td>
</tr>
<tr>
<td>skip_reader_properties</td>
<td>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 RTI Monitoring Library and the PropertyQosPolicy of a DataReader has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified</td>
</tr>
<tr>
<td>skip_writer_properties</td>
<td>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 RTI Monitoring Library and the PropertyQosPolicy of a DataWriter has a name longer than 127 characters or a value longer than 511 characters. Default: false if unspecified</td>
</tr>
</tbody>
</table>
### Table 4.1 Configuration Properties for RTI Monitoring Library

<table>
<thead>
<tr>
<th>Property Name (all must be prepended with “rti.monitor.config.”)</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>Filter for monitoring topics, with regular expression matching syntax as specified in the DDS 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</td>
</tr>
<tr>
<td>usertopics</td>
<td>Filter for user topics, with regular expression matching syntax as specified in the DDS 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</td>
</tr>
</tbody>
</table>
| 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.  
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).  
The default setting is -1, meaning memory is always obtained from the writer buffer pool, whose size is determined by the maximum serialized size. |