RTI Distributed Logger

Getting Started Guide

Version 5.1.0

Your systems. Working as one.

Trademarks

Copy and Use Restrictions

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/

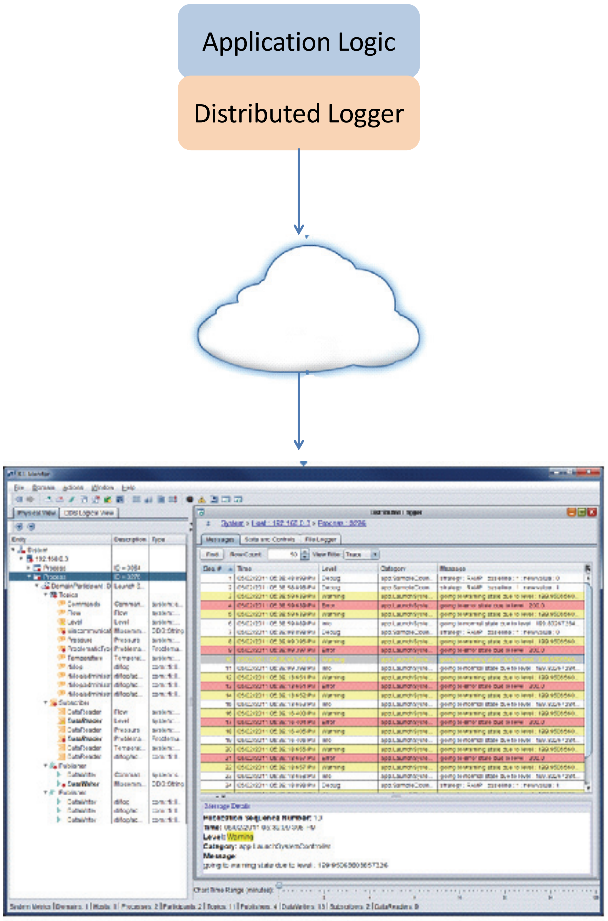
Contents

Chapter 1 Welcome to RTI Distributed Logger

RTI® Distributed Logger is a library that enables applications to publish their log messages to *Connext*.

The log message data can be visualized with *RTI Monitor*, a separate GUI application that can run on the same host as your application or on a different host. Since the data is provided in a Topic, you can also use *rtiddsspy* or even write your own visualization tool.

Distributed Logger can send *Connext* errors, warnings and other internal messages as a DDS topic. In fact, *Distributed Logger* also provides a remote command topic so that its behavior can be remotely controlled at run time.



Distributed Logger Panel in RTI Monitor

Chapter 2 Installing Distributed Logger

RTI Distributed Logger is automatically installed as part of *RTI Connext™ Professional*. Use the installation instructions in this chapter only if you are installing *RTI Distributed Logger* independently (not as part of *RTI Connext Professional).*

1. Make sure you have already installed a compatible version of *Connext* (see the *Distributed Logger Release Notes* for compatible versions).
2. Extract the contents of the distribution file, **RTI_DistributedLogger.<*version*>-<*architecture*>.[zip | tar.gz]**, into the same directory where you installed *Connext*. For instance, if you have **C:\Program Files\RTI\ndds.<*version*>**, extract to **C:\Program Files\RTI**. This will install the libraries in **C:\Program Files\RTI\ndds.<*version*>\lib\<*architecture*>**.
3. Optional: Install *RTI Monitor.*

   Distributed Logger publishes log messages to *Connext*. Once you have installed the software and integrated *Distributed Logger* into your application (described in Chapter 3), you can see the messages in *Monitor*, a separate application that can run on the same host as *Distributed Logger* or on a different host. If you have not yet installed *Monitor*, you may want to do so now. You will need to use a compatible version of *Monitor* (see the *Distributed Logger Release Notes*). 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/).

❏ **If you are using Visual Studio 2005:**

You must have the Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package MFC Security Update installed on the machine where you are *running* an application built with the release or debug libraries of the following RTI architecture packages:

- i86Win32VS2005 and x64Win64VS2005, built with dynamic libraries
- i86Win32jdk and x64Win64jdk
- i86Win32dotnet2.0 and x64Win64dotnet2.0

The Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package MFC Security Update can be obtained from the following Microsoft website:

- http://www.microsoft.com/download/en/details.aspx?id=26347

❑ **If you are using Visual Studio 2008:**

You must have Visual Studio 2008 Service Pack 1 or the Microsoft Visual C++ 2008 SP1 Redistribution Package installed on the machine where you are *running* an application built with the following RTI architecture packages:

- x64Win64VS2008 built with dynamic libraries
- i86Win32VS2008 built with dynamic libraries

The Microsoft Visual C++ 2008 SP1 Redistribution Package can be downloaded from the following Microsoft website:

- For x86 architectures: http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en
- For x64 architectures: http://www.microsoft.com/downloads/details.aspx?FamilyID=ba9257ca-337f-4b40-8c14-157cfdffee4e&displaylang=en

❑ **If you are using Visual Studio 2010:**

You must have Visual Studio 2010 Service Pack 1 or the Microsoft Visual C++ 2010 SP1 Redistribution Package installed on the machine where you are *running* an application built with the release libraries of the following RTI architecture packages:

- i86Win32VS2010 built with dynamic libraries
- x64Win64VS2010 built with dynamic libraries
- i86Win32dotnet4.0 and x64Win64dotnet4.0

To run an application built with *debug* libraries of the above RTI architecture packages, you must have Visual Studio 2010 Service Pack 1 installed.

The Microsoft Visual C++ 2010 Service Pack 1 Redistribution Package can be obtained from the following Microsoft website:

- For x86 architectures:

  http://www.microsoft.com/download/en/details.aspx?id=5555

- For x64 architectures:

  http://www.microsoft.com/download/en/details.aspx?id=14632

❑ **If you are using Visual Studio 2012:**

You must have Visual C++ Redistributable for Visual Studio 2012 Update 3 installed on the machine where you are *running* a C++ application built the release libraries of the following RTI architecture packages:

- i86Win32VS2012 built with dynamic libraries
- x64Win64VS2012 built with dynamic libraries
- i86Win32dotnet4.5 and x64Win64dotnet4.5

You can download Visual C++ Redistributable for Visual Studio 2012 Update 3 from this Microsoft website: http://www.microsoft.com/en-ca/download/details.aspx?id=30679

Chapter 3 Using Distributed Logger in Your Connext Application

There are two ways to use *Distributed Logger*: directly through its API or by attaching it to an existing logging framework as an 'appender' or a 'handler.' Using the API directly is straightforward, but keep in mind that *Distributed Logger* is not intended to be a full-featured logging library. In particular, it does *not* contain the ability to log messages to standard out/error. Rather, it is primarily intended to be integrated into third-party logging infrastructures.

Distributed Logger comes with third-party integrations for the open-source project log4j (http://logging.apache.org/log4j/) as well as Java's built-in logging library (java.util.logging). Please see Examples (Section 3.3) for examples that illustrate these integrations.

Distributed Logger captures and forwards *Connext* internal information, warning, and error messages using a DDS topic. It monitors these messages using the same mechanism as user log messages.

These *Connext* log messages are sent over DDS automatically as soon as you initialize *Distributed Logger* (by calling **RTI_DL_DistLogger_getInstance()** in C or C++, or **Logger.getLogger(...)** in Java; see the API Reference HTML documentation for details).

3.1 Distributed Logger Libraries

Table 3.1 lists the libraries you will need in order to use *Distributed Logger*.

Table 3.1 **Required Libraries**

Platform	**Language**	**Static**		**Dynamic**	
		Release	**Debug**	**Release**	**Debug**
Linux®	C++	rtidlcz rtidlcppz	rtidlczd rtidlcppzd	rtidlc rtidlcpp	rtidlcd rtidlcppd
	Java ®	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar
VxWorks™	C	rtidlcz.a rtidlccppz.a	rtidlczd.a rtidlcppzd.a	rtidlc.so rtidlcpp.so	rtidlcd.so rtidlcppd.so
Windows®	C	rtidlcz	rtidlczd	rtidlc	rtidlcd

3.2 Using the API Directly

Details on using the *Distributed Logger* APIs are provided in the API Reference HTML documentation: ***<install-directory***[1]**>/doc/html/rti_distributed_logger/<*language*>**. Start by opening **index.html**.

If you plan to use the *Distributed Logger*'s API directly, here are a couple of notes. To configure the options, create an options object and update its fields. Once your updates are complete, set the options on *Distributed Logger*. It is important that this be done before *Distributed Logger* is instantiated. *Distributed Logger* acts as a singleton and there is no way to change the options after it has been created.

When your application is ready to exit, use the 'delete' method. This will delete all Entities and threads associated with *Distributed Logger*.

3.3 Examples

Distributed Logger includes several examples in ***<install-directory***[1]**>/example**:

1. **C/Distributed_Logger/Hello_distributed_logger**

   This is a simple example of how to use the API directly and does not publish or subscribe to any Topics except the ones related to *Distributed Logger*.

2. **CPP/Distributed_Logger/Hello_distributed_logger**

   This is a simple example of how to use the API directly and does not publish or subscribe any Topics except the ones related to *Distributed Logger*.

3. **JAVA/Distributed_Logger/Hello_direct_usage**

   This is a simple example of how to use the API directly and does not publish or subscribe any Topics except the ones related to *Distributed Logger*.

4. **JAVA/Distributed_Logger/Hello_file_logger**

   This example shows how an application can use the information provided by *Distributed Logger*. As the name suggests, this example subscribes to log messages and writes them to a file. Multiple domains can be subscribed to simultaneously if desired. The example is meant to strike a balance between simplicity and function. Certainly more features could be added to make it a production-ready application but that would obscure the goal of the example.

5. **JAVA/Distributed_Logger/Hello_java_util_logging**

   This is an adaptation of the Hello_idl example which replaces all System.{out/err} invocations with Java logging library equivalents. It adds *Distributed Logger* through a configuration file.

6. **JAVA/Distributed_Logger/Hello_log4j_logging**

   This is an adaptation of the Hello_idl example which replaces all System.{out/err} invocations with log4j library equivalents. It adds *Distributed Logger* through a configuration file.

Each example has a **READ_ME.txt** file which explains how to build and run it.

[1] <*install-directory*> is where you installed *Connext* and *Distributed Logger*, such as /opt/rti/ndds.<*version*> on UNIX-based systems, or C:\Program Files\RTI\ndds.<*version*> on Windows systems.

3.4 Data Type Resource

You can find the data types used by *Distributed Logger* in **resource/rtidistlogger/idl/distog.idl**.

If you want to generate code and interact with *Distributed Logger* through Topics, you can use this file to do so. You will need to provide extra command-line arguments to *rtiddsgen*. (This allows us to accommodate multiple language bindings within the same file. As a consequence, we've used preprocessor definitions to achieve this functionality.) The command-line options which must be added to *rtiddsgen* are as follows:

- ❑ For C or C++: **–D LANGUAGE_C**
- ❑ For Java: **–D LANGUAGE_JAVA**
- ❑ For .Net: **–D LANGUAGE_DOTNET**

Important: If you plan to use the generated code in your application (to subscribe to log messages, for instance) be aware that the type names used might not match the default ones. *Do not* use the generated generated type names obtained when calling **get_type_name()** or found in **distlogSupport.h**. Use the variables in Table 3.2 instead.

Table 3.2 **Registration Names for each Distributed Logger Type**

Type	Registered Typename	Variable
Log Message	com::rti::dl::LogMessage	C/C++: RTI_DL_LOG_MESSAGE_TYPE_NAME Java: LOG_MESSAGE_TYPE_NAME.VALUE
Administration State	com::rti::dl::admin::State	C/C++: RTI_DL_STATE_TYPE_NAME Java: STATE_TYPE_NAME.VALUE
Administration Command Request	com::rti::dl::admin:: CommandRequest	C/C++: RTI_DL_COMMAND_REQUEST_TYPE_NAME Java: COMMAND_REQUEST_TYPE_NAME.VALUE
Administration Command Response	com::rti::dl::admin:: CommandResponse	C/C++: RTI_DL_COMMAND_RESPONSE_TYPE_NAME Java: COMMAND_RESPONSE_TYPE_NAME.VALUE

For instance, to subscribe to log messages in C you will need to do the following:

```
retcode = RTI_DL_LogMessageTypeSupport_register_type(
                              participant, RTI_DL_LOG_MESSAGE_TYPE_NAME);
```

3.5 Distributed Logger Topics

Distributed Logger uses four Topics to publish log messages, state, and command responses and one topic to subscribe to command requests. These are detailed in Table 3.3.

Table 3.3 **Topics Used by Distributed Logger**

Topic	Type Name	Quality of Service
rti/distlog	com::rti::dl::LogMessage	Reliable Transient Local
rti/distlog/administration/state	com::rti::dl::admin::State	Reliable Transient Local
rti/distlog/administration/command_request	com::rti::dl::admin::CommandRequest	Reliable
rti/distlog/administration/command_response	com::rti::dl::admin::CommandResponse	Reliable

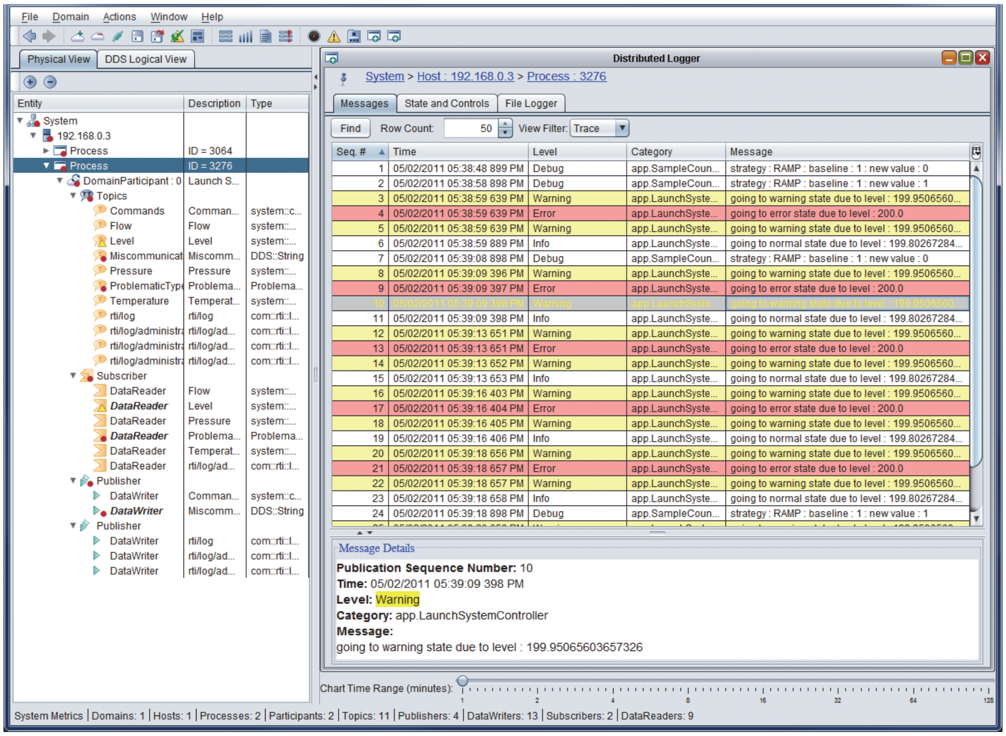
3.6 Distributed Logger IDL

The IDL describing the types used for Topics created by *Distributed Logger* are in ***<Distributed Logger distribution folder>*****/resource/rtidistlogger/idl/distlog.idl**. You can use this IDL to create custom applications that use the data provided by *Distributed Logger* and/or to remotely control any *Distributed Logger* instances that are running in your system. The IDL has been designed to take advantage of the latest type-support features in *RTI Connext*.

3.7 Viewing Log Messages

One way to see the messages from *Distributed Logger* is to use *Monitor*, a separate GUI-based application. *Monitor* is a component of *Connext Messaging*.

Figure 3.1 **Viewing Log Messages with RTI Monitor**



Entity	Description	Type
System		
192.168.0.3		
Process	ID = 3064	
Process	ID = 3276	
DomainParticipant : 0	Launch S...	
Topics		
Commands	Comman...	system::c...
Flow	Flow	system::...
Level	Level	system::...
Miscommunicat	Miscomm...	DDS::String
Pressure	Pressure	system::...
ProblematicType	Problema...	Problema...
Temperature	Temperat...	system::...
rti/log	rti/log	com::rti::l...
rti/log/administra	rti/log/ad...	com::rti::l...
rti/log/administra	rti/log/ad...	com::rti::l...
rti/log/administra	rti/log/ad...	com::rti::l...
Subscriber		
DataReader	Flow	system::...
DataReader	Level	system::...
DataReader	Pressure	system::...
DataReader	Problema...	Problema...
DataReader	Temperat...	system::...
DataReader	rti/log/ad...	com::rti::l...
Publisher		
DataWriter	Comman...	system::c...
DataWriter	Miscomm...	DDS::String
Publisher		
DataWriter	rti/log	com::rti::l...
DataWriter	rti/log/ad...	com::rti::l...
DataWriter	rti/log/ad...	com::rti::l...

Seq. #	Time	Level	Category	Message
1	05/02/2011 05:38:48 899 PM	Debug	app.SampleCoun...	strategy : RAMP : baseline : 1 : new value : 0
2	05/02/2011 05:38:58 898 PM	Debug	app.SampleCoun...	strategy : RAMP : baseline : 1 : new value : 1
3	05/02/2011 05:38:59 639 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
4	05/02/2011 05:38:59 639 PM	Error	app.LaunchSyste...	going to error state due to level : 200.0
5	05/02/2011 05:38:59 639 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
6	05/02/2011 05:38:59 889 PM	Info	app.LaunchSyste...	going to normal state due to level : 199.80267284...
7	05/02/2011 05:39:08 898 PM	Debug	app.SampleCoun...	strategy : RAMP : baseline : 1 : new value : 0
8	05/02/2011 05:39:09 396 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
9	05/02/2011 05:39:09 397 PM	Error	app.LaunchSyste...	going to error state due to level : 200.0
10	05/02/2011 05:39:09 398 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
11	05/02/2011 05:39:09 398 PM	Info	app.LaunchSyste...	going to normal state due to level : 199.80267284...
12	05/02/2011 05:39:13 651 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
13	05/02/2011 05:39:13 651 PM	Error	app.LaunchSyste...	going to error state due to level : 200.0
14	05/02/2011 05:39:13 652 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
15	05/02/2011 05:39:13 653 PM	Info	app.LaunchSyste...	going to normal state due to level : 199.80267284...
16	05/02/2011 05:39:16 403 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
17	05/02/2011 05:39:16 404 PM	Error	app.LaunchSyste...	going to error state due to level : 200.0
18	05/02/2011 05:39:16 405 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
19	05/02/2011 05:39:16 406 PM	Info	app.LaunchSyste...	going to normal state due to level : 199.80267284...
20	05/02/2011 05:39:18 656 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
21	05/02/2011 05:39:18 657 PM	Error	app.LaunchSyste...	going to error state due to level : 200.0
22	05/02/2011 05:39:18 657 PM	Warning	app.LaunchSyste...	going to warning state due to level : 199.9506560...
23	05/02/2011 05:39:18 658 PM	Info	app.LaunchSyste...	going to normal state due to level : 199.80267284...
24	05/02/2011 05:39:18 898 PM	Debug	app.SampleCoun...	strategy : RAMP : baseline : 1 : new value : 1

Message Details

Publication Sequence Number: 10
Time: 05/02/2011 05:39:09 398 PM
Level: Warning
Category: app.LaunchSystemController
Message:
going to warning state due to level : 199.95065603657326

System Metrics | Domains: 1 | Hosts: 1 | Processes: 2 | Participants: 2 | Topics: 11 | Publishers: 4 | DataWriters: 13 | Subscribers: 2 | DataReaders: 9

Other ways to see the log messages include using *rtiddsspy* or writing your own visualization tool. If you do want to write your own application that interacts with *Distributed Logger*, you can find the IDL in the **resource/distlog/idl** folder of the installation.

3.8 Logging Levels

Log levels in *Distributed Logger* are organized as follows (ordered by importance). This table also shows the mapping between logging levels in the *Connext* middleware and *Distributed Logger*.

Connext Logger Log Level	Distributed Logger Log Level
NDDS_CONFIG_LOG_LEVEL_ERROR	RTI_DL_ERROR_LEVEL
NDDS_CONFIG_LOG_LEVEL_WARNING	RTI_DL_WARNING_LEVEL
NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL	RTI_DL_NOTICE_LEVEL
NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE	RTI_DL_INFO_LEVEL
NDDS_CONFIG_LOG_LEVEL_DEBUG	RTI_DL_DEBUG_LEVEL

3.9 Distributed Logger Quality of Service Settings

To ensure that *Distributed Logger* works correctly with other RTI tools, some QoS settings are hard-coded and cannot be modified by customized profiles. Table 3.4 lists the QoS values that are set in *Distributed Logger*. Values in bold are hard-coded; therefore even if they appear in an XML profile, they remain as noted in the table.

Table 3.4 **QoS Values Used by Distributed Logger**

Entity	Property	Value
Subscriber	Presentation.access_scope	PRES_INSTANCE_PRESENTATION_QOS
	Presentation.coherent_access	false
	Presentation.ordered_access	false
Publisher	Presentation.access_scope	PRES_INSTANCE_PRESENTATION_QOS
	Presentation.coherent_access	false
	Presentation.ordered_access	false
Log Message Topic	**Reliability.kind**	**DDS_RELIABLE_RELIABILITY_QOS**
	Durability.kind	**DDS_TRANSIENT_LOCAL_DURABILITY_QOS**
Administration State Topic	**Reliability.kind**	**DDS_RELIABLE_RELIABILITY_QOS**
	Durability.kind	**DDS_TRANSIENT_LOCAL_DURABILITY_QOS**
Administration Command Request Topic	**Reliability.kind**	**DDS_RELIABLE_RELIABILITY_QOS**
Administration Command Response Topic	**Reliability.kind**	**DDS_RELIABLE_RELIABILITY_QOS**
Log Message DataWriter	Ownership.kind	DDS_SHARED_OWNERSHIP_QOS
	Latency_budget.duration.sec	0
	Latency_budget.duration.nanosec	0
	Liveliness.kind	DDS_AUTOMATIC_LIVELINESS_QOS
	Destination_order.kind	DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
	Reliability.kind	DDS_RELIABLE_RELIABILITY_QOS
	Durability.kind	DDS_TRANSIENT_LOCAL_DURABILITY_QOS
	History.kind	**DDS_KEEP_LAST_HISTORY_QOS**
	History.depth	**10**

Table 3.4 **QoS Values Used by Distributed Logger**

Entity	Property	Value
Administration State DataWriter	Ownership.kind	DDS_SHARED_OWNERSHIP_QOS
	Latency_budget.duration.sec	0
	Latency_budget.duration.nanosec	0
	Liveliness.kind	DDS_AUTOMATIC_LIVELINESS_QOS
	Destination_order.kind	DDS_BY_RECEPTION_TIMESTAMP_ DESTINATIONORDER_QOS
	Reliability.kind	DDS_RELIABLE_RELIABILITY_QOS
	Durability.kind	DDS_TRANSIENT_LOCAL_DURABILITY_QOS
	History.kind	**DDS_KEEP_LAST_HISTORY_QOS**
	History.depth	**1**
Administration Command Response DataWriter	Ownership.kind	DDS_SHARED_OWNERSHIP_QOS
	Latency_budget.duration.sec	0
	Latency_budget.duration.nanosec	0
	Liveliness.kind	DDS_AUTOMATIC_LIVELINESS_QOS
	Destination_order.kind	DDS_BY_RECEPTION_TIMESTAMP_ DESTINATIONORDER_QOS
	Reliability.kind	DDS_RELIABLE_RELIABILITY_QOS
	History.kind	**DDS_KEEP_LAST_HISTORY_QOS**
	History.depth	**10**
Administration Command Request DataReader	Ownership.kind	DDS_SHARED_OWNERSHIP_QOS
	Latency_budget.duration.sec	DDS_DURATION_INFINITE_SEC
	Latency_budget.duration.nanosec	DDS_DURATION_INFINITE_NSEC
	Deadline.period.sec	DDS_DURATION_INFINITE_SEC
	Deadline.period.nanosec	DDS_DURATION_INFINITE_NSEC
	Liveliness.kind	DDS_AUTOMATIC_LIVELINESS_QOS
	Destination_order.kind	DDS_BY_RECEPTION_TIMESTAMP_ DESTINATIONORDER_QOS
	Reliability.kind	DDS_RELIABLE_RELIABILITY_QOS
	History.kind	**DDS_KEEP_LAST_HISTORY_QOS**
	History.depth	**10**

Chapter 4 Enabling Distributed Logger in RTI Services

Many RTI components provide integrated support for *Distributed Logger* (check the component's *Release Notes*) and include the *Distributed Logger* library in their distribution. To enable *Distributed Logger* in these components, modify their XML configuration file. In the <administration> section, add the <distributed_logger> tag as shown in this example:

```
<persistence_service name="default">
    <administration>
        <domain_id>10</domain_id>
        <distributed_logger>
            <enabled>true</enabled>
            <filter_level>DEBUG</filter_level>
            <queue_size>2048</queue_size>
            <thread>
               <priority>THREAD_PRIORITY_BELOW_NORMAL</priority>
               <stack_size>8192</stack_size>
               <cpu_list>
                    <element>0</element>
                    <element>1</element>
               </cpu_list>
               <cpu_rotation>THREAD_SETTINGS_CPU_NO_ROTATION</cpu_rotation>
            </thread>
        </distributed_logger>
    </administration>
    ...
</persistence_service>
```

The tags supported within the <distributed_logger> tag are described in Table 4.1.

4.1 Relationship Between Service Verbosity and Filter Level

A service's verbosity influences the way the log messages reach *Distributed Logger* and their quantity. If a service (such as *RTI Persistence Service*, *RTI Routing Service,* or another service that is integrated with *Distributed Logger*) is configured with a low verbosity, it will not pass a lot of messages to *Distributed Logger*, even if the *Distributed Logger* filter level is set to a very verbose one (such as TRACE). On the contrary, a high verbosity will work better, because it will pass more messages to *Distributed Logger;* in this case the filter level will have more effect.

Note: Since *Distributed Logger* uses a separate thread to send log messages, there is little impact on performance with more verbose filter levels. However, there is some performance penalty in services that use a higher verbosity.

Table 4.1 **Distributed Logger Tags**

Tags within <distributed_logger>	Description	Number of Tags Allowed
<enabled>	Controls whether or not *Distributed Logger* should be enabled at start up. This field is required. Allowed values: TRUE or FALSE	1 (required)
<filter_level>	The filter level for the log messages to be sent. *Distributed Logger* uses the filter level to discard log messages before they can be sent from the application/service. This is the minimum log level that will be sent out over the network. For example, when using the NOTICE level, any INFO, DEBUG and TRACE-level log messages will be filtered out and not sent from the application/service to *Connext*. **See important information in** Section 4.1. Can be set to these values: • SILENT • FATAL • SEVERE • ERROR • WARNING • NOTICE • INFO • DEBUG • TRACE (most verbose level, default)	0 or 1
<queue_size>	The size of an internal message queue used to store log messages before they are written to DDS. Default, 128 log messages.	0 or 1
<thread>	See Table 4.2.	0 or 1

Table 4.2 **Distributed Logger Thread Tags**

Tags within <distributed_logger> <thread>	Description	Number of Tags Allowed
<cpu_list>	Each <element> specifies a processor on which the *Distributed Logger* thread may run. <cpu_list> <element>*value*</element> </cpu_list> Only applies to platforms that support controlling CPU core affinity (see the *Core Libraries and Utilities Platform Notes*).	0 or 1
<cpu_rotation>	Determines how the CPUs in <cpu_list> will be used by the *Distributed Logger* thread. The value can be either: • THREAD_SETTINGS_CPU_NO_ROTATION The thread can run on any listed processor, as determined by OS scheduling. • THREAD_SETTINGS_CPU_RR_ROTATION The thread will be assigned a CPU from the list in round-robin order. Only applies to platforms that support controlling CPU core affinity (see the *Core Libraries and Utilities Platform Notes*).	0 or 1
<mask>	A collection of flags used to configure threads of execution. Not all of these options may be relevant for all operating systems. May include these bits: • STDIO • FLOATING_POINT • REALTIME_PRIORITY • PRIORITY_ENFORCE It can also be set to a combination of the above bits by using the "or" symbol (|), such as STDIO|FLOATING_POINT. Default: MASK_DEFAULT	0 or 1
<priority>	Thread priority. The value can be specified as an unsigned integer or one of the following strings. • THREAD_PRIORITY_DEFAULT • THREAD_PRIORITY_HIGH • THREAD_PRIORITY_ABOVE_NORMAL • THREAD_PRIORITY_NORMAL • THREAD_PRIORITY_BELOW_NORMAL • THREAD_PRIORITY_LOW When using an unsigned integer, the allowed range is platform-dependent.	0 or 1
<stack_size>	Thread stack size, specified as an unsigned integer or set to the string THREAD_STACK_SIZE_DEFAULT. The allowed range is platform-dependent.	0 or 1