

RTI Distributed Logger

Getting Started Guide

Version 5.0



Your systems. Working as one.



© 2012 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
August 2012.

Trademarks

Real-Time Innovations, RTI, and Connexant are trademarks or registered trademarks of Real-Time Innovations, Inc. All other trademarks used in this document are the property of their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

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

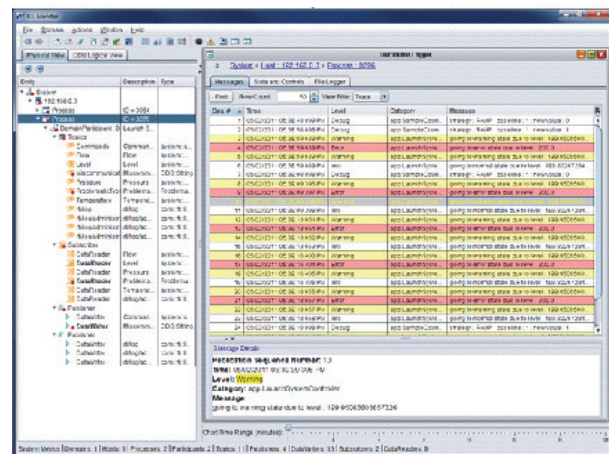
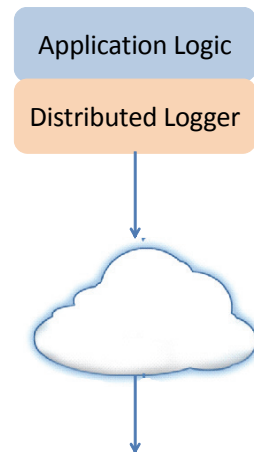
1	Welcome to RTI Distributed Logger	
2	Installing Distributed Logger	
3	Using Distributed Logger in Your Connex Application	
3.1	Enabling Distributed Logging.....	3-1
3.2	Using the API Directly.....	3-2
3.3	Examples	3-2
3.4	Data Type Resource.....	3-3
3.5	Distributed Logger Topics.....	3-4
3.6	Viewing Log Messages	3-5
3.7	Logging Levels.....	3-6
4	Enabling Distributed Logger in RTI Services	
4.1	Relationship Between Service Verbosity and Filter Level	4-4

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 *rtiddspy* 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 ConnexTM Professional Edition*. Use the installation instructions in this chapter only if you are installing *RTI Distributed Logger* independently (not as part of *RTI Connex Professional Edition*).

1. Make sure you have already installed a compatible version of *Connex* (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 *Connex*. 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 *Connex*. 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/>).

Chapter 3 Using Distributed Logger in Your Connex 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.

Connex internal information, warning, and error messages can captured and forwarded using a DDS topic and monitored using the same mechanism as user log messages. These *Connex* 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 Enabling Distributed Logging

[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
Windows®	C	rtidlcz	rtidlczd	rtidlc	rtidlcd
Linux®	C++	rtidlcz rtidlcppz	rtidlczd rtidlcppzd	rtidlc rtidlcpp	rtidlcd rtidlcppd
	Java ®	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

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*.

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

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 **README.txt** file which explains how to build and run it.

3.4 Data Type Resource

You can find the data types used by *Distributed Logger* in **resource/dl/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`

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 3.2](#).

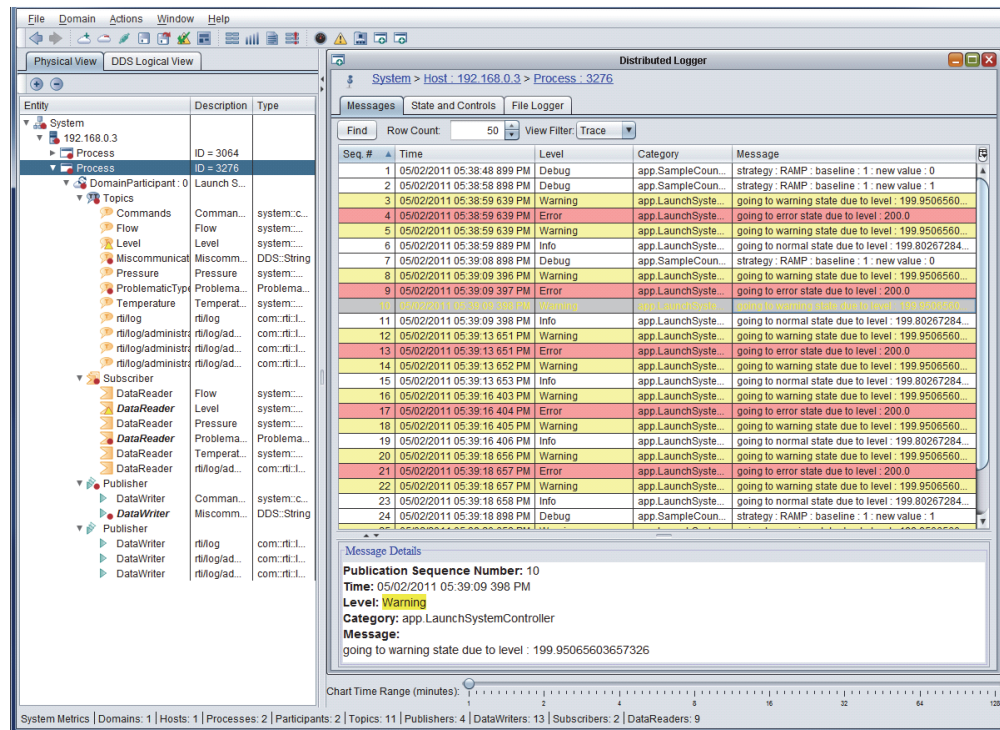
Table 3.2 **Topics Used by Distributed Logger**

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 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 *Connnext Messaging*.

Figure 3.1 Viewing Log Messages with RTI Monitor



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.

3.7 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 *Connex* middleware and *Distributed Logger*.

Connex 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

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](#).

Table 4.1 Distributed Logger Tags

Tags within <distributed_logger>	Description	Number of Tags Allowed
<enabled>	Controls whether or not <i>Distributed Logger</i> 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. <i>Distributed Logger</i> 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 <i>Connex</i> . 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 the message queue used to store the messages that are awaiting to be sent. Default: 128 bytes	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 <i>Distributed Logger</i> thread may run. <pre><cpu_list> <element>value</element> </cpu_list></pre> Only applies to platforms that support controlling CPU core affinity (see the <i>Core Libraries and Utilities Platform Notes</i>).	0 or 1
<cpu_rotation>	Determines how the CPUs in <cpu_list> will be used by the <i>Distributed Logger</i> thread. The value can be either: • <code>THREAD_SETTINGS_CPU_NO_ROTATION</code> The thread can run on any listed processor, as determined by OS scheduling. • <code>THREAD_SETTINGS_CPU_RR_ROTATION</code> 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 <i>Core Libraries and Utilities Platform Notes</i>).	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: • <code>STDIO</code> • <code>FLOATING_POINT</code> • <code>REALTIME_PRIORITY</code> • <code>PRIORITY_ENFORCE</code> It can also be set to a combination of the above bits by using the “or” symbol (<code> </code>), such as <code>STDIO FLOATING_POINT</code>. Default: <code>MASK_DEFAULT</code>	0 or 1

Table 4.2 Distributed Logger Thread Tags

Tags within <distributed_logger> <thread>	Description	Number of Tags Allowed
<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

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.