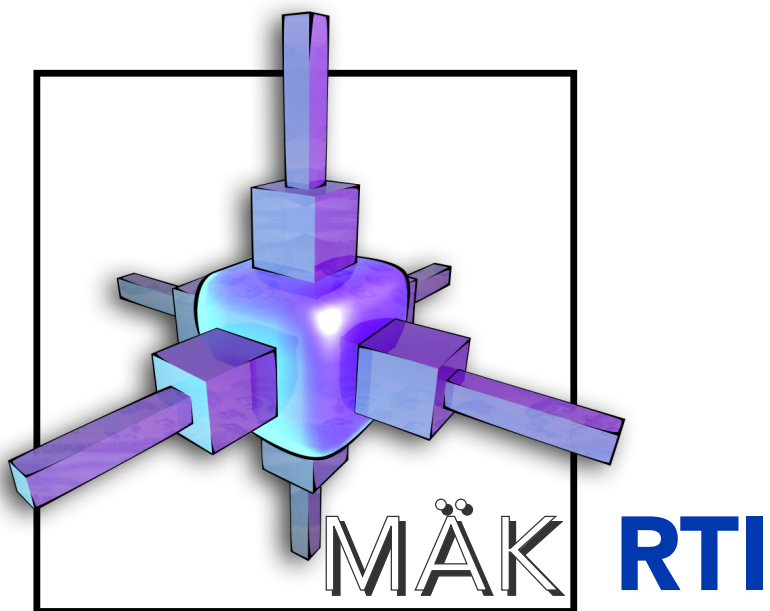
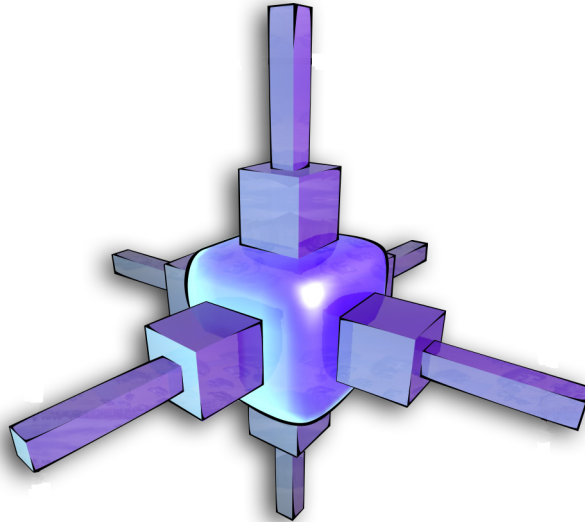




Reference Manual



MÄK RTI

Reference Manual

Copyright © 2010 VT MÄK
All rights Reserved. Printed in the United States.

Under copyright laws, no part of this document may be copied or reproduced in any form without prior written consent of VT MÄK.

VR-Exchange™ and VR-Vantage™ are trademarks of VT MÄK. MÄK Technologies®, VR-Forces®, RTIspy®, B-HAVE®, and VR-Link® are registered trademarks of VT MÄK.

The ZLIB library is copyright 1995-2004 Jean-loup Gailly and Mark Adler.

All other trademarks are owned by their respective companies.

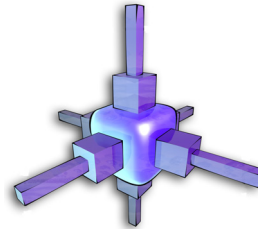
For additional third party license information, please see [“Third Party Licenses,”](#) on page xviii.

VT MÄK
68 Moulton St.
Cambridge, MA 02138 USA

Voice: 617-876-8085
Fax: 617-876-9208

info@mak.com
www.mak.com

Revision RTI-4.0-1-100722



Contents

Preface

How the Manual is Organized	xi
Documentation	xiii
MÄK Products	xiv
How to Contact Us	xvi
Document Conventions	xvii
Hardware and Operating System Naming Conventions	xvii
Mouse Button Naming Conventions	xviii
Third Party Licenses	xviii
Boost License	xviii
libXML and libICONV	xix
pThreads Library	xix
EHS, PCRE, and PME	xix

Chapter 1 Introduction to the MÄK RTI

1.1. About RTIs	1-2
1.1.1. Obtaining HLA RTI Specification Documents	1-2
1.1.2. Obtaining HLA Certification for a Federate	1-2
1.2. Compatibility with Other RTI Implementations	1-3
1.2.1. Dynamic Linking	1-3
1.2.2. The FED File and FDD File	1-4
1.2.3. The RID File	1-4
1.2.4. Network Compatibility	1-5
1.3. MÄK RTI Applications	1-5
1.4. The RTIsy Diagnostic GUI	1-6
1.5. The RTIsy Plug-in API	1-6
1.6. Support for the HLA 1516 Specification	1-7
1.6.1. RTI Interoperability	1-8

Chapter 2 MÄK RTI Concepts

2.1. Using the MÄK RTI with Multithreaded Applications	2-2
2.2. The MÄK RTI Process Model	2-3
2.2.1. The Role of the tick() Function in the MÄK RTI	2-4
2.2.2. Asynchronous I/O	2-5
2.2.3. Asynchronous Message Processing	2-6
2.2.4. Asynchronous Callbacks	2-6
2.3. Network Topologies for RTI Messages	2-7
2.3.1. Best Effort Transport on a LAN	2-7
2.3.2. Reliable Transport and TCP Forwarding on a LAN	2-7
2.3.3. Best Effort Transport on a WAN	2-7
2.3.4. Centralized TCP Forwarding on a WAN	2-8
2.3.5. Distributed Forwarding on a WAN	2-8

Chapter 3 Compiling and Linking with the MÄK RTI

3.1. Library Names	3-2
3.2. Compiling and Linking with the MÄK RTI 1.3	3-2
3.2.1. Compiling and Linking on Linux	3-2
3.2.2. Compiling and Linking on Windows	3-3
3.3. Compiling and Linking with the MÄK RTI 1516	3-3
3.3.1. Compiling and Linking on UNIX	3-4
3.3.2. Compiling and Linking on Windows	3-4
3.4. Compiling and Linking with HLA Evolved	3-5
3.5. Federation Time	3-5
3.5.1. The HLA 1.3 Time Implementation	3-5
3.5.2. The RTI 1516 Time Implementation	3-6
3.5.3. The HLA Evolved Time Implementation	3-7
3.6. Java Bindings for the MÄK RTI	3-7

Chapter 4 The RID File

4.1. The RID File	4-2
4.1.1. Search Order for RID Files	4-2
4.1.2. Configuring HLA Switch Values	4-3
4.1.3. Configuring Use of Advisory Messages	4-3
4.1.4. Consistency of RID Files	4-3
4.1.5. Specifying RID Parameters Programmatically in HLA 1516 ..	4-4
4.2. Displaying Configuration Settings in the RTI Federations View	4-5
4.3. Displaying RTI Configuration Parameters in the RTIspy	4-7
4.3.1. Editing a Parameter	4-8
4.4. The RID File Consistency Checker	4-9
4.4.1. Enabling RID File Consistency Checking	4-10
4.4.2. Selecting the Parameters to be Checked	4-11

Chapter 5 RTI Applications

5.1. The rtiexec	5-2
5.1.1. Configuring the rtiexec	5-3
5.1.2. Configuring Diagnostic Messages	5-3
5.2. The RTI Assistant	5-4
5.2.1. Specifying the RTI Assistant's Log Update Interval	5-5
5.2.2. Shutting Down the RTI Assistant	5-5
5.2.3. Configuring the RTI Assistant's View Log Prompt	5-6
5.2.4. Configuring the Display of RTI Assistant Messages	5-7
5.2.5. Specifying the Warning Tick Interval	5-7
5.2.6. Common Operations in RTI Assistant Log Windows	5-8
5.2.7. Specifying the RTI Assistant's Ports	5-9
5.2.8. Specifying the RTI Assistant's Multicast Address	5-9
5.2.9. Disabling the RTI Assistant	5-9
5.3. The RTI Forwarder Application	5-10
5.3.1. Running the RTI Forwarder	5-10
5.4. The RTI Command Line Tool	5-12
5.4.1. Listing Federation Executions and Federates	5-13
5.4.2. Deleting Federates	5-14

Chapter 6 Setting Up the RTIsPy

6.1. Installing and Configuring the RTIsPy	6-2
6.1.1. Configuring the RTIsPy	6-2
6.2. Running the RTIsPy Web Servers	6-4
6.2.1. Opening the RTIsPy Web Page	6-4
6.2.2. URL Structure	6-5
6.2.3. Choosing Between Local and Centralized HTTP Servers	6-6
6.3. Connecting to the RTIsPy Web Servers	6-8
6.3.1. Bookmarking	6-9
6.3.2. Pop-up Blockers	6-9
6.3.3. Minimizing Overhead	6-9
6.4. Performance Issues for the RTIsPy	6-10

Chapter 7 Managing Federations

7.1. Introduction	7-3
7.2. Specifying the FED File	7-3
7.2.1. Search Order for FED Files	7-4
7.2.2. Distributing the FED File	7-4
7.3. Using Lightweight Mode	7-5
7.3.1. Limitations of Lightweight Mode	7-6
7.4. Viewing Network Components	7-7
7.4.1. The RTIsPy Network Map	7-7
7.4.2. The RTI Network Component View	7-9
7.4.3. Viewing the MÄK RTI Components List	7-10

7.5. Viewing Information about a Federation	7-11
7.5.1. Viewing Federation Information in the RTIspy	7-12
7.6. Viewing Time Management State Information	7-13
7.6.1. Viewing Time Management State Information in the RTIspy	7-15
7.7. Viewing Synchronization Points	7-16
7.8. Viewing Federation-Wide Notification Messages	7-17
7.9. Configuring Fault Tolerance Strategies	7-18
7.9.1. Responding to Abnormal Termination	7-18
7.9.2. Reconnecting to the RTI Forwarder	7-20
7.9.3. Specifying the Response Interval	7-20
7.9.4. Processing Discovery of Unknown Objects	7-21
7.9.5. Choosing Silent Failure Instead of Exceptions	7-21
7.10. Configuring Save/Restore	7-22
7.11. Viewing Notification Messages from the rtiexec	7-22
7.11.1. Displaying Recent Log Entries	7-22

Chapter 8 Managing Federates

8.1. Introduction	8-2
8.1.1. Configuring Federate Connections in the RID File	8-2
8.2. Displaying Federate Information in the RTIspy	8-3
8.2.1. Displaying Federate LRC Details	8-4
8.3. Displaying Federate Objects and Interactions	8-5
8.3.1. Displaying the Object Instances Page	8-5
8.3.2. Displaying Object Attribute Information	8-6
8.3.3. Undiscovering Objects	8-7
8.3.4. Displaying Federate Interactions	8-7
8.4. Displaying FOM Object and Interaction Classes	8-8
8.4.1. Displaying a Class's Attributes or Parameters	8-8
8.4.2. Displaying the Classes a Federate is Publishing and Subscribing To	8-9
8.5. Displaying the Federate LRC Event Log	8-10
8.6. Viewing LRC Diagnostic Messages	8-11
8.6.1. Enabling LRC Diagnostic Data Logging in the RID File	8-11
8.6.2. Enabling LRC Diagnostic Data Logging from the RTI Assistant	8-12
8.6.3. Displaying LRC Diagnostic Messages	8-13
8.7. Using the RTIspy to Debug an Application	8-14

Chapter 9 Monitoring the Network

9.1. Introduction	9-2
9.2. Testing Latency Between Federates	9-3
9.2.1. Enabling Latency Testing	9-4
9.2.2. Running Latency Tests	9-4
9.2.3. Interpreting the Results of Latency Tests	9-5

9.2.4. Running Latency Tests Manually	9-8
9.2.5. Running Continuous Latency Tests	9-8
9.2.6. Configuring Latency Tests	9-9
9.3. The RTIsPy Network Monitoring Tool	9-10
9.3.1. Enabling Network Monitoring	9-10
9.3.2. Logging Network Monitoring Statistics	9-10
9.3.3. Displaying Object and Interaction Statistics	9-11
9.3.4. Displaying RTI Messages	9-12
9.3.5. The RTI API Call Statistics Page	9-13
9.4. Examples of Using the RTIsPy Network Monitoring Tool	9-14
9.4.1. Finding Bottlenecks Caused by Processing Callbacks	9-14
9.4.2. Identifying the Cause of Slow-Downs Under Time Management	9-15
9.4.3. Comparing the Costs and Benefits of RTI Features	9-16

Chapter 10 Managing Network Traffic

10.1. Choosing Best Effort or Reliable Transport	10-2
10.2. Packet Bundling for Reliable and Best Effort Messages	10-3
10.3. Automatically Locating the RTI Forwarder and rtiexec	10-3
10.4. Configuring Sender-Side Filtering (Smart TCP Forwarding)	10-4
10.4.1. Configuring Sender-Side Filtering for rtiexec Messages	10-6
10.5. Compressing RTI Messages	10-6
10.6. Enabling Support for Large Attributes and Parameters	10-7
10.7. Configuring the Network Buffer Sizes	10-7
10.8. Choosing the Network Device to Use	10-8
10.9. Controlling Congestion for Best Effort Sockets	10-8
10.10. Configuring Asynchronous Processing	10-8
10.11. Buffering Automatic Update Requests (1516 and Evolved)	10-9
10.12. Filtering by Multicast Groups Using Declaration Management ...	10-9
10.12.1. Assigning DM Multicast Groups to Network Interfaces .	10-10

Chapter 11 Configuring Message Forwarding

11.1. Configuring UDP and Centralized TCP Forwarding on a LAN	11-3
11.1.1. Configuring Best Effort (UDP) on a LAN	11-3
11.1.2. Configuring Centralized TCP Forwarding on a LAN	11-4
11.2. Configuring UDP and Centralized TCP on a WAN	11-4
11.2.1. Configuring Centralized TCP Forwarding	11-5
11.2.2. Configuring UDP Over a WAN without Distributed Forwarding	11-6
11.3. Distributed Forwarding Topologies on a LAN	11-7
11.3.1. Forwarder Groups	11-7
11.3.2. Singleton Forwarders	11-9
11.3.3. Advanced Distributed Forwarding on a LAN	11-10
11.4. Distributed Forwarding Topologies on a WAN	11-11
11.4.1. Optimizing Distributed Forwarding across a WAN	11-12

11.5. Configuring Federates for Distributed Forwarding	11-15
11.6. Configuring RTI Forwarders for Distributed Forwarding	11-16
11.6.1. Configuring RTI Forwarders Using the RTI Assistant	11-17
11.6.2. Creating a Routes File	11-18
11.6.3. Optional RTI Forwarder Configuration Parameters	11-20
11.6.4. Configuring Single-Portal Forwarder Groups	11-21
11.6.5. Configuring Load-Balanced Forwarder Groups	11-23
11.6.6. Configuring Distributed UDP Forwarding	11-24
11.7. Configuring Routers or Firewalls for WAN Federations	11-26
11.8. Special Distributed Forwarder Networks	11-26
11.8.1. Configuring Hierarchical Forwarder Networks	11-27

Chapter 12 Configuring HLA Services

12.1. Enabling and Disabling HLA Services	12-2
12.1.1. Configuring MOM Services	12-2
12.1.2. Configuring Time Management Services	12-2
12.2. Configuring Internal Messages	12-3
12.2.1. Configuring the Delay Time for Processing Internal Messages	12-3
12.2.2. Transmitting Internal Messages Across Ticks	12-4

Chapter 13 Using Data Distribution Management

13.1. Configuring DDM Services	13-2
13.1.1. Enabling DDM	13-2
13.1.2. Configuring Distributed Region DDM	13-2
13.1.3. Configuring a Fixed Grid DDM	13-3
13.1.4. Configuring Implicit DDM	13-5
13.2. The Distributed Region Approach	13-7
13.2.1. Attribute Passelization	13-9
13.3. The Fixed Grid Approach	13-9
13.3.1. The Fixed Grid Representation	13-11
13.3.2. Federate Region Intersections with the Fixed Grid	13-12
13.3.3. Normalization of Federate Data for Range Bounds	13-14
13.3.4. The Fixed Grid Approach and DDM Services	13-15
13.4. Implicit DDM	13-17
13.4.1. Implicit DDM Implementation	13-18
13.4.2. Adding Implicit DDM to the FED or FDD	13-18
13.4.3. Implicitly Associating DDM Regions	13-20
13.4.4. Modifying Region Extents	13-20
13.4.5. Modifying Fixed Grid Region Extents	13-22

Chapter 14 Running the MÄK RTI Using Shared Memory

14.1. Introduction	14-2
14.2. Configuring Use of Shared Memory	14-2

14.2.1. Enabling Shared Memory Mode	14-3
14.2.2. Specifying the Name of the Shared Memory Region	14-4
14.2.3. Specifying the Shared Memory Queue Length	14-4
14.2.4. Specifying the Shared Memory Manager Maximum Wait Interval	14-5
14.2.5. Message Queue Limits	14-5
14.3. Starting the Shared Memory Manager	14-5
14.3.1. Shared Memory Queue Manager Command-Line Options	14-6
14.4. Terminating the Shared Memory Manager	14-8
14.4.1. Abnormal Termination of the Shared Memory Manager	14-9
14.5. Tuning Shared Memory	14-9
14.5.1. Calculating the Queue Length	14-9
14.5.2. The RTI tick() Member Function	14-11
14.5.3. The Shared Memory Manager	14-12

Chapter 15 Using FOM Modules

15.1. Using FOM Modules	15-2
15.2. Merging FOM Modules into the Current FOM	15-3
15.3. Creating and Joining with FOM Modules	15-4
15.4. Validating FOM Modules	15-5
15.5. Configuring Use of Modular FOMs	15-6
15.5.1. Configuring a MOM Module	15-6
15.5.2. Configuring Create FOM Modules	15-7
15.5.3. Configuring Join FOM Modules	15-7
15.5.4. Configuring Use of a Local FOM Module File	15-8
15.5.5. Example FOM Module Configuration for 1.3 and 1516	15-8

Chapter 16 Update Rate Reduction

16.1. Update Rate Reduction	16-2
16.2. How Update Reduction is Implemented	16-4
16.3. Configuring Update Rate Reduction	16-5
16.3.1. Specifying the Update Rate in 1.3 and 1516	16-5
16.3.2. Specifying the Rate for Object Class Subscriptions in 1.3 and 1516	16-6
16.3.3. Specifying the Receive Filtering Tolerance	16-6
16.3.4. Example Update Rate Reduction Configuration (1.3 and 1516 Only)	16-7

Chapter 17 The RTIsPy Plug-in API

17.1. Introduction	17-2
17.2. The RTI Managers	17-3
17.3. Initializing a Plug-in	17-4
17.4. Monitoring RTI Service Invocations	17-6
17.4.1. Monitoring Federate Ambassador Services	17-7

Contents

17.5. Creating Custom RTI Managers	17-9
17.6. Subclassing DtRtiAmbassadorImplementor	17-10
17.7. Communicating with Remote LRCs and the rtiexec	17-11
17.7.1. Time Factory Wrapper	17-11
17.7.2. The Connection Manager	17-12
17.7.3. The Asynchronous I/O Connection Manager	17-13
17.7.4. Changing the RTI's Communications Infrastructure	17-14
17.7.5. Messages	17-17
17.8. Responding to Dropped Federates	17-19
17.8.1. Responding to Missing Heartbeats	17-19
17.9. Finding Out when Data is Available to be Read	17-20
17.10. <i>rtiexec</i> Plug-ins	17-21
17.11. Accessing RID File Parameters	17-22
17.12. Compiling and Linking	17-23
17.12.1. Compiling and Linking on UNIX	17-23
17.12.2. Compiling and Linking on Windows	17-24
17.13. Loading Plug-Ins	17-24

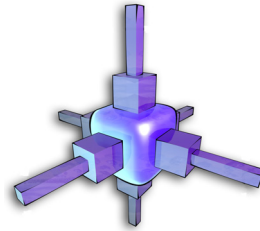
Appendix A RID File Parameters Reference

A.1. MÄK Technologies Lisp (MTL) Files	A-2
A.1.1. Using Environment Variables in an MTL File	A-2
A.1.2. Using Conditional Statements	A-3
A.2. RID File Parameters	A-4
A.3. Federation-Wide Parameters	A-4
A.4. LRC-Specific Parameters	A-14
A.5. Configuring tick(min,max)	A-27

Appendix B Example Federates

B.1. Example Federates	B-2
B.2. The rtisimple Example	B-2
B.3. The simpleDDM Example	B-4
B.4. The simpletime Example	B-5
B.5. HLA Bounce	B-7
B.5.1. Running HLA Bounce	B-7
B.5.2. Demonstrating Ownership Management	B-9
B.5.3. Demonstrating DDM	B-9

Index



Preface

This manual is for developers of HLA simulation applications and for persons who need to install or use the MÄK RTI.

This manual assumes that you are familiar with the command-line and graphical windowing environments for your operating system.

For information about changes to the MÄK RTI since this manual went to press, please see *MÄK RTI Release Notes*.

How the Manual is Organized

This manual is organized as follows:

Chapter 1, *Introduction to the MÄK RTI* describes the MÄK RTI and its features.

Chapter 2, *MÄK RTI Concepts*, provides conceptual information about the MÄK RTI that will help you understand when and why you would want to use particular features and configuration options.

Chapter 3, *Compiling and Linking with the MÄK RTI*, explains how to link applications to the MÄK RTI.

Chapter 4, *The RID File*, explains how to maintain and distribute RID files. It does not describe parameters. Parameters are described in Appendix A, *RID File Parameters Reference*.

Chapter 5, *RTI Applications*, describes the rtiexec, the RTI command line tool, the RTI Forwarder, and how to configure the RTI Assistant.

Chapter 6, *Setting Up the RTIspy*, explains how to configure the RTIspy web-based GUI.

Chapter 7, *Managing Federations*, explains how to use the RTI Assistant and RTIspy GUI to manage federations. It also describes lightweight mode and fault tolerance strategies.

Chapter 8, *Managing Federates* explains how to use the RTI Assistant and RTIspy GUI to manage federates.

Chapter 9, *Monitoring the Network*, describes how to use the RTI Assistant and RTIspy GUI to monitor the performance of federates. It also describes how to run latency tests.

Chapter 10, *Managing Network Traffic* describes parameters used to configure the RTI's interaction with the network, except for those used to configure message forwarding.

Chapter 11, *Configuring Message Forwarding*, explains basic message forwarding strategies, the different types of distributed forwarding topologies, and how to configure them.

Chapter 12, *Configuring HLA Services*, explains how to configure MOM and time management services and how to configure internal messages.

Chapter 13, *Using Data Distribution Management*, provides conceptual background for the MÄK RTI's DDM implementations.

Chapter 14, *Running the MÄK RTI Using Shared Memory*, explains how to configure use of shared memory for co-located federates.

Chapter 15, *Using FOM Modules*, explains the MÄK RTI's implementation of FOM modules.

Chapter 16, *Update Rate Reduction*, explains the MÄK RTI's implementation of update rate reduction.

Chapter 17, *The RTIspy Plug-in API*, explains how to use the RTIspy API to create plug-ins for the MÄK RTI.

Appendix A, *RID File Parameters Reference* describes the format of the RID file and each RID file parameter.

Appendix B, *Example Federates*, describes the sample federates provided with the MÄK RTI.

Documentation

The MÄK RTI documentation set is as follows:

- ♦ *MÄK RTI Users Guide* is a brief introduction to HLA and the MÄK RTI and explains how to configure the MÄK RTI startup environment and run a federate.
- ♦ *MÄK RTI Reference Manual* provides detailed information about configuring the MÄK RTI, designing network topologies for HLA federations, and how to use the RTIspy API.
- ♦ *MÄK RTI First Experience Guide* shows you how easy it is to run HLA federates with the MÄK RTI.
- ♦ *MÄK RTI Release Notes* system information and updates for the current release.
- ♦ MÄK RTIspy API class documentation and MÄK RTI API class documentation in HTML format. You can access the class documentation from the Start menu in Windows or by opening *./classdoc/index.html*.
- ♦ Online help. The RTI Assistant windows and the RTIspy web GUI have online help.
- ♦ *MÄK Products Interoperability Guide* explains how MÄK products work together and provides troubleshooting information.

MÄK Products

The MÄK RTI is a member of the VT MÄK line of software products designed to streamline the process of developing and using networked simulated environments. The VT MÄK product line includes the following:

- ♦ VR-Link® Network Toolkit. VR-Link is an object-oriented library of C++ functions and definitions that implement the High Level Architecture (HLA), the Distributed Interactive Simulation (DIS) protocol, and the Test and Training Enabling Architecture (TENA). VR-Link has built-in support for the RPR FOM and allows you to map to other FOMs. This library minimizes the time and effort required to build and maintain new HLA or DIS-compliant applications, and to integrate such compliance into existing applications.

VR-Link includes a set of sample debugging applications and their source code. The source code serves as an example of how to use the VR-Link Toolkit to write applications. The executables provide valuable debugging services such as generating a predictable stream of HLA or DIS messages, and displaying the contents of messages transmitted on the network.

- ♦ MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is optimized for high performance. It has an API, RTIsPy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface (the RTI Assistant) that helps users with configuration tasks and managing federates and federations.
- ♦ VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a GUI, that gives you a 2D and 3D views of a simulated environment.

You can create and view local entities, aggregate them into hierarchical units, assign tasks, set state parameters, and create plans that have tasks, set statements, and conditional statements. VR-Forces also functions as a plan view display for viewing remote entities taking part in an exercise. Using the toolkit, you can extend the VR-Forces application or create your own application for use with another user interface.

- ♦ VR-Vantage™. VR-Vantage is a line of products designed to meet your simulation visualization needs. It includes three end-user applications — VR-Vantage Stealth, VR-Vantage XR, and VR-Vantage IG — and the VR-Vantage Toolkit. VR-Vantage Stealth displays a realistic, 3D view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway.

VR-Vantage IG is a configurable desktop image generator (IG) for out the window (OTW) scenes and remote camera views. It has most of the features of the Stealth, but is optimized for its IG function.

VR-Vantage XR provides a common operational picture using the same 3D views as VR-Vantage Stealth, a 2D plan view, and an exaggerated reality (XR) view. Together these views provide both situational awareness and the big picture of the simulated world.

The VR-Vantage Toolkit is a 3D visual application development toolkit. Use it to customize or extend MÄK's VR-Vantage applications, or to integrate VR-Vantage capabilities into your custom applications. VR-Vantage is built on top of OpenSceneGraph (OSG). The toolkit includes the OSG version used to build VR-Vantage.

- **MÄK Data Logger.** The Data Logger, also called the Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal and can quickly jump to areas of interest. The Logger has a GUI and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- **MÄK Plan View Display.** The Plan View Display (PVD) provides a 2D, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display.
- **VR-Exchange™.** VR-Exchange allows simulations that use incompatible communications protocols to interoperate. For example, within the HLA world, using VR-Exchange, federations using the HLA RPR FOM 1.0 can interoperate with simulations using RPR FOM 2.0, or federations using different RTIs can interoperate. VR-Exchange supports HLA, TENA, and DIS translation.

How to Contact Us

For RTI technical support, information about upgrades, and information about other MÄK products, you can contact us in the following ways:

Telephone

Call or fax us at:	Voice:	617-876-8085 (extension 3 for support)
	Fax:	617-876-9208

E-mail

Sales and upgrade information:	info@mak.com
Technical support:	support@mak.com
VR-Vantage support:	vrv-support@mak.com

Internet

MÄK web site home page:	www.mak.com
License key requests:	www.mak.com/support/licenses.php
Product version and platform information:	www.mak.com/support/productversions.php
For the free, unlicensed MÄK RTI:	www.mak.com/products/rti.php
Support FAQ:	www.mak.com/support/faq.php

Post

Send postal correspondence to:	VT MÄK 68 Moulton St. Cambridge, MA, USA 02138

When requesting support, please tell us the product you are using, the version, and the platform on which you are running.

Document Conventions

This manual uses the following typographic conventions:

<code>Monospaced</code>	Indicates commands or values you enter.
Monospaced Bold	Indicates a key on the keyboard.
<i>Monospaced Italic</i>	Indicates command variables that you replace with appropriate values.
{ }	Indicates required arguments.
[]	Indicates optional arguments.
	Separates options in a command where only one option may be chosen at a time.
()	In command syntax, indicates equivalent alternatives for a command-line option, for example, (-h --help).
/	Indicates a directory. Since MÄK products run on both UNIX and Windows PC platforms, we use the / (slash) for generic discussions of pathnames. If you are running on a PC, substitute a \ (backslash) when you type pathnames.
<i>Italic</i>	Indicates a file name, pathname, or a class name.
sans Serif	Indicates a parameter or argument.
➤	Indicates a one-step procedure.
Menu → Option	Indicates a menu choice. For example, an instruction to select the Save option from the File menu appears as: Choose File → Save .
i	Indicates supplemental or clarifying information.
!	Indicates additional information that you must observe to ensure the success of a procedure or other task.

Directory names are preceded with dot and slash characters that show their position with respect to the RTI home directory. For example, the directory *makRtix.x/doc* appears in the text as *./doc*.

Hardware and Operating System Naming Conventions

Unless otherwise specified, references to UNIX include Linux, regardless of the type of computer it is running on. References to PCs refer to Intel processor-compatible computers running a version of Microsoft Windows.

Mouse Button Naming Conventions

An instruction to click the mouse button, refers to clicking the primary mouse button, usually the left button for right-handed mice and the right button for left-handed mice. The popup menu refers to the menu displayed when you click the secondary mouse button, usually the right button on right-handed mice and the left button on left-handed mice.

Third Party Licenses

MÄK software products may use code from third parties. This section contains the license documentation required by these third parties.

Boost License

VR-Link, and all MÄK software that uses VR-Link uses some code which is distributed under the Boost License. All header files that contain Boost code are properly attributed. The boost web site is: www.boost.org.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the “Software”) to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libXML and libICONV

VR-Link and all MÄK software that uses VR-Link, links in libXML and libICONV. On some platforms the compiled libraries and header files are distributed with MÄK Products. MÄK has made no modifications to these libraries. For more information about these libraries please see the following web sites:

- ♦ The LGPL license is available at: <http://www.gnu.org/licenses/lgpl.html>
- ♦ Information about IconV is at: <http://www.gnu.org/software/libiconv/>
- ♦ Information about LibXML is at: <http://xmlsoft.org/>

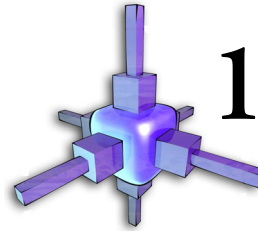
pThreads Library

VR-Exchange links with the pThreads win32 library. The library is distributed under the GNU Lesser General Public License (LGPL). MÄK has made no modification to this library. For information about the pThreads win32 library please see: <http://sourceware.org/pthreads-win32/>

EHS, PCRE, and PME

The MÄK RTI links with the EHS, PCRE, and PME libraries. These libraries are distributed under the GNU Lesser General Public License (LGPL). MÄK has made modification to these libraries only to allow them to compile on the supported platforms. For more information about these libraries please see the following web sites:

- ♦ EHS: <http://xaxxon.slackworks.com/ehs/>
- ♦ PCRE: <http://www.pcre.org/>
- ♦ PME: <http://xaxxon.slackworks.com/pme/index.html>



Introduction to the MÄK RTI

The MÄK RTI is an implementation of the Run-Time Infrastructure portion of the High-Level Architecture (HLA).

About RTIs	1-2
Obtaining HLA RTI Specification Documents	1-2
Obtaining HLA Certification for a Federate	1-2
Compatibility with Other RTI Implementations	1-3
Dynamic Linking	1-3
The FED File and FDD File	1-4
The RID File	1-4
Network Compatibility	1-5
MÄK RTI Applications	1-5
The RTIsy Diagnostic GUI	1-6
The RTIsy Plug-in API	1-6
Support for the HLA 1516 Specification	1-7
RTI Interoperability	1-8

1.1. About RTIs

The MÄK RTI (Run-Time Infrastructure) is a software library (and supporting executables) that implements the HLA 1.3, 1516 (SISO DLC HLA API 1516), and HLA Evolved (IEEE 1516-2010) interface specifications. In HLA, applications exchange FOM data through RTI calls, which means that all HLA applications need to use an RTI. (Applications that use VR-Link typically do not need to make RTI calls directly, but only because VR-Link makes the RTI calls for them.) The MÄK RTI has been optimized for the high performance needs of the real-time simulation community.

The MÄK RTI has been verified by the Modeling and Simulation Coordination Office (MSCO, formerly DMSO) as fully compliant with the HLA Interface Specification, version 1.3 and with the IEEE 1516 version of the HLA Interface Specification (SISO DLC HLA API 1516 (SISO-STD-004.1-2004)). All services are implemented.



In this manual and *MÄK RTI Users Guide*, when we use the term “HLA 1516” or 1516, we mean the SISO DLC HLA API 1516 specification. When we use the term “HLA Evolved”, we mean IEEE 1516-2010. When we discuss a subject that applies equally to HLA 1516 and HLA Evolved, we refer generically to the “HLA 1516 specifications”.

1.1.1. Obtaining HLA RTI Specification Documents

The HLA 1.3, IEEE 1516, and HLA Evolved specifications are copyrighted by their owners and MÄK cannot distribute them to customers. You can get documentation for the RTI interface specifications at:

- <http://www.msco.mil/TechSpecs.html> (HLA 1.3)
- <http://shop.ieee.org/store/> (HLA 1516 and HLA Evolved specifications)
- www.sisostds.org (SISO DLC HLA API 1516).

1.1.2. Obtaining HLA Certification for a Federate

If you plan to seek HLA certification for a federate, note the following:

- You must use an RTI that is fully compliant with the relevant HLA standard.
- If you use the MÄK RTI, you must run it in fully compliant mode. To force full compliance, set the `RTI_forceFullCompliance` parameter to 1 in *rid.mtl*. For more information, please see “[Enabling and Disabling HLA Services](#),” on page 12-2.

1.2. Compatibility with Other RTI Implementations

This section discusses the following aspects of compatibility:

- ♦ Dynamic linking
- ♦ Static linking
- ♦ The FED file and FDD file
- ♦ The RID file
- ♦ Network compatibility.

1.2.1. Dynamic Linking

The MÄK RTI 1.3 implements the API dictated by the 1.3 version of the HLA Interface Specification, as amended by MSCO's official HLA 1.3 Interpretations document. All RTIs that are verified as compliant with the HLA 1.3 specification are written to this exact API standard. This insures that the MÄK RTI is link-compatible with other RTIs that have been verified as HLA compliant.

The MÄK RTI 1516 implements the SISO DLC HLA API¹ 1516 (SISO-STD-004.1-2004). This API supports dynamic link compatibility, which was problematic with the original IEEE 1516 API. The IEEE 1516 API only supports compile-time compatibility (it uses C++ templates and implementation-specific header files). The MÄK RTI is compatible with any RTI written to the SISO DLC HLA API 1516.

The MÄK RTI HLA Evolved implements the IEEE 1516-2010 standard. HLA Evolved incorporates the SISO Dynamic Link Compatible API. Therefore, federates compiled with the HLA Evolved version of MÄK RTI can be used with other RTIs that support the specification without being recompiled. Other important features include Modular FOMs, update rate reduction, improvements to reliability and connectivity, and other changes that make using HLA easier.

Because an RTI is typically accessed by an application through a dynamic library (*.dll* on Windows, *.so* on UNIX), applications do not need to be recompiled or relinked to switch among dynamic-link-compatible RTI implementations that have been built using the same compilers. The only requirement is that the appropriate version of the library be located within the shared library search path. For more information, please see Chapter 3, *Compiling and Linking with the MÄK RTI*.

1. Simulation Interoperability Standards Organization Dynamic Link Compatible High Level Architecture Application Program Interface

1.2.2. The FED File and FDD File

In both the HLA 1.3 specification and the 1516 specifications, you must provide FOM data to the RTI. In the 1.3 specification, this is done with the federation execution data (FED) file. The 1516 specifications use a FOM Document Data (FDD) file. However the format is slightly different between 1516 and HLA Evolved.

All versions of the MÄK RTI support both the FDD and FED file formats. FDD files must have a file extension of either *.xml* or *.fdd*. FED files are expected to have an extension of *.fed* or *.mtl*; however, any other extensions are assumed to be in the FED format.

The FED file format is part of the HLA 1.3 standard, so the MÄK RTI 1.3 uses the same FED file format as other compliant RTIs. You do not need to modify your FOM or FED file to switch between the MÄK RTI 1.3 and other RTI 1.3 implementations.

There may be differences among RTI implementations as to where the FED file must be located. By default, the MÄK RTI requires that the FED file must be available to be read by each federate in the federation. An alternative is for the federation execution to distribute the FED file to each Local RTI Component (LRC) when the federate joins. For more details, please see [“Specifying the FED File,”](#) on page 7-3.



Since the FED file and FDD file perform the same function, and since for the MÄK RTI they are essentially interchangeable, in this manual, unless otherwise noted, references to the FED file should be considered as references to the FED file and the FDD file.

Before you run an HLA application, you must make sure that your federates (and the rtiexec if you are running it) can find the FED file for the federation you are running. For details, please see [“Specifying the FED File,”](#) on page 7-3.

1.2.3. The RID File

The RTI initialization data (RID) file is the configuration file for an RTI. The MÄK RTI RID file is written using MÄK Technologies Lisp (MTL). Unless you want to rely on default values for all RTI configuration parameters, you need to make sure that all federates (and the rtiexec if you are running it) can find the appropriate RID files. For details, please see Chapter 4, [The RID File](#).

1.2.4. Network Compatibility

The MÄK RTI is not network compatible with RTI implementations from other vendors. This means that within a given federation, you cannot run one federate with the MÄK RTI and another federate with a different RTI. In general, no two RTI implementations are network compatible, due to differences in the low-level network mechanisms that they use (which include, but are not limited to packet layout).

The term Run-Time Infrastructure refers to the entire system used to communicate among federates. By HLA's design, choosing an RTI is a federation-wide choice, and not a federate-by-federate choice. Applications that want to interoperate in the same federation must all use the same RTI. A federation can easily switch from one RTI implementation to another between runs, but during each run, all participants must agree on which RTI to use, much as they must also agree on which FOM to use.

1.3. MÄK RTI Applications

The MÄK RTI includes the following core and utility applications:

- ♦ The rtiexec implements any centralized functionality required by the RTI. ([“The rtiexec,”](#) on page 5-2)
- ♦ The RTI Assistant monitors federate and rtiexec activity and provides helpful messages when needed. It also provides access to error logs and configuration options. It is accessed through an icon in the system tray. ([“The RTI Assistant,”](#) on page 5-4)
- ♦ The RTI Forwarder supports centralized TCP forwarding, distributed TCP forwarding, and distributed UDP forwarding. ([“The RTI Forwarder Application,”](#) on page 5-10)
- ♦ The RTI Chooser (Windows only) helps you configure your RTI environment. (For details, please see Section 8.2, [“Configuring the MÄK RTI Version and RID File to Use”](#), in *MÄK RTI Users Guide*.)
- ♦ The RTI Command Line Tool lets you send a limited number of commands to the rti from a console window. ([“The RTI Command Line Tool,”](#) on page 5-12)

1.4. The RTIspy Diagnostic GUI

The RTIspy GUI is a web-based graphical user interface (GUI) for observing the rtiexec, RTI Forwarders, and federates in a federation. It allows you to monitor one or more of these components using a web browser from anywhere in the federation (or even remotely). The views range from a network map that shows all the components in the federation, to inspection of the internals of a specific federate. You can also monitor multiple RTI components simultaneously for further analysis and side-by-side comparisons. With the web-based GUI you can:

- View a network map
- View information about the federation and federates
- View information about RTI Forwarders
- View RID parameters
- View an LRC's FOM subscriptions and publications
- View a federate's registered and discovered objects
- View the interactions a federate has sent and received
- View a federate's log of calls invoked by the federate ambassador and RTI ambassador
- Monitor a federate's LRC network performance
- Profile a federate's interaction with the RTI API.

The RTIspy duplicates much of the federation information provided by the RTI Assistant. Its advantage is the ability to monitor a federation from any web browser. It also adds the ability to monitor the LRC information for individual federates and view the network connections between RTI components. The procedures for configuring and running the RTIspy are in Chapter 6, *Setting Up the RTIspy*. Procedures for using the RTIspy web GUI are provided in the chapters that explain how to manage federations and federates and how to monitor federations.

1.5. The RTIspy Plug-in API

The RTIspy Plug-in API allows you to write code that can monitor the internal state of the RTI and modify the RTI's behavior. For information about the RTIspy Plug-in API, please see Chapter 17, *The RTIspy Plug-in API*.

1.6. Support for the HLA 1516 Specification

The MÄK RTI 1516 supports the full set of services defined by the HLA 1516 interface specification. The MÄK RTI 1516 implements SISO-STD-004.1-2004.

The C++ API that was published as part of the IEEE 1516 standard had two significant problems. First, because it was published before any reference implementations had been developed, it contained a series of bugs that made it impossible to implement as-is. Second, its heavy reliance on templates made it impossible to assure that different RTI implementations would be link-compatible.

The SISO Dynamic Link Compatibility HLA API Product Development Group (HLA API PDG, <http://www.sisostds.org/index.cfm>) developed an alternative mapping of the IEEE 1516 Interface Specification to a C++ API (SISO-STD-004.1-2004). This alternative 1516 API addresses the bugs in the API that was published along with the IEEE standard, and also allows for dynamic-link compatibility among different RTI implementations. As part of this process, the HLA API PDG migrated the API away from the use of templates for defining the API classes and towards simpler C++ declarations.

Some of the key changes are:

- The use of the “pImpl” idiom for handles (decouples the class interfaces from their private member declarations)
- The use of simple enumerations (which are strongly typed in C++)
- Passing parameters by reference instead of `auto_ptr` (except for factories).

Although the Dynamic Link Compatible 1516 API more closely resembles the 1.3 API than the original 1516 API does, some significant differences remain. For example, an RTI namespace is used instead of a top-level RTI class, templates are still used to represent handles (to allow different implementations of handles), and the API uses Standard Template Library (STL) maps to represent sets of attribute handle/value pairs.

By implementing against the SISO 1516 API, MÄK promotes link compatibility with other RTI vendors, which is not possible with the original IEEE 1516 API. The IEEE 1516 API only allowed compile-time compatibility (especially true with the use of templates). However, link compatibility is important because it allows federates built using one RTI to execute with the local RTI component (LRC) library and central RTI component (CRC) executable of a different RTI without having to recompile the federate.

i

While the 1516 API uses the `std::wstring` to exchange string information, internally, the MÄK RTI 1516 operates on narrow strings only (that is, representable by `std::string`). Therefore, the contents of any wide string passed through the RTI API must be convertible to a narrow string.

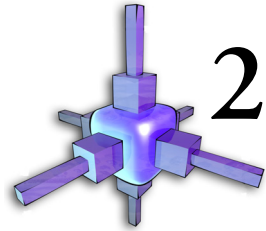
1.6.1. RTI Interoperability

All versions of the MÄK RTI use the same code base. As a result, they share the same on-the-wire protocol for exchanging internal RTI information. For a subset of RTI services, a federate using the MÄK RTI 1516 or HLA Evolved can interoperate with a federate using the MÄK RTI 1.3. However, the differences in the functionality or format between the 1.3 API and the 1516 APIs create some conflicts that do not allow interoperability, as follows:

- ♦ Ownership management is not interoperable between the 1.3 and 1516 APIs. The 1516 interface specification introduced some additional ownership management calls that do not exist in the 1.3 interface specification. Ownership management is interoperable between HLA 1516 and HLA Evolved except for the Attribute Ownership Release Denied service, which is new for HLA Evolved.
- ♦ MOM is not interoperable between any of the three specifications.
- ♦ 1.3 federates and 1516 federates can use natural FOM files when interoperating in the same federation if the files have symmetric class and attribute/parameter definitions.
- ♦ The publish and subscribe services in the 1516 and HLA Evolved interface specifications are additive when successive calls are made, versus replacement for the 1.3 interface specification. The MÄK RTI 1516 and HLA Evolved implementations follow the additive policy; the MÄK RTI 1.3 follows the replacement policy. As a result, the advisories and discoveries may conflict when a federation is composed of both 1.3 and 1516 federates. The `RTI_processUnknownUpdatesForDiscovery` RID parameter should be enabled as a work around for discoveries.
- ♦ The interoperability of DDM services between 1.3 and 1516 requires special consideration. Because the 1516 DDM services do not use routing spaces, the 1.3 routing space dimensions must be placed into a global routing space. As a result, all dimensions with the same name across different routing spaces resolve to a single dimension. The 1.3 federates can still manipulate regions as if the FOM routing spaces were defined; however, the region matching and overlap calculations only consider the dimensions with respect to a global space. This feature is supported by enabling the `RTI_extend13and1516interop` parameter. When this feature is disabled, the 1.3 DDM services retain the use of the FOM routing spaces.

DDM is interoperable between HLA 1516 and HLA Evolved.

- ♦ Use of FOM modules is interoperable among the three specifications.
- ♦ Update Rate Reduction works across protocols, as long as the RID file defines the same update rates as are present in the HLA Evolved FOM.



MÄK RTI Concepts

This chapter provides a brief introduction to how MÄK has implemented the RTI specifications. This information will help you understand and put to use the various configuration options described in later chapters.

Using the MÄK RTI with Multithreaded Applications.....	2-2
The MÄK RTI Process Model.....	2-3
The Role of the tick() Function in the MÄK RTI.....	2-4
Asynchronous I/O	2-5
Asynchronous Message Processing	2-6
Asynchronous Callbacks	2-6
Network Topologies for RTI Messages	2-7
Best Effort Transport on a LAN	2-7
Reliable Transport and TCP Forwarding on a LAN.....	2-7
Best Effort Transport on a WAN	2-7
Centralized TCP Forwarding on a WAN.....	2-8
Distributed Forwarding on a WAN	2-8

2.1. Using the MÄK RTI with Multithreaded Applications

The MÄK RTI LRC is thread-safe, but by default, it is not re-entrant. The LRC is only re-entrant when using the asynchronous callback process model described in [“Asynchronous Callbacks,”](#) on page 2-6.

The concept of thread-safety versus re-entrancy is very important for multi-threaded federates. Since the MÄK RTI LRC is thread-safe, multiple federate threads can create RTI and Federate Ambassadors within a single process. This is important for several classes of federate such as bridge federates, which may join multiple federations simultaneously. So, thread-safety means that it is safe to use single or multiple instances of the LRC in a multi-threaded federate, but it does not mean that multiple federate threads can access a single RTI Ambassador instance.

To optimize performance for the majority of federates, the MÄK RTI is not re-entrant by default. To guarantee re-entrancy, the RTI would have to lock internally during each access. Locking of this sort at the API level can create a significant amount of overhead, which offers no benefit in most federates. This means that all access to a single RTI Ambassador instance must be made by a single federate thread. If multiple threads require access to the LRC, then the federate is responsible for ensuring correct locking of all data passed to the RTI. In general, the federate is able to perform more selective and optimal locking than the RTI through its knowledge of interactions between the federate threads. Therefore, the recommended practice is to create a separate thread for RTI access and provide thread-safe communication from the RTI thread to the other federate threads.

If a federate requires multi-threaded access to a single LRC instance and cannot follow the procedure described above, then the federate must use the asynchronous callback processing model. When asynchronous callbacks are enabled, multiple federate threads can access the RTI Ambassador simultaneously, and the RTI will provide Federate Ambassador callbacks in a new thread created by the RTI. However, the asynchronous callback model usually increases federate complexity significantly.

2.2. The MÄK RTI Process Model

The MÄK RTI supports the following process models:

- ♦ Fully synchronous. All RTI processing is done in one thread and RTI network I/O, message processing, and federate ambassador callbacks are evoked when the federate calls tick().
- ♦ Asynchronous networking/synchronous message processing and callbacks. Network input is handled in a separate thread, but the input is buffered, not processed. Message processing and callbacks are handled in the main thread when the federate calls tick().
- ♦ Asynchronous networking and message processing/synchronous callbacks. Network input and message processing is handled in a separate thread. Callbacks are cached and delivered in the main thread when the federate calls tick().
- ♦ Fully asynchronous. Network I/O, message processing, and callbacks are handled in a separate thread. The callbacks are delivered asynchronously and these calls provided by the federate code must be thread safe. The tick() call becomes a NULL operation and the federate does not need to call it.

For details about configuring the process model, please see [“Configuring Asynchronous Processing,”](#) on page 10-8.

2.2.1. The Role of the tick() Function in the MÄK RTI

In the synchronous, asynchronous I/O, and asynchronous messages processing models, it is important to understand the role of the tick() function in HLA 1.3 and the evokeCallback() and evokeMultipleCallbacks() functions in HLA 1516 and HLA Evolved. For simplicity, the term tick will be used unless a distinction is necessary. Proper use of tick() is critical to federate and federation performance. As the names of the evokeCallback() and evokeMultipleCallbacks() member functions suggest, the primary purpose of tick() is to give the RTI processing cycles that it can use to invoke callbacks through the Federate Ambassador. In both the synchronous and asynchronous I/O models, Federate Ambassador callbacks are delivered only during the processing of the tick() function. For example, updates and interactions are delivered only when the federate calls tick(). Similarly, time advances can only be granted during the tick() call. For this reason alone, all federates (excluding federates using the asynchronous callbacks processing model) must call tick() regularly and at appropriate times (for example, after a name reservation or subscription) to ensure smooth operation within the federation.

Except for the asynchronous message processing and asynchronous callback modes, almost all internal RTI processing is performed during the tick() call. This provides federates with much greater control of the RTI's CPU usage, but it also means that federates must tick often in order to prevent starving the RTI of CPU time. This can be a major problem for any RTI algorithm that requires multiple messages to pass between joined federates or between federates and the rtiexec. Algorithms that require multiple message exchanges progress only during the call to tick(), and may require multiple calls to tick(). For example, the time management algorithms employed by the MÄK RTI sometimes require multiple messages between a federate and the rtiexec for a single Time Advance Grant. This means that if a federate does not tick frequently while waiting for the grant, then the time advance will slow down accordingly – this can affect the performance of all time constrained federates.

Synchronous mode federates are even more sensitive to tick frequency than federates using asynchronous I/O. Federates operating in synchronous mode read data from the network only during tick(). Therefore, ticking too slowly may lead to two distinct problems. First, if the federate ticks infrequently, then its local network buffers may overflow, leading to dropped best effort messages. Secondly, once the federate's network buffer overflows, reliable messages will begin to build up at the federate's RTI Forwarder. The forwarder will then gradually increase its memory usage until the federate clears the backlog of messages.

Finally, federates occasionally connect to the RTI (by creating an RTI Ambassador) well before they intend to participate in a federation. Such federates must still tick the RTI (even before they have Joined) in order to avoid the reliable messaging problem mentioned previously. This problem affects all synchronous and asynchronous I/O mode federates.

2.2.2. Asynchronous I/O

Asynchronous I/O allows a federate to better schedule its time and to better respond to peaks in I/O processing.

Many of the RTI services require data to be transmitted over the network. The federate LRC creates a separate I/O thread that processes network I/O operations separately from other internal RTI processing. The asynchronous I/O thread separates the network transmission portion of this processing from the federate's thread of execution. Since the network I/O is not performed during the processing of RTI Ambassador service calls, the RTI returns the thread of execution to the federate significantly faster. The federate is better able to respond to time-critical processing that might in turn generate an increased output load (for example, aircraft maneuvering). The I/O thread can also use spare CPU cycles when the federate is waiting for other system operations (for example, disk access). Similarly, the asynchronous I/O thread receives data from the network separately from the federate's thread of execution.

The I/O thread only performs network I/O operations. The federate must still invoke the `tick()` member function to allocate CPU cycles to the RTI for its internal processing and to generate federate ambassador callbacks. However, the federate is less likely to affect data transmission (for example, dropped messages or blocked TCP connections) if it fails to invoke the `tick()` function frequently enough (for example, during initialization phases).

Disadvantages of Asynchronous I/O

A disadvantage of asynchronous I/O is that it can increase message transmission latency. The timing of when messages are sent and received becomes less determined, since the federate LRC thread is no longer performing the actual I/O operations. However, the MÄK RTI uses a signaling strategy between the threads that allows the latency to be small while the RTI is not heavily loaded, and for it to take advantage of queuing when loads increase. The threads signal each other when messages are available for sending and receiving.

Another disadvantage of asynchronous I/O is that the federate thread can fail to yield enough processing cycles to the I/O thread. If the I/O thread becomes starved (that is, it cannot get enough processing cycles), it will not process network I/O. The RTI messages will not be sent or received, resulting in high latency and buffer overflows. In addition, operating system mechanisms can take effect that can be detrimental to federate performance. (For example, Windows gives starved threads control for some number of cycles.)

You can configure the interaction between the federate LRC thread and the I/O thread with parameters in the RID file ([Table A-1](#)). One of these parameters controls whether the LRC yields to the I/O thread during the `tick()` call (on by default). Having the federate LRC thread yield during the `tick()` call ensures that the I/O thread will be executed. We recommend that this parameter value should not be changed unless the federate has been designed to manage multithreading issues so that it sufficiently yields execution to other threads.

2.2.3. Asynchronous Message Processing

Asynchronous message processing is a process model that is halfway between asynchronous I/O, which simply supports asynchronous message buffering, and asynchronous callbacks, which supports fully asynchronous processing and callbacks. In the asynchronous message processing mode, the asynchronous thread handles the network I/O and processes received messages. However, any federate ambassador callbacks that are generated enter a queue to be delivered to the federate synchronously during its call to `tick()` or the `evokeCallback` RTI Ambassador service.

This processing mode relieves the federate from having to call `tick()` to provide the RTI with processing cycles. It may improve the response and efficiency of some distributed algorithms (for example, time management). However, it requires the caching of callback parameters, which results in additional memory allocation and data copy operations.

2.2.4. Asynchronous Callbacks

The MÄK RTI supports an asynchronous callback mode. Instead of buffering messages to be processed during the next RTI `tick()`, the asynchronous thread processes the messages directly. In this mode, the federate does not have to invoke `tick()`, because all internal RTI processing is performed in the RTI Ambassador calls or the asynchronous thread. This configuration requires that the Federate Ambassador callbacks be thread-safe.

Asynchronous callbacks are enabled in the RID file.



Specifying a callback model of `HLA_IMMEDIATE` in the `RTIambassador` connect service in HLA Evolved forces the RTI into asynchronous callbacks mode, regardless of the RID file setting.

2.3. Network Topologies for RTI Messages

The RTI can send messages using best effort transport (UDP) or reliable transport (TCP). It provides alternatives for optimizing performance both within a LAN and over a WAN.

2.3.1. Best Effort Transport on a LAN

When you use best effort transport within a LAN, messages are sent to the other federates on the LAN via UDP multicast or broadcast. There is no guarantee of receipt. Configuration is simple and there is little overhead. To configure best effort transport, configure the MÄK RTI as described in [“Choosing Best Effort or Reliable Transport,”](#) on page 10-2.

2.3.2. Reliable Transport and TCP Forwarding on a LAN

Reliable transport guarantees receipt of messages. When you send data to multiple recipients using TCP, a separate copy of each packet must be sent to each recipient. In general, there are two approaches for achieving this:

- ♦ A sender can maintain a TCP connection to each remote federate and can send a copy directly to each.
- ♦ A sender can maintain a connection to one or more RTI Forwarders – applications that receive TCP traffic from a sender, and then forward it to many recipients.

The first approach minimizes latency in federations that have a small number of federates, because packets reach their destinations in a single network hop. However, the second approach minimizes CPU processing time at each federate and minimizes the number of connections.

The MÄK RTI uses TCP forwarding for all reliable transmissions. It supports centralized TCP forwarding (one forwarder) using the RTI Forwarder application. The MÄK RTI supports distributed TCP forwarding through the use of multiple RTI Forwarder applications. This allows you to distribute the total amount of network traffic among the forwarders. In large federations, this can improve performance over that of a single centralized forwarder. For details about how to configure reliable transport on a LAN, please see [“Configuring UDP and Centralized TCP Forwarding on a LAN,”](#) on page 11-3.

2.3.3. Best Effort Transport on a WAN

Most routers do not support broadcast or multicast messages. Therefore, to send best effort messages over a WAN, you must specifically configure the addresses of the recipients or use a packet forwarding strategy. For details of distributed UDP forwarding, please see Distributed Forwarding later in this section. For details about configuring UDP over a WAN without packet forwarding, please see [“Configuring UDP Over a WAN without Distributed Forwarding,”](#) on page 11-6.

2.3.4. Centralized TCP Forwarding on a WAN

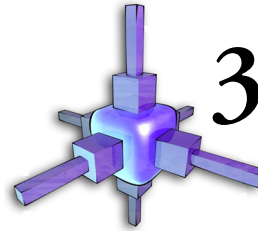
The centralized TCP forwarding configuration for the MÄK RTI works over a WAN. However it provides no additional performance benefits. For WAN communications, distributed forwarding is the recommended strategy.

2.3.5. Distributed Forwarding on a WAN

The MÄK RTI's distributed forwarding option supports both TCP and UDP. The distributed forwarder allows federation designers to create a network of forwarders, each of which manages a subset of the federates in the federation. Distributed forwarding has two main benefits:

- ♦ It spreads the network load across multiple forwarders, rather than putting it all on a single forwarder
- ♦ It achieves optimal network bandwidth use by reducing the number of message copies sent across the WAN from one per federate to one copy, or even none (if you are using sender-side filtering).

To configure distributed forwarding, please see Chapter 11, [Configuring Message Forwarding](#).



Compiling and Linking with the MÄK RTI

This chapter provides details about how to compile and link applications with the MÄK RTI.

Library Names	3-2
Compiling and Linking with the MÄK RTI 1.3	3-2
Compiling and Linking on Linux.....	3-2
Compiling and Linking on Windows	3-3
Compiling and Linking with the MÄK RTI 1516	3-3
Compiling and Linking on UNIX.....	3-4
Compiling and Linking on Windows	3-4
Compiling and Linking with HLA Evolved	3-5
Federation Time.....	3-5
The HLA 1.3 Time Implementation	3-5
The RTI 1516 Time Implementation.....	3-6
Java Bindings for the MÄK RTI.....	3-7

3.1. Library Names

The procedures for compiling and linking applications with the MÄK RTI are essentially the same for each version of the RTI. The main difference is that they have different library names. [Table 3-1](#) lists the libraries used.

Table 3-1: MÄK RTI libraries

	Windows	UNIX
HLA 1.3	<i>libRTI-NG.lib</i> and <i>.dll</i>	<i>libRTI-NG.so</i>
	<i>libfedtime.lib</i> and <i>.dll</i>	<i>libfedtime.so</i>
HLA 1516	<i>librti1516.lib</i> and <i>.dll</i>	<i>librti1516.so</i>
	<i>libfedtime1516.lib</i> and <i>.dll</i>	<i>libfedtime1516.so</i>
HLA Evolved	<i>librti1516e.lib</i> and <i>.dll</i>	<i>librti1516e.so</i>
	<i>libfedtime1516e.lib</i> and <i>.dll</i>	<i>libfedtime1516e.so</i>

3.2. Compiling and Linking with the MÄK RTI 1.3

To build an application that uses the MÄK RTI 1,3:

1. Include *RTI.hh* in any source files that use RTI functionality.
2. Add the *./include/HLA13* and *./lib* directories to the appropriate search paths.
3. Link with the RTI libraries.

3.2.1. Compiling and Linking on Linux

If you have installed the MÄK RTI in, for example, */usr/products/makRtix.x*, the following should appear on your compile line:

```
-I/usr/products/makRtix.x/include/HLA13
```

and the following should appear on your link line:

```
-L/usr/products/makRtix.x/lib -lRTI-NG -lfedtime
```


3.2.2. Compiling and Linking on Windows

Assume that the MÄK RTI is installed in *C:\MAK\makRtix.x*. (The Microsoft Visual C++ user interfaces vary slightly between versions, so the following description is somewhat genericized.)

To compile for HLA 1.3, on the C/C++ section of the Project Property Pages:

1. In the General section, add *C:\MAK\makRtix.x\include\HLA13* to the Additional Include Directories property.
2. In the Preprocessor section, add `RTI_USES_STD_FSTREAM=1` to the Preprocessor Definitions (ostream is in the std namespace).
3. Set `_SECURE_SCL` to 0.
4. Set `_HAS_ITERATOR_DEBUGGING` to 0.
5. For MÄK RTI plug-ins, add *C:\MAK\makRtix.x\include\MAK* to the Additional Include Directories property.
6. In the Code Generation section, in the Run-time Library property, select Multithreaded DLL (/MD) for release builds. Select Multithreaded Debug DLL (/MDd) for debug builds.

To link to the HLA 1.3 libraries, on the Linker section of the Project Property Pages dialog box:

1. In the General section, add *C:\MAK\makRtix.x\lib* to the Additional Library Directories property.
2. In the Input section, on the Additional Dependencies property, for release builds, add the following:

```
libRTI-NG.lib libFedTime.lib ws2_32.lib netapi32.lib
comctl32.lib
```

For debug builds, add the following:

```
libRTI-NGd.lib libFedTimed.lib ws2_32.lib netapi32.lib
comctl32.lib
```

3. In the Input category, on the Ignore Specific Library property, for release builds, add *msvcrt.lib*. For debug builds, add *msvcrt.lib*.

3.3. Compiling and Linking with the MÄK RTI 1516

To build an application that uses the MÄK RTI 1516:

1. Include *RTI1516.h* in any source files that use RTI functionality.
2. Add the MÄK RTI 1516 *./include/HLA1516* and *./lib* directories to the appropriate search paths.
3. Link with the MÄK RTI 1516 libraries.

3.3.1. Compiling and Linking on UNIX

If you have installed the MÄK RTI in, for example, */usr/products/makRtix.x*, the following should appear on your compile line:

```
-I/usr/products/makRtix.x/include/HLA1516
```

and the following should appear on your link line:

```
-L/usr/products/makRtix.x/lib -lrti1516 -lfedtime1516
```

3.3.2. Compiling and Linking on Windows

Assume that the MÄK RTI is installed in *C:\MAK\makRtix.x*. (The Microsoft Visual C++ 7.1 and 8.0 user interfaces vary slightly, so the following description is somewhat genericized.)

To compile for HLA 1516, on the C/C++ section of the Project Property Pages:

1. In the General section, add *C:\MAK\makRtix.x\include\HLA1516* to the Additional Include Directories property.
2. For MÄK RTI plug-ins, add *C:\MAK\makRtix.x\include\MAK* to the Additional Include Directories property.
3. Set `_SECURE_SCL` to 0.
4. Set `_HAS_ITERATOR_DEBUGGING` to 0.
5. In the Code Generation section, in the Run-time Library property, select Multithreaded DLL (/MD) for release builds. Select Multithreaded Debug DLL (/MDd) for debug builds.

To link to the HLA 1516 libraries, on the Linker section of the Project Property Pages dialog box:

1. In the General section, add *C:\MAK\makRtix.x\lib* to the Additional Library Directories property.
2. In the Input section, on the Additional Dependencies property, for release builds, add the following:

```
librti1516.lib libfedtime1516.lib ws2_32.lib netapi32.lib  
comctl32.lib
```

For debug builds, add the following:

```
librti1516d.lib libfedtime1516d.lib ws2_32.lib  
netapi32.lib comctl32.lib
```

3. In the Input category, on the Ignore Specific Library property, for release builds, add *msvcrtd.lib*. For debug builds, add *msvcrtd.lib*.

3.4. Compiling and Linking with HLA Evolved

To compile and link federates with HLA Evolved, follow the instructions for compiling and linking with HLA 1516, except that you use the HLA Evolved directories and libraries instead of the HLA 1516 directories and libraries. (Change 1516 to 1516e and HLA1516 to HLA1516E.)

3.5. Federation Time

According to the HLA specifications, the RTI requires all federations to supply an implementation of time. The RTI API also requires each federate to link with a time library even if it does not use the time management services. The MÄK RTI provides default time implementation libraries to help generate federate executables. The implementations can also be used for federation execution. The RTI API allows a federate to link using any time implementation library and then later execute the federate in a federation using a different time implementation library. If you are not using the time management services it does not matter which library is used during execution.

3.5.1. The HLA 1.3 Time Implementation

The RTI 1.3 API requires that all federates supply an implementation of the logical time abstraction in a library called *libfedtime*. The default implementation provided in the MÄK RTI contains a derivation of RTI::FedTime called *RTIfedTime*, which is defined in *fedtime.hh*.

In addition to the *RTIfedTime* class, the library contains an implementation of the abstraction's destructor RTI::FedTime::~~FedTime and the factory RTI::FedTimeFactory. The factory class is primarily for use by the RTI so that it can create instances of RTI::FedTime and *LogicalTimeInterval*. When a federate uses the time management services, it should use one of the concrete logical time derivations directly. The derivations provide constructors and methods that allow the federate to work with the logical time abstraction using the native C++ type (double).

3.5.2. The RTI 1516 Time Implementation

The RTI 1516 API requires that all federates supply an implementation of the logical time abstraction in a library called *libfedtime1516*. The default logical time library supplied with the MÄK RTI implements the following concrete derivations of the logical time abstraction:

- An implementation based on integer representation.
- An implementation based on floating point representation.
- A deprecated implementation is the default. It duplicates the floating point representation.

The following classes are provided to support the logical time derivations:

- Integer: *LogicalTimeFactoryImplInteger*, *LogicalTimeImplInteger*, and *LogicalTimeIntervalImplInteger*
- Floating point: *LogicalTimeFactoryImplFloat*, *LogicalTimeImplFloat*, and *LogicalTimeIntervalImplFloat*
- Deprecated Floating point: *LogicalTimeFactoryImpl*, *LogicalTimeImpl*, and *LogicalTimeIntervalImpl*.

In addition to these classes, the library contains a *LogicalTimeFactoryFactory*. The factory-factory and factory classes are primarily for use by the RTI so that it can create the appropriate logical time factory and use the factory to create instances of *LogicalTime* and *LogicalTimeInterval*. When a federate uses the time management services, it should use one of the concrete logical time derivations directly. The derivations provide constructors and methods that allow the federate to work with the logical time abstraction using the native C++ types (double and unsigned integer).

The class declarations are contained in files that have the same name as the class and end in *.h*. They are in *./include/HLA1516/RTI*.

3.5.3. The HLA Evolved Time Implementation

The RTI HLA Evolved API requires that all federates supply a library called *libfedtime1516e* which contains the implementation for the *LogicalTimeFactoryFactory* class. The factory-factory and factory classes are primarily for use by the RTI so that it can create the appropriate logical time factory and use the factory to create instances of *LogicalTime* and *LogicalTimeInterval*.

The HLA Evolved API also requires that the RTI provide an implementation of the class *HLAlogicalTimeFactoryFactory*. This factory can be used to create factories for the two standardized time implementations, integer and floating point, which are also supplied by the RTI. The default logical time library supplied with the MAK RTI contains a *LogicalTimeFactoryFactory* that simply forwards all time factory creation requests to the *HLAlogicalTimeFactoryFactory*.

The following classes are defined in the HLA Evolved API to support the logical time derivations:

- Integer: *HLAinteger64TimeFactory*, *HLAinteger64Time*, and *HLAinteger64Interval*
- Floating point: *HLAfloat64TimeFactory*, *HLAfloat64Time*, and *HLAfloat64Interval*

When a federate uses the time management services, it should use one of the concrete logical time derivations directly. The derivations provide constructors and methods that allow the federate to work with the logical time abstraction using the native C++ types (double and unsigned integer).

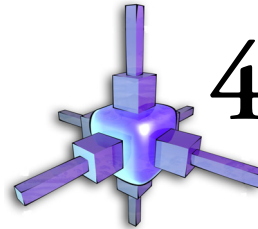
The class declarations are contained in files that have the same name as the class and end in *.h*. They are in *./include/HLA1516E/RTI/time*.

3.6. Java Bindings for the MÄK RTI

The MÄK RTI provides a set of Java bindings for the RTI 1.3-NG API and RTI 1516. The Java binding is a thin layer of C++ code that exposes the native C++ API of the RTI to Java applications.



The MÄK RTI does not include Java bindings for HLA Evolved.



The RID File

This chapter provides general information about editing the RID file. For descriptions of the parameters in the RID file, please see the comments in the file, the sections in this manual that explain how to configure the MÄK RTI, and Appendix A, *RID File Parameters Reference*.

The RID File	4-2
Search Order for RID Files	4-2
Configuring Use of Advisory Messages	4-3
Consistency of RID Files	4-3
Specifying RID Parameters Programmatically in HLA 1516	4-4
Displaying Configuration Settings in the RTI Federations View	4-5
Displaying RTI Configuration Parameters in the RTIspy	4-7
Editing a Parameter	4-8
The RID File Consistency Checker	4-9
Enabling RID File Consistency Checking	4-10
Selecting the Parameters to be Checked	4-11

4.1. The RID File

The RTI initialization data (RID) file (default *rid.mtl*) is the MÄK RTI's configuration file. In the RID file, you can set parameters that control the RTI's networking configuration, enable or disable services, and tune performance. Both the rtiexec and each federate's Local RTI Component read the RID file when they are initialized. For a description of the RID file format and RID file parameters, please see Appendix A, [RID File Parameters Reference](#).

i

- The RID file is not required for basic RTI use. Use it if you want to change the default parameter values and if you want to use plug-ins.
 - The connection configurations for the RTI Assistant override the RID file parameters that specify RTI connection values unless you force the RTI to use the RID file parameters. For details, please see Section 8.3, “[Managing Connection Configurations](#)”, in *MÄK RTI Users Guide*.
-

4.1.1. Search Order for RID Files

The search order for the RID file depends on how you specify it. If you do not specify the RTI_RID_FILE environment variable, the RTI searches for *rid.mtl* as described in this section. If you specify RTI_RID_FILE, you can use a configuration file with a name other than *rid.mtl*. (It must use the same MTL syntax as *rid.mtl*.) In HLA 1516 (but not HLA Evolved), you can override RTI_RID_FILE by passing a wstring to the createRTIambassador call. (For details, please see “[Specifying RID Parameters Programmatically in HLA 1516](#),” on page 4-4.)

i

On Windows, you can use the RTI Chooser to specify the RID file to use with a particular version of the RTI. The RTI Chooser takes care of setting the environment so that the correct RID file is used. For details about using the RTI Chooser, please see Section 8.2, “[Configuring the MÄK RTI Version and RID File to Use](#)”, in *MÄK RTI Users Guide*.

If the RTI_RID_FILE environment variable is not defined, the search order is:

1. Try to open *rid.mtl* in the directory from which you run the application (that is, the working directory, which is not necessarily the directory in which the executable is located).
2. Try to open the file specified by *RTI_CONFIG/rid.mtl*.
3. If a RID file is not found, print a warning and open using default values.

If the `RTI_RID_FILE` environment variable is defined, the search order is:

1. Try to open the file specified in `RTI_RID_FILE`.
2. Try to open the file specified by `RTI_CONFIG/RTI_RID_FILE`.
3. If a RID file is not found, the MÄK RTI stops looking for the RID file and fails.

4.1.2. Configuring HLA Switch Values

HLA 1516 and HLA Evolved both define switches in the FDD file that can be used to control various settings of the federation. You can override the FDD file settings with values defined in the RID file. To do so, you must set `RTI_preferRidToFedSwitchTable` to 1 (the default). If force full compliance is enabled, either through the RTI Assistant or via the RID file, `RTI_preferRidToFedSwitchTable` is disabled and the values in the FDD file are used.

4.1.3. Configuring Use of Advisory Messages

The following parameters specify the default state (enable (1) or disable (0)) for advisory callbacks for classes, attributes, attribute scope, and interactions. Any advisory switches in a 1516 FDD file override these default values. To use any of these parameters, `RTI_preferRidToFedSwitchTable` must be set to 1.

- ♦ `RTI_objectClassRelevanceAdvisorySwitch` (default: enabled)
- ♦ `RTI_attributeRelevanceAdvisorySwitch` (default: disabled)
- ♦ `RTI_interactionRelevanceAdvisorySwitch` (default: enabled)
- ♦ `RTI_attributeScopeAdvisorySwitch` (default: disabled).

4.1.4. Consistency of RID Files

The RID files read by the different federates' LRCs and by the `rtiexec` do not need to be identical. There are many cases where you might want to configure some LRCs differently from others. For example, you might want to have only some LRCs deliver a single callback per tick, or you might want to have different socket buffer sizes from one PC to another. However, some parameters must have the same value for all federates. All RID files must have the same value for `RTI_useRtiExec`. Similarly, all federates in a federation must have the same value for `RTI_udpPort` if they expect to be able to communicate. The tables in Appendix A, *RID File Parameters Reference*, indicate whether a parameter must be identical across the federation or if it can vary from one LRC to another. The RTI can check the consistency of RID files. For details, please see [“The RID File Consistency Checker,”](#) on page 4-9.

4.1.5. Specifying RID Parameters Programmatically in HLA 1516

The HLA 1516 `createRTIambassador` function takes a `vector<wstring>`, which you can use to specify configuration parameters, including the RID file name, as follows:

- If a RID file name is passed to `createRTIambassador`, it overrides any other method of specifying the RID file, such as the `RTI_RID_FILE` environment variable.
- If a list of RID settings is passed to `createRTIambassador`, they are applied after reading the RID file (in effect overriding the values of the RID file). Each string must be a complete lisp expression conforming to the syntax found in the RID file. Only valid lisp expressions will be applied to the RTI environment.

This feature allows individual federates to override individual RID parameters while still specifying use of a global baseline RID file using the `RTI_RID_FILE` or `RTI_CONFIG` environment variables.

The exact syntax is:

```
RTI_RID_FILE pathToRIDFile lisp_expression1 ... lisp_expressionN
```

You can specify additional wstrings that contain lisp expressions to override the settings in the RID file.

If you specify `RTI_RID_FILE` programmatically using `rti1516::createRTIambassador(std::vector<std::wstring>)`, the sort order is:

1. Try to open the file specified by the argument.
2. If the specified file is not found, the MÄK RTI stops looking for the RID file and fails.

For more details about the RID file, please see “[The RID File](#),” on page 4-2.



This feature is not supported in HLA Evolved due to changes to `createRTIambassador`.

4.2. Displaying Configuration Settings in the RTI Federations View

You can display a list of RID parameters and their settings if you are using an rtiexec connection.

To display the RTI's configuration settings:

1. Right-click the RTI Assistant icon in the system tray. A popup menu opens.
2. Choose **Federations View**. The RTI Federations View window opens (Figure 4-1).

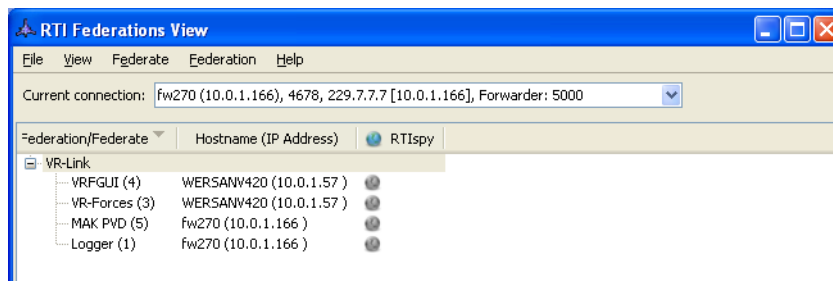


Figure 4-1. RTI Federations View

3. Choose **View** → **RID Parameters**. The RID Settings dialog box opens (Figure 4-2).

The Compliant column indicates compliance of a parameter value, as follows:

- A check mark in the Compliant column means that the current value for that parameter is the correct value for the RTI to be in fully compliant mode.
- An exclamation point in a red circle in the Compliant column means that the parameter is set to a non-compliant value.
- Parameters that do not have a check mark or exclamation point do not affect compliance.

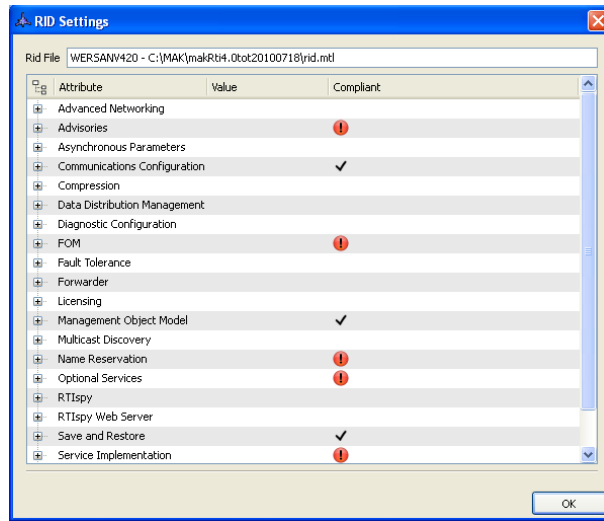
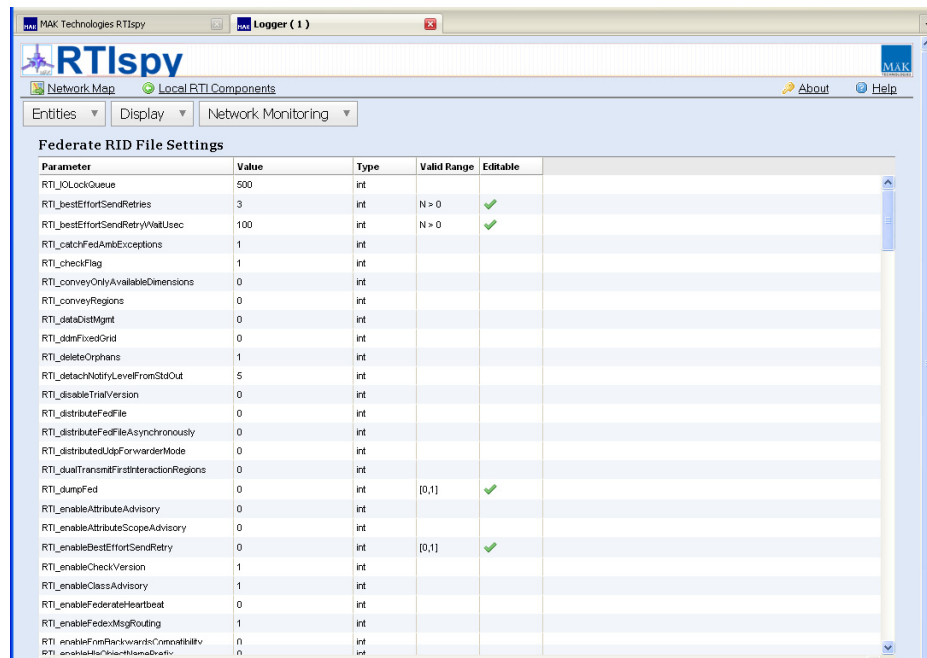


Figure 4-2. RID Settings dialog box

4.3. Displaying RTI Configuration Parameters in the RTIspy

The RTIspy Federate RID File Settings page (Figure 4-3) allows you to view MÄK RTI configuration parameters. You can edit the parameters that are checked in the Editable column.

- To display the RID file settings for a federate, choose **Display** → **RID**.



The screenshot shows the RTIspy application window with the 'Display' menu open and 'RID' selected. The 'Federate RID File Settings' table is displayed with the following parameters:

Parameter	Value	Type	Valid Range	Editable
RTI_JLockQueue	500	int		
RTI_bestEffortSendRetries	3	int	N > 0	✓
RTI_bestEffortSendRetryWaitUseC	100	int	N > 0	✓
RTI_catchFedAmbExceptions	1	int		
RTI_checkFlag	1	int		
RTI_conveyOnlyAvailableDimensions	0	int		
RTI_conveyRegions	0	int		
RTI_dataDistMgmt	0	int		
RTI_dsnFixedGrid	0	int		
RTI_deleteOrphans	1	int		
RTI_detachNotifyLevelFromStdOut	5	int		
RTI_disableTrialVersion	0	int		
RTI_distributeFedFile	0	int		
RTI_distributeFedFileAsynchronously	0	int		
RTI_distributedJspForwarderMode	0	int		
RTI_dsnTransmitFirstInteractionRegions	0	int		
RTI_dumpFed	0	int	[0,1]	✓
RTI_enableAttributeAdvisory	0	int		
RTI_enableAttributeScopeAdvisory	0	int		
RTI_enableBestEffortSendRetry	0	int	[0,1]	✓
RTI_enableCheckVersion	1	int		
RTI_enableClassAdvisory	1	int		
RTI_enableFederateHeartbeat	0	int		
RTI_enableFedexMsgRouting	1	int		
RTI_enableFedexMsgRouting	0	int		
RTI_enableFedexMsgRouting	0	int		

Figure 4-3. RID file settings

4.3.1. Editing a Parameter

You can edit some parameters while a federate is running. Editable parameters have a check mark in the Editable column of the Federate RID File Settings page (Figure 4-3).

To edit a parameter:

1. Choose **Display** → **RID**. The Federate RID File Settings page is displayed.
2. On the Federate RID File Settings page, click the Value for the parameter you want to edit.
3. An edit box is displayed at the bottom of the window (Figure 4-4).

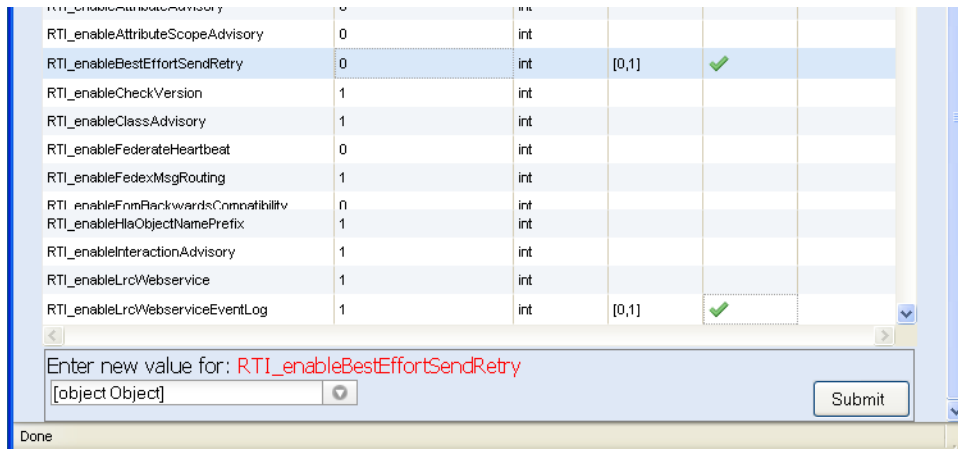


Figure 4-4. Editing a configuration parameter

4. Type or select the value that you want to apply to the parameter.
5. Click Submit.

4.4. The RID File Consistency Checker

The MÄK RTI can provide runtime verification of RID file consistency across all federation executions. The RID file used by the rtiexec acts as a master RID file and all other RID files are compared to it. The RID file consistency checker allows you to guarantee that certain parameters are identical throughout the federation. (As noted in “[Consistency of RID Files](#),” on page 4-3, some parameters can vary among federates within a federation.)

RID file consistency checking has a small cost, as follows:

- ♦ It increases startup time for all federates, because they must pass messages to the RTI to verify their configuration. If this is a concern, federation managers may want to enable consistency checking during integration, and then disable it once all differences have been resolved.
- ♦ Use of consistency checking requires the rtiexec and is therefore unavailable in lightweight mode.



It is strongly recommended that you enable reliable messaging for internal RTI messages (`RTI_internalMsgReliableWhenUsingRtiexec`) when using RID file consistency checking to guarantee that no consistency check messages are lost.

4.4.1. Enabling RID File Consistency Checking

You can enable and disable consistency checking on a per-federate basis. It is disabled by default. If you enable consistency checking, it must be enabled in the RID file for the rtiexec and all federates that want to verify their RID files.

You enable and disable RID consistency checking with the `RTI_ridConsistencyChecking` parameter. `RTI_ridConsistencyChecking` has the following options:

- ♦ 0 — Disable consistency checking.
- ♦ 1 — Enable consistency checking and automatically correct differences in federate-wide parameter settings.
- ♦ 2 — Enable consistency checking and throw a `FederateInternalError` exception if the RID consistency check failed. Setting the `RTI_ridConsistencyChecking` parameter to 2 is useful for federate developers who want to shut down their federates and correct any inconsistencies before joining the federation.



The RID consistency feature relies on the ability to establish a connection between a federate and the RID tester (presumably the rtiexec), however if the federate's `RTI_enableTcpCompression` parameter and the RID tester's parameter are different, the two cannot communicate, and consequently the RID consistency test request and response messages cannot be exchanged. The workaround is to ensure that the `RTI_enableTcpCompression` parameter is consistent among all participants prior to launch.

4.4.2. Selecting the Parameters to be Checked

By default the consistency checker tests a large subset of the parameters that must be consistent across the federation in most federations. The default *rid.mtl* file lists the parameters in the default consistency set.

However, you may want to enforce a stricter checking policy. You can change the consistency set using the `RTI-addRidParametersToOverride` and `RTI-removeRidParametersToOverride` commands in the `rtiexec`'s RID file. Uncomment the command you want to use and list each parameter to be added to or removed from the set.

For example, although `RTI_enableTcpCompression` and `RTI_udpCompressionLevel` must be consistent at start-up and cannot be checked for consistency, the `RTI_tcpCompressionLevel` and `RTI_udpCompressionLevel` parameters can vary within a federation, so they are not checked for consistency by default. Federations that have enabled compression and want to enforce a specific level of compression can add the following to the RID file:

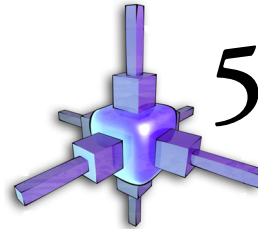
```
(RTI-addRidParametersToOverride
  (list "RTI_tcpCompressionLevel" "RTI_udpCompressionLevel") )
```



The following RID parameters are evaluated before the RID consistency check is performed. Therefore, they cannot have their consistency enforced by the consistency checker. It is your responsibility to ensure that, if required, they are the same in all federates.

- ♦ `RTI_enableBigMessage`
- ♦ `RTI_udpPort`
- ♦ `RTI_enablePacketBundling`
- ♦ `RTI_enableTcpCompression`
- ♦ `RTI_enableUdpCompression`
- ♦ `RTI_internalMsgReliableWhenUsingRtiexec`
- ♦ `RTI_mcastDiscoveryEnabled`
- ♦ `RTI_ridConsistencyChecking`
- ♦ `RTI_tcpForwarderAddr`
- ♦ `RTI_useRtiExec`.

Most of these parameters are federation-wide settings and must be the same in all RID files. However, a few such as `RTI_tcpForwarderAddr` may vary if you use distributed forwarders.



RTI Applications

This chapter describes the rtiexec and applications included with the MÄK RTI that help customers better understand or use the HLA environment.

The rtiexec.....	5-2
Configuring the rtiexec.....	5-3
Configuring Diagnostic Messages.....	5-3
The RTI Assistant.....	5-4
Specifying the RTI Assistant's Log Update Interval.....	5-5
Shutting Down the RTI Assistant.....	5-5
Configuring the RTI Assistant's View Log Prompt.....	5-6
Configuring the Display of RTI Assistant Messages.....	5-7
Common Operations in RTI Assistant Log Windows.....	5-8
Specifying the RTI Assistant's Ports.....	5-9
Specifying the RTI Assistant's Multicast Address.....	5-9
Disabling the RTI Assistant.....	5-9
The RTI Forwarder Application.....	5-10
Running the RTI Forwarder.....	5-10
The RTI Command Line Tool.....	5-12
Listing Federation Executions and Federates.....	5-13
Deleting Federates.....	5-14

5.1. The rtiexec

The rtiexec implements any centralized functionality required by the RTI. For example, the rtiexec keeps track of which federates are joined to each federation execution, assigns federate handles, keeps track of which federates have not yet achieved synchronization points, computes LBTS (Lower Bound Time Stamps) for time management, orchestrates saves and restores, and so on. To use the MÄK RTI in its fully compliant mode, you must run the rtiexec before you start any federates.

You must use the rtiexec if your federation relies on any of the following:

- ♦ Reliable transport (TCP)
- ♦ Time management
- ♦ Synchronization points
- ♦ Federation save and restore
- ♦ The RTIspy GUI network map.

Other RTI services (including MOM, DDM, and ownership management) work in lightweight mode, but in certain cases functionality is limited. For more information, please see Section 7.3.1, “[Limitations of Lightweight Mode](#)”.

If you are not using any of the services that require the rtiexec, then the choice of whether or not to use it is a trade-off between the convenience of lightweight mode, and the full compliance and robustness of running with the rtiexec. In general, you are not giving up performance by using the rtiexec. Object attribute updates and interactions sent with best effort transport always get exchanged directly among the federates without going through the rtiexec. Furthermore, while the rtiexec is often associated with reliable message forwarding, reliable forwarding is a separate performance issue. The RTI can be configured to use best effort transport for internal messages or a separate forwarder process can perform the forwarding of reliable messages. (For more information, please see “[The RTI Forwarder Application](#),” on page 5-10.) Either way, it is not the rtiexec processing that is an issue.

The rtiexec is required for just a few services that are best processed using a centralized approach. For most federations, the only time that the rtiexec’s centralized processing affects a federate is during create, destroy, join, and resign. During most of your federation, it neither sends nor processes any messages.



If you use the rtiexec, internal message reliable mode is recommended, rather than best effort.

5.1.1. Configuring the rtiexec

In addition to the command-line options described in [Table 4-1](#) in *MÄK RTI Users Guide*, you can configure the rtiexec in *rid.mtl*. The MÄK RTI provides several options for configuring the rtiexec. Among the most important options to consider are the use of best effort or reliable transport, and whether or not you will use packet forwarding. For details, please see “[The RTI Forwarder Application](#),” on page 5-10, and “[Choosing Best Effort or Reliable Transport](#),” on page 10-2.

5.1.2. Configuring Diagnostic Messages

The MÄK RTI can provide a variety of diagnostic data to help you debug problems. In addition to the diagnostic resources referenced in this section, you can use the RTIspy web-based GUI to provide additional diagnostic information. For details, please see Chapter 9, [Monitoring the Network](#).

Logging rtiexec Output

You can log rtiexec output to file. The `RTI_rtiExecLogFileName` parameter specifies the name of the log file for the rtiexec process. A unique identifier is inserted in the file name when `RTI_reuseLogFile` is enabled. This parameter is overridden by the `rtiexec -l` option.

Limiting Diagnostic Messages

- To prevent the output of diagnostics, use the `-q` command line option.

5.2. The RTI Assistant

The RTI Assistant is the user interface for the MÄK RTI. It starts automatically when you log in or start a federate (this behavior is configurable) and monitors the status of all RTI-related activity on your computer. It also communicates with RTI Assistants on other computers on the network. The RTI Assistant lets you:

- ♦ View federation information (in the RTI Federations View)
- ♦ View information about your RTI network connections (in the Network Component View)
- ♦ Configure connections
- ♦ Configure some RTI behaviors
- ♦ Configure and run latency testing
- ♦ Resign federates and shut down the rtiexec
- ♦ View federate logs and notification history for federates running on the local machine.

i

The federate-specific features of the RTI Assistant, such as viewing log files and resigning federates, are only available for federates that are running on the local machine.

You can configure the following aspects of the RTI Assistant:

- ♦ How it starts up. For details, please see Section 6.3, “[Configuring the RTI Assistant’s Startup Behavior](#)”, in *MÄK RTI Users Guide*.
- ♦ Notification and log behaviors
- ♦ Whether or not to keep running the RTI Assistant after all other RTI components have shut down. For details, please see Section 6.4, “[Specifying the RTI Assistant’s Exit Policy](#)”, in *MÄK RTI Users Guide*.
- ♦ How to display RTI Assistant messages.

You can clear the default license and connection parameters, and you can set an environment variable to disable the RTI Assistant.

For more general information about the RTI Assistant, please see Chapter 6, [The RTI Assistant](#), in *MÄK RTI Users Guide*.

5.2.1. Specifying the RTI Assistant's Log Update Interval

To specify how frequently the RTI Assistant updates the notification logs:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Preferences**. The RTI Preferences dialog box opens.
3. Select the Notifications and Logs page (Figure 5-1).

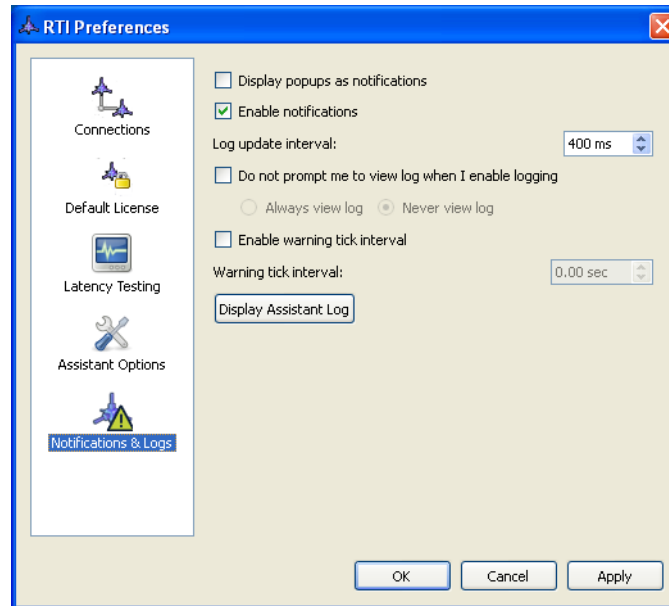


Figure 5-1. RTI Preferences dialog box, Notifications and Logs

4. Enter a value, in milliseconds, in the Log Update Interval text box.
5. Click OK.

5.2.2. Shutting Down the RTI Assistant

If you do not configure the RTI Assistant to shut down automatically, you can shut it down manually. When you shut down the RTI Assistant, it forces all LRCs and rtiexecs on the local computer to shut down. It does not affect RTI components on remote computers.

To shut down the RTI Assistant:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Shut Down All**.

5.2.3. Configuring the RTI Assistant's View Log Prompt

The RTI Assistant lets you enable logging for federates and the rtiexec. You can configure whether or not the RTI Assistant prompts you to view a log file when you enable it.

To configure the view log prompt:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Preferences**. The RTI Preferences dialog box opens.
3. Select the Notifications and Logs page ([Figure 5-1](#)).
4. If you do not want the RTI Assistant to prompt you when you enable logging, select the Do Not Prompt Me to View Log When I Enable Logging check box. If you select this check box, two option buttons become enabled. Configure them as follows:
 - If when you enable logging, you always want to view the log file, select Always View Log.
 - If when you enable logging, you never want to view the log file, select Never View Log.

If you want the RTI Assistant to prompt you to view the log when you enable logging, clear the Do Not Prompt Me to View Log When I Enable Logging check box.

5.2.4. Configuring the Display of RTI Assistant Messages

By default, when an LRC detects a serious problem that is not characterized as a Fatal or Warning message (which would trigger a notification message), it displays a message in a pop-up window in the middle of the screen. You can configure the RTI Assistant to display these messages as notifications (the messages displayed above the RTI Assistant icon).

To configure the display of RTI Assistant messages:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Preferences**. The RTI Preferences dialog box opens.
3. Select the Notifications and Logs page ([Figure 5-1](#)).
4. Select or clear the Display Popups as Notifications check box.
5. Click OK.

5.2.5. Specifying the Warning Tick Interval

The Warning Tick Interval is the interval at which you would like the RTI Assistant to give you a warning if a federate has not ticked. This allows you to identify federates that are doing so much processing that they cannot keep up the necessary tick rate. If the federate goes too long without ticking, the RTI Assistant displays a message indicating this. You can configure the interval to meet the needs of your federates.

To configure the display of RTI Assistant messages:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Preferences**. The RTI Preferences dialog box opens.
3. Select the Notifications and Logs page ([Figure 5-1](#)).
4. Select the Enable Warning Tick Interval check box.
5. Specify the interval time, in seconds.
6. Click OK.

5.2.6. Common Operations in RTI Assistant Log Windows

The RTI Assistant has several windows for displaying messages — the rtiexec Console, the federate log window, the Assistant Log, and the Local Component Notification History window. These windows all allow you to pause and resume display of messages, clear the window, and save the log history using icons that are common to the windows (Figure 5-2).

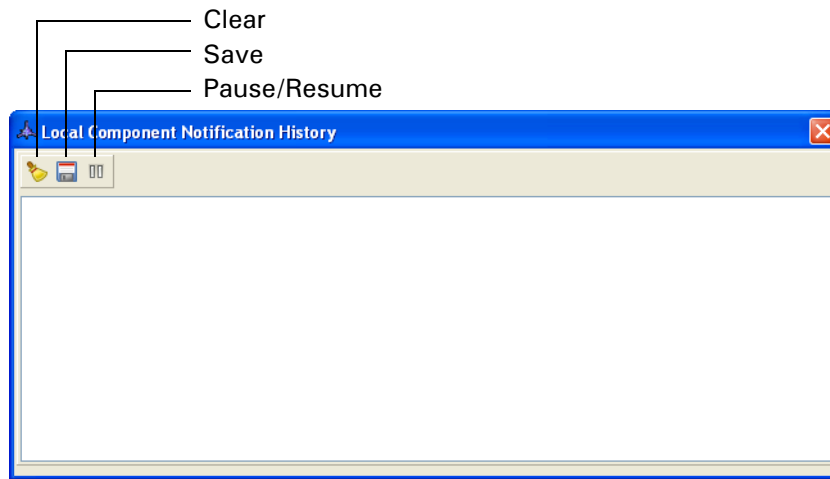


Figure 5-2. Console window



The Assistant Log is primarily for the use of MÄK support engineers when they provide customer support.

Pausing and Resuming the Display of Console Messages

If a log or console window is receiving messages faster than you can read them, you can pause the display. While the display is paused, new messages are saved and they are printed to the console when you resume the display of messages.

- To pause or resume display of console or log messages, click the Pause/Resume icon (Figure 5-2).

Saving Console Output to a File

To save console or log window output to a file:

1. Click the Save icon (Figure 5-2).
2. Type a filename in the Save File dialog box. The RTI saves the current contents of the console.

Clearing the Console Window

- To clear a log or console window, click the Clear icon ([Figure 5-2](#)).

5.2.7. Specifying the RTI Assistant's Ports

The RTI Preferences, Assistant Options page displays the RTI Assistant's peer and client ports. This peer port is used to communicate with local RTI components and with other RTI Assistants. The default port is 6003. You can change the peer port that the RTI Assistant uses.

- To specify the RTI Assistant's port, set the `RTI_ASSISTANT_PORT` environment variable.

Specifying the RTI Assistant's Client Port

Sometimes the RTI Assistant may need to use a different port to communicate with local RTI components than it does to communicate with other RTI Assistants. For example, on Linux machines with multiple users, each user must have their RTI components connect to their own instance of the RTI Assistant. Therefore each user needs to use a different port for their RTI Assistant socket. At the same time, the RTI Assistants must still be able to communicate with each other.

The `RTI_ASSISTANT_CLIENT_PORT` environment variable allows each user to set a port for communication between their local RTI components and the RTI Assistant. RTI Assistants will communicate with each other through the default port or the port specified by `RTI_ASSISTANT_PORT`.

5.2.8. Specifying the RTI Assistant's Multicast Address

You can specify the multicast address that RTI Assistants use to communicate with each other. The default address is 229.7.7.10.

- To specify the RTI Assistant's multicast address, set the `RTI_ASSISTANT_ADDRESS` environment variable.

5.2.9. Disabling the RTI Assistant

- To disable the RTI Assistant, create an environment variable called `RTI_ASSISTANT_DISABLE`. It does not require a value. Its existence causes the RTI to not create the RTI Assistant.

5.3. The RTI Forwarder Application

As explained in [“Reliable Transport and TCP Forwarding on a LAN,”](#) on page 2-7, the MÄK RTI uses TCP forwarding for all reliable communications. This is done using the RTI Forwarder. When you start the rtiexec, if it is configured to use a local RTI Forwarder, it checks to see if one is running. If an RTI Forwarder is running, the rtiexec connects to it. If an RTI Forwarder is not running, the rtiexec starts one using the same connection configuration that the rtiexec uses.

Since the rtiexec usually requires minimal processing time, federations with smaller reliable message loads can easily be supported by centralized TCP forwarding running on the same machine as the rtiexec. As the reliable message load increases, performance can be significantly improved by running the RTI Forwarder (*rtiForwarder.exe*) on a separate machine.

You may decide to run multiple RTI Forwarders to implement a distributed forwarding scheme. For details, please see Chapter 11, [Configuring Message Forwarding](#).

5.3.1. Running the RTI Forwarder

To run the RTI Forwarder:

1. Run:

```
./bin/rtiForwarder.exe options
```

Unless you specified the `-M` command-line option, or `RTI_configureConnectionWithRid` is set to 1, the Choose Connection Configuration dialog box opens.

2. Select a connection configuration. (If you are creating a distributed forwarder network and have not previously created connection configurations that specify an address for the forwarder that you want to connect to, edit the connection configuration to add the forwarder address. For more information, please see [“Configuring RTI Forwarders Using the RTI Assistant,”](#) on page 11-17.)
3. Click Connect.

[Table 5-1](#) describes the command-line options for the RTI Forwarder.

Table 5-1: RTI Forwarder command-line options

Option	Description
(<code>--</code> <code>--ignore_rest</code>)	Ignore all command-line options that follow this one.
(<code>-A</code> <code>--destAddrString</code>) <i>address</i>	Specifies the multicast group address to use if reliable transport is disabled. Equivalent to the <code>RTI_destAddrString</code> parameter.
(<code>-d</code> <code>--doNotUseRoutes-</code> <code>File</code>)	Specifies that the forwarder not use a routes file.
(<code>-D</code> <code>--distributedForward-</code> <code>erPort</code>) <i>port</i>	Specifies the port that RTI Forwarders use to communicate with each other.

Table 5-1: RTI Forwarder command-line options

Option	Description
(-F --forwarderToConnectTo) <i>address</i>	Specifies the address of an additional forwarder that this forwarder should try to connect to instead of using the routes file.
(-h --help)	Displays a list of command-line options.
(-l --setLogFileName) <i>filename</i>	Enables logging to a file with the specified name.
(-M --manual)	Specifies that the RTI Forwarder use the connection configuration in the RID file or supplied on the command line rather than the configuration stored by the RTI Assistant. It is equivalent to setting <code>RTI_configureConnectionWithRid</code> to 1. Since command-line options override the RID file, using this option enables you to specify the connection configuration from the command line.
(-n --notifyLevel) <i>level</i>	Sets the diagnostic output level.
(-N --networkInterfaceAddr) <i>address</i>	Specifies the IP address of the network card that you want this forwarder to use.
(-P --setUdpPort) <i>port</i>	Specifies the port to listen to. Equivalent to the <code>RTI_udpPort</code> parameter.
(-R --setRidFileName) <i>RID_file</i>	Specifies the RID file to use. <code>-R</code> overrides the <code>RTI_RID_FILE</code> environment variable.
(-r --routesFileName) <i>routeFileName</i>	Specifies the forwarder route file to use. Overrides the <code>RTI_forwarderRoutesFile</code> parameter in the RID file.
(-v --version)	Display version information and exit.

5.4. The RTI Command Line Tool

You can send commands to the RTI from a console window using the RTI command line tool. This tool is not interactive.

To use the RTI command line tool:

1. Open a console window.
2. Change directory to *./bin*.
3. Enter a command in the following format:

```
rti option
```

[Table 5-2](#) lists the commands that you can use with RTI command line tool.

Table 5-2: rtiexec runtime commands

Command	Inputs	Description
delete	<i>componentHandle</i> <i>federationHandle</i> <i>federateHandle</i>	Resigns the federate with the given <i>componentHandle</i> , <i>federationHandle</i> , and <i>federateHandle</i> from the rtiexec. Can be used with local and remote federates. To get handles, call: <pre>rti list rtiexec</pre> The <i>componentHandle</i> is the handle the RTI Assistant uses to designate the different local components connected to it. In the results from the list command, it is the number displayed at the beginning of the line.
list	[all rtiexec rtiForwarder]	Lists components based on the input parameter, as follows: ♦ all (or no input). All of the local RTI components are listed. ♦ rtiexec. All federations and federates of the local rtiexecs are listed. ♦ rtiForwarder. All connections of the local RTI Forwarders are listed.
kill	<i>componentHandle</i>	Resigns local federates and shuts down local rtiexecs and RTI Forwarders. To get handles, call: <pre>rti list</pre> The <i>componentHandle</i> is the number displayed at the beginning of the line.
shutdownall	None.	Shuts down the local RTI Assistant and all connected local RTI components.

The following example uses the list command to generate information about components. Then it deletes a federate.

```
C:\source\makRti\bin>rti list rtiexec
Information from local rtiexec Connections
  1) dubois370 (192.168.3.226), 4226, 229.7.7.7, fully compliant
    [192.168.3.226]
  4) MAKsimple
    2) rtisimple1516 (dubois370)

C:\source\makRti\bin>rti delete 1 4 2
Delete federate Component Command sent to rtiexec
```

This command tells the rtiexec with LRC handle 1, to delete the federate with federate handle 2, in the federation with federation handle 4.

5.4.1. Listing Federation Executions and Federates

To list all running federation executions and their federates, use the list command. Output from the rti list command might look like the following:

```
C:\MAK\makRti3.4\bin>rti list
Printing Local Components

Locally Connected rtiexecs by Assistant ID
  0) fw270 (10.0.1.166), 4678, 229.7.7.7, fully compliant [10.0.1.166]

Locally Connected Forwarders by Assistant ID
  1) fw270 (10.0.1.166), 4678, 229.7.7.7, fully compliant [10.0.1.166]

Locally Connected LRCs by Assistant ID
  2) Logger (1)
  3) MAK PVD (2)
  4) VR-Link F18 (3)
```

Output from rti list rtiexec would look like the following:

```
C:\MAK\makRti3.4\bin>rti list rtiexec
Information from local rtiexec Connections
  0) fw270 (10.0.1.166), 4678, 229.7.7.7, fully compliant [10.0.1.166]
  2) VR-Link
    1) Logger (fw270)
    2) MAK PVD (fw270)
    3) VR-Link F18 (fw270)
```

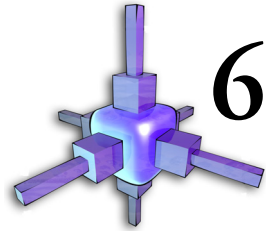
5.4.2. Deleting Federates

The `delete` command deletes the specified federate by handle and federation execution name or deletes all federates with the specified handle. You can use `delete` to explicitly resign a federate. This is useful if a federate crashes before it resigns properly. For example, assuming the output from the `list rtiexec` command in [“Listing Federation Executions and Federates,”](#) on page 5-13, to delete the Logger federate from the VR-Link federation execution, you would enter the command:

```
kill 1 Logger
```

To delete all federates with the handle 1 from all federation executions, enter the command:

```
kill 1
```

Setting Up the RTIsPy

The RTIsPy allows centralized monitoring of all components of the RTI. You can monitor the rtiexec or any LRC from a single machine. You can also monitor multiple RTI components simultaneously for further analysis and side-by-side comparisons.

Installing and Configuring the RTIsPy	6-2
Configuring the RTIsPy	6-2
Running the RTIsPy Web Servers	6-4
Opening the RTIsPy Web Page.....	6-4
URL Structure.....	6-5
Choosing Between Local and Centralized HTTP Servers	6-6
Connecting to the RTIsPy Web Servers.....	6-8
Bookmarking.....	6-9
Pop-up Blockers	6-9
Minimizing Overhead	6-9
Performance Issues for the RTIsPy	6-10

6.1. Installing and Configuring the RTIspy

The RTIspy and HTTP Server are installed with the MÄK RTI. Web monitoring is disabled in the default RID file configuration.

6.1.1. Configuring the RTIspy

You can run the RTIspy without running the rtiexec. However, if you want to use the RTIspy to view federation information, or if you want to use the network map, you must run the rtiexec.

To enable the rtiexec and RTIspy, the minimal configuration settings for each federate are as follows:

```
(setqb RTI_enableRtiexecWebservice 1)
(setqb RTI_enableLrcWebservice 1)
```



You must enable `RTI_internalMsgReliableWhenUsingRtiexec` to access the network map (described in “[Viewing Network Components](#),” on page 7-7). If the network map is not available, you can access RTI components through the component list (described in “[Viewing the MÄK RTI Components List](#),” on page 7-10).

[Table 6-1](#) lists the parameters that enable and configure the RTIspy.

Table 6-1: RTIspy configuration parameters (Sheet 1 of 3)

Parameter	Description
<code>RTI_enableRtiexecWebservice</code>	Enables (1) or disables (0) RTIspy for the rtiexec. Default: 0.
<code>RTI_enableLrcWebservice</code>	Enables (1) or disables (0) the LRC RTIspy services. This is set per LRC.
<code>RTI_pluginDirectory</code>	Specifies the RTI plug-in directory. The RTIspy directory structure is fixed relative to the plug-in directory specified by the <code>RTI_pluginDirectory</code> parameter. The <code>www</code> sub-directory must be present within the chosen plug-in directory for the RTIspy to function. If you change the RTI plug-in directory, you must copy the entire <code>www</code> directory from the installed <i>plugins</i> directory to the new location. Default: “ <code>../plugins</code> ”.

Table 6-1: RTIsPy configuration parameters (Sheet 2 of 3)

Parameter	Description
RTI_enableLrcWebserviceEventLog	Enables (1) or disables (0) the LRC Event Log in the RTIsPy. The event log can be very useful in tracking down initial startup problems with misbehaving federates. However, over time, as the number of logged events grows, the event log may consume significant memory at the federate. The event log grows even when a federate is not actively monitored. Therefore, the log is disabled by default. Normally, it should only be enabled at specific federates for debugging and during federation testing. Default: 0.
RTI_webserviceHttpPort	Specifies the port for the HTTP interface to the RTIsPy. You must point your browser to the specific machine running the HTTP server and this specific port (using the syntax: <code>http://serverAddress:RTI_webserviceHttpPort/</code>). Default: 8008.
RTI_webserviceRtiPort	Specifies the port which each RTI component will use to connect to the RTIsPy Web service HTTP server. The RTI components must be allowed to open a TCP connection to their HTTP server to enable monitoring via the RTIsPy. Unlike the <code>RTI_webserviceHttpPort</code> , you do not need to be aware of this port in order to view the RTIsPy, since it is not part of the URL passed to the browser. Default: 4002.
RTI_webserviceAddr	Allows you to instruct each RTI component to attach to a specific HTTP server. By default, an HTTP server runs at each host and federates running on that host should connect to that HTTP server. In general, the default provides the expected behavior and this parameter rarely needs to be changed. (For details, please see “Choosing Between Local and Centralized HTTP Servers,” on page 6-6.) Default: "127.0.0.1".
RTI_webserviceEnableServerAutoStart	Instructs the RTI components to try to start an HTTP server if none is running when they are started. This is usually the desired behavior, but when using a centralized HTTP server it is unnecessary. (For details, please see “Choosing Between Local and Centralized HTTP Servers,” on page 6-6.) Default: 1.

Table 6-1: RTIsPy configuration parameters (Sheet 3 of 3)

Parameter	Description
RTI_webserviceEnableForwarding	Enables (1) or disables (0) connection proxying through the rtiexec web services. The only reason to disable this parameter is to explicitly disallow proxy connections. (For details, please see “Connecting to the RTIsPy Web Servers,” on page 6-8.) Default: 1.
RTI_webserviceNotifyLevel	Specifies the debug output level at the HTTP server. By default, the HTTP server is mostly silent. Increasing the notify level to 2 allows you to monitor connections to the HTTP server for debugging purposes. Normally, this parameter should only be raised above 2 when instructed to do so by VT MÄK technical support. (Due to the nature of the HTTP server, the output is far less useful to end users than the output from the LRC or rtiexec.) Default: 1.

6.2. Running the RTIsPy Web Servers

If the MÄK RTI has been installed and the RTIsPy has been enabled and configured as described in [“Installing and Configuring the RTIsPy,”](#) on page 6-2, the web service should start automatically whenever you start a federate or the rtiexec.

6.2.1. Opening the RTIsPy Web Page

To open the RTIsPy in your browser, you can enter a URL as described in the following sections, or you can open it from the RTI Federations View or RTI Assistant menu.

To open the RTIsPy from the RTI Federations View:

- To open the RTIsPy to the network map, choose **View** → **Open RTIsPy**.
- To open the RTIsPy to a federate page, select the federate and choose **Federate** → **Open Federate RTIsPy**.

Opening the RTIsPy from the RTI Assistant Menu

To open the RTIsPy from the RTI Assistant:

1. Right-click the RTI Assistant icon. A menu is displayed.
2. Choose the federate for which you want to open an RTIsPy page. A submenu is displayed.
3. On the submenu, choose **Open RTIsPy**.

6.2.2. URL Structure

The RTIsPy Web Server provides a component web page. The component web page will either be a graphical network map of all the RTI components or a list of the components connected to the web server. To open the component web page, use a URL formatted as:

```
http://Host:Port/
```

where:

- ♦ *Host* is the IP address or hostname of the Web Server host machine. This is usually where an RTI component is executing, unless a remote or central server is being used. (For more information, please see [“Choosing Between Local and Centralized HTTP Servers,”](#) on page 6-6.)
- ♦ *Port* is the `RTI_webServiceHttpPort` setting in the RID file.



The network map page is supported when `RTI_internalMsgReliableWhenUsingRtiexec` is enabled; otherwise, a component list is displayed.

It is also possible to open a page directly to a specific component. To access a specific component, use a URL formatted as:

```
http://Host:Port/?id=ConnectionID
```

where:

- ♦ *Host* and *Port* are the same as previously described.
- ♦ *ConnectionId* is the RTI connection ID of the RTI target federate LRC, or the text `rtiexec`, to connect directly to the `rtiexec`.

For example, if you want to use the default parameters to connect to a server running on the same machine as the browser, use one of the following URLs:

- ♦ Use `http://localhost:8008/` to view the network map or local component list



The trailing slash (/) at the end of the URL is required.

- ♦ Use `http://localhost:8008/?id=rtiexec` to view the `rtiexec` (if it is directly connected to this server instance)
- ♦ Use `http://localhost:8008/?id=2` to view the LRC with connection ID 2 (if it is directly connected to this server instance).

If you select port 80 for the `RTI_webServiceHttpPort`, you can omit the `:port` portion of the URL when accessing the RTIsPy. However, many hosts already have a web server running on port 80, so it is recommended that you select a different port for the RTIsPy web service.

6.2.3. Choosing Between Local and Centralized HTTP Servers

The RTIsPy web services can run using a central server or they can run with servers located on each computer that has an LRC running on it.

The RTIsPy is designed to run as a lightweight web service hosted by the same machine that runs the RTI component to be viewed (local server). By default, each RTI component (`rtiexec` and `LRC`) can start a web service HTTP server to host its own web services. In this configuration, you can simply point your web browser to the machine running the RTI component to load the RTIsPy for that component. The web service HTTP server is a very lightweight process and the additional overhead should be negligible for most hosts.

There are some major advantages to running the web service locally, because it allows direct connection rather than proxied connection (as described in [“Connecting to the RTIsPy Web Servers,”](#) on page 6-8). However, in some cases (for example to limit the number of machines with open HTTP ports), you may want to run the RTIsPy web service HTTP server on a central host and the individual RTI components can be instructed to connect to the central server.

Starting the RTIsPy Web Service Automatically (Local Server)

By default, when connecting to a local HTTP server, each RTI component attempts to start the HTTP server process automatically if it is not already running. In this configuration, `RTI_webServiceEnableServerAutoStart` is set to 1 and `RTI_webServiceAddr` is set to the local loopback address (127.0.0.1).

To disable auto-start set the `RTI_webServiceEnableServerAutoStart` parameter to 0. HTTP server auto-start is automatically disabled if `RTI_webServiceAddr` is set to a remote host address.

Running the Web Service in Stand-Alone Mode (Central Server)

If the web service auto-start feature is disabled, or to run the web service HTTP server at a central host, you can run the `wwwSpyServer` application in stand-alone mode.



The web service requires access to both the RID file and the *plugins* directory from the MÄK RTI installation, so it is recommended that the MÄK RTI be installed on any machine that will be used to host the web service.

To run the server in standalone mode:

1. In a console window, change to the `./bin` directory.
2. Run the `wwwSpyServer` application:

```
wwwSpyServer [options]
```

- To shut down the web service, enter `q` in its console.

Table 6-2 describes the `wwwSpyServer` command-line options.

Table 6-2: `wwwSpyServer` command-line options

Option	Description
<code>(-D --runDaemonized)</code>	Run <code>ShmExec</code> in the background. (Linux only)
<code>(-h --help)</code>	Displays a syntax help screen.
<code>(-K --AutoExit)</code>	Run the web server and automatically exit after the last subscriber disconnects.
<code>(-l --enableLogging)</code>	Enable RTIsPy Web Server logging to <i>makWebSpy.log</i> .
<code>(-n --notifyLevel) level</code>	Sets the verbosity of the web server's debug output. It is recommended that this parameter be set to 3 or less. Set it to 0 to run the web server silently. The default value is 1. It overrides the parameter configured in the RID file. <i>level</i> can be: 0 — Fatal 1 — Warn (default) 2 — Notify 3 — Verbose 4 — Debug.
<code>(-t --testPage)</code>	Show a test demonstration.
<code>(-v --version)</code>	Displays version information and exits.
<code>(-- --ignore_rest)</code>	Ignore the command-line options that follow this option.

6.3. Connecting to the RTIsPy Web Servers

The RTIsPy allows web clients to connect in two ways, depending upon their configuration and upon the configuration of the RTI network: direct connection and proxy connection.

The preferred connection method is direct connection. In a direct connection, the information for the selected federate is served directly from the web service running on that federate's host. You can access a direct connection through the network map, the component list, specifying a direct URL, or through the rtiexec page.

However, there are cases in which clients do not have direct access to the target host machine. For example in a WAN environment, each LAN may use a firewall that performs network address translation (NAT) and blocks all ports except the port specified by `RTI_udpPort`. In that case, federates on LAN A cannot connect to the web servers running behind LAN B's firewall (not to mention that the IP addresses of machines in LAN B might duplicate the local IP addresses in LAN A if NAT is in use).

In such cases, the direct connection mode is unavailable for some hosts. You must use the proxy connection method. In the proxy connection method, the web client connects specifically to the rtiexec host (assuming that the rtiexec's web service is running on the same machine), and makes all requests to the rtiexec web service. The rtiexec is then responsible for proxying those requests through the RTI network to the individual LRCs and returning the response data to the client. In other words, the HTTP request always goes to the rtiexec's web service, and the rtiexec is responsible for collecting and returning the response data from the desired RTI component.

Since the federation manager has presumably already dealt with all of the connection and firewall issues involved in setting up the RTI network across the WAN, the rtiexec can communicate directly with any LRC, even though the web browser cannot. Therefore, in many WAN environments you must use proxy connections to view remote federates. However, proxy connections involve more overhead than direct connections, because the web browser's request must travel to the HTTP server, then the rtiexec, then the LRC, and the response must return via the same route.



Proxy connections are only available through the web-based rtiexec. The rtiexec must be brought up first, and then a proxy connection to an LRC page can be made.

In general, direct connection is preferable to proxy connection because the overhead is lower and the effect on the federation is minimized. Proxy connections should be used only to avoid firewalls and other network barriers.

6.3.1. Bookmarking

When you connect directly to an RTI component, you can create a browser bookmark to return quickly to the same component. To create a bookmark, format the URL as described in “[URL Structure](#),” on page 6-5. However, it is important to note that the URL for a specific LRC is dynamic, since the ID field is based on the RTI's assigned connection ID. If the target federation tends to start up in a predictable order, then connection IDs are likely to be assigned in a predictable manner, but this is certainly not guaranteed. The only guaranteed URL is the rtiexec's:
`http://localhost:8008/?id=rtiexec.`

6.3.2. Pop-up Blockers

If your browser has a pop-up blocker enabled, it might stop new windows from opening. The RTIsPy opens new windows for the About and Help buttons, but more importantly for the View Directly button. Therefore, you should enable pop-ups from the rtiexec's URL. For more information, please see the web browser or pop-up blocker's documentation for more information.

6.3.3. Minimizing Overhead

The RTIsPy web service adds a small amount of overhead to the operation of the federation. In federations for which even a small amount of additional overhead may be unacceptable, you should disable the web services. However, the HTTP server itself is very lightweight, so simply enabling the web services should not affect any but the most demanding federations.

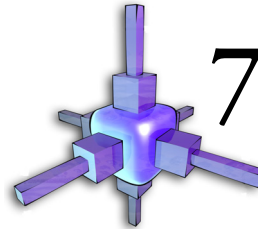
For most federations, the most important consideration regarding performance of the RTIsPy is the number of RTI components under active monitoring. After startup, no additional bandwidth and very little CPU time is consumed when RTI components are not actively monitored. While a web browser window is open to the RTI component's page, that component periodically generates and sends state updates to the client. In proxied connections, these updates also pass through the rtiexec – adding an additional hop and some additional overhead. Therefore the overhead increases with each additional RTI component under simultaneous monitoring.

When the browser window is closed, or the proxy user clicks the Return to RTI Exec button, the RTI component stops sending updates and the overhead returns to a nominal amount. Therefore, to minimize overhead, be sure that all browser windows are closed when the client is done monitoring the RTI component.

6.4. Performance Issues for the RTIsPy

Because remote monitoring requires data to be sent across the network, the RTIsPy introduces some overhead. The state of the monitored LRC must be sent to the web server, then to your browser. This creates additional network load (usually on the same physical network as the federation execution). The load increases if you monitor multiple components simultaneously (the data for all components must be updated periodically). Thus, the RTIsPy will probably be most useful as a testing and verification tool, but it can be invaluable to federation managers during federate integration testing.

Despite these concerns, the cost of using the RTIsPy is not as great as you might expect. In particular, state data is only sent when an RTI component is under active monitoring and the only data sent is that required for the specific elements being monitored. Therefore, when you switch to monitoring a different component or close the browser connection, the network overhead of the RTIsPy is eliminated (there is some local book-keeping required to maintain a state database at each component). In federations that do not have limited bandwidth (this is the case for most federations), you can use the RTIsPy to perform central monitoring throughout the federation's lifetime. In particular, federation managers may want to monitor the federation state as a whole and only connect to individual LRCs to diagnose problems.



Managing Federations

This chapter explains how to view information about and configure federation executions.

Introduction	7-3
Specifying the FED File	7-3
Search Order for FED Files	7-4
Distributing the FED File	7-4
Using Lightweight Mode	7-5
Limitations of Lightweight Mode	7-6
Viewing Network Components.....	7-7
The RTIspy Network Map	7-7
Viewing Network Components	7-7
Viewing the MÄK RTI Components List.....	7-10
Viewing Information about a Federation.....	7-11
Viewing Federation Information in the RTIspy	7-12
Viewing Time Management State Information	7-13
Viewing Time Management State Information in the RTIspy	7-15
Viewing Synchronization Points.....	7-16
Viewing Federation-Wide Notification Messages	7-17
Configuring Fault Tolerance Strategies	7-18
Responding to Abnormal Termination	7-18
Reconnecting to the RTI Forwarder	7-20
Specifying the Response Interval.....	7-20
Processing Discovery of Unknown Objects.....	7-21
Choosing Silent Failure Instead of Exceptions	7-21
Configuring Save/Restore.....	7-22

Managing Federations

Viewing Notification Messages from the rtiexec.....	7-22
Displaying Recent Log Entries	7-22

7.1. Introduction

The MÄK RTI has the following graphical user interfaces (GUIs) for viewing information about federations and federates:

- ♦ RTI Federations View
- ♦ RTI Network Component View
- ♦ RTIspy web-based GUI.

This chapter covers high-level federation management issues and explains how to use the RTI Assistant and RTIspy GUIs to help manage federations.

7.2. Specifying the FED File

When you use the `rtiexec`, the federate that successfully creates the federation execution dictates the name of the FED file for all federates to use by passing it to `createFederationExecution()`. You can configure the RTI so that when a federate joins the federation, the RTI exchanges either just the FED file name or the entire FED file contents between the `rtiexec` and the federate's LRC.

Exchanging just the FED file name is very efficient, but each federate's LRC must read the FED file locally and there may be consistency issues among federates. To prevent problems caused by inconsistent FED files, when the federation is created, the LRC of the creating federate sends the CRC (cyclic redundancy check) of the FED file to the `rtiexec`. Thereafter, as federates join the federation, each LRC sends the CRC of the local FED file. If `RTI_crcCheckFedFile` is set to 0, and the federate's CRC does not match the CRC of the FED file used to create the federation, the joining federate is notified that the FED files are different and it joins the federation. If `RTI_crcCheckFedFile` is set to 1 and the CRCs do not match, the federate cannot join the federation.

Exchanging the entire FED file ensures that each federate uses the same FED file, but introduces some overhead. (For details, please see “[Distributing the FED File](#),” on page 7-4.)



In full compliance mode, `RTI_distributeFedFile` is enabled.

In lightweight mode, the name of the FED file is not communicated among federates. If you want the federates to use a FED file other than the default (the federation execution name with `.fed` appended to the end), then either use the RID parameter `RTI_defaultFedFile` to specify the name of the FED file to read, or at each federate use the `createFederationExecution` service prior to the `joinFederationService` to create a mapping between the federation name and the FED file name.

7.2.1. Search Order for FED Files

Regardless of how the name of the FED file is specified, the MÄK RTI looks for the file in the current working directory first. Then it looks in the directory specified by the RTI_CONFIG environment variable. If the RTI cannot find the appropriate FED file in either of these locations, the federate cannot join.

7.2.2. Distributing the FED File

You can distribute the FED file from the federate that creates the federation to all joining federates instead of reading a local copy of the file at each federate. When the first federate creates the federation, it passes the FED file to the rtiexec. As additional federates join, the rtiexec passes the file to them along with their join confirmation.

To enable distribution of the FED file, set the `RTI_distributeFedFile`, `RTI_useRtiExec`, and `RTI_internalMsgReliableWhenUsingRtiexec` parameters to 1.



In full compliance mode, `RTI_distributeFedFile` is enabled.

The primary reasons to enable FED file distribution are to ensure that the FOM used by each federate is consistent while you are developing a federation or to support FOM modules. During federation development, the FOM may be in a state of flux. If different federates try to join the federation using different revisions of the FOM, unexpected and difficult-to-diagnose problems may arise.

However, distributing the FED file is a relatively expensive network operation, which may slow the join process and, in the case of late joining federates, may even impede the performance of a running federation. Therefore, unless you are using FOM modules, we recommend that you only use FED file distribution in the federation development phase, and use only statically distributed FED files when the simulation is deployed. (If you use the FOM module feature, you must distribute the FED file. For details, please see [“Using FOM Modules,”](#) on page 15-2.)



Any federate that attempts to create a federation must have a local copy of the FED file, because the `createFederationExecution` API routine requires a valid FED file.

Distributing an HLA 1516 FDD File

In 1516, the HLA DTD is not distributed with the FDD file. In order for the LRC to process the FDD file, the HLA DTD must be accessible as specified in the FDD DTD attribute. The DTD is found relative to the federate's working directory.

7.3. Using Lightweight Mode

The MÄK RTI supports a lightweight mode in which the `rtiexec` is not necessary. In general, lightweight mode is appropriate for small, simple federations that do not use the more complex RTI services, that require only best effort transport, and that do not require strict compliance with the HLA Specification.

Many MÄK RTI users use lightweight mode during development and debugging of their federates because of the convenience of being able to quickly and repeatedly restart federations without worrying about running the `rtiexec`. However, many switch to using the `rtiexec` when playing in real exercises so that they can gain strict compliance with the HLA specification, use reliable transport, and take advantage of the fault tolerance and guarantee of federate handle uniqueness that is supported only with the `rtiexec`.

When you run in lightweight mode, you can use the RTI Assistant to:

- ♦ View the notification history
- ♦ Enable federate logging
- ♦ Force a federate to resign
- ♦ Run latency tests
- ♦ Shut down all federates.



The federation management features of the RTI Assistant and the RTIsPy web-based GUI are not available in lightweight mode.

7.3.1. Limitations of Lightweight Mode

In lightweight mode, services that require the central processing provided by the rtiexec (reliable transport, time management, save and restore, and synchronization points) are not available. Other RTI services, (including MOM, DDM, and ownership management) work in lightweight mode, but in certain cases functionality is limited. In lightweight mode:

- ♦ Released attributes are permanently orphaned. (When a federate resigns from a federation with a resign action `RELEASE_ATTRIBUTES DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES`, the RTI is supposed to allow other federates to acquire ownership of attribute instances that the resigning federate is releasing.)
- ♦ There is no central place to store information about which federation executions exist and which federates are joined to them. Therefore:
 - The `createFederationExecution()` service returns successfully even if the named execution has already been created
 - The `destroyFederationExecution()` service always returns successfully even if the named execution does not currently exist
 - The `joinFederationExecution()` service returns successfully even if the named federation execution has never been created using `createFederationExecution()`.
- ♦ Each federate unilaterally chooses its federate ID without asking the rtiexec to make sure it is unique. This makes it possible for two federates to have the same federate ID. (If this occurs, you will need to restart your federation.) The chances of two federates choosing the same ID depend on the strategy the federate uses for choosing its ID. By default, each federate uses its process ID modulo 10,000 as its federate ID. But if `RTI_useRandomNumberForFedHandle` is set to 1 in your RID file, each federate chooses a handle at random.
- ♦ The first three characters plus the last two characters of the federation execution name are used to construct a federation execution identifier. If you have more than one federation execution running at the same time on the same port or address, their names must differ within the first four characters or the last character
- ♦ The RTI cannot detect crashed federates and automatically resign them from the federation execution.
- ♦ The RTI does not pass the name of the FED file from the federate that created the federation execution to the other federates. If your FED file name is something other than the name of your federation execution followed by *.fed*, you must call `createFederationExecution()` in every federate, passing the name of the appropriate FED file.
- ♦ When using RTI 1516 or HLA Evolved, name reservation is only checked against what is known at the local RTI component (previously reserved names, registered objects, and discovered objects).
- ♦ FOM module merging is not supported.

7.4. Viewing Network Components

The MÄK RTI offers two network-centric (as opposed to federation-centric) views of the applications running on the RTI network – the RTI Network Component View and the RTIspy network map. The RTIspy also has a simple component list that you can use to access federation and federate information. These views let you see what applications are running without regard to their participation in a particular federation. This type of view may be particularly helpful when you set up a distributed forwarding network, as described in Chapter 11, *Configuring Message Forwarding*, and you want a way to see which federates are connected to which RTI Forwarders to determine if they are interacting as designed.

7.4.1. The RTIspy Network Map

The RTIspy can display a list of components (the MÄK RTI Components list) or a network map, which displays the RTI components graphically (Figure 7-1). (For details about the MÄK RTI Components list, please see “[Viewing the MÄK RTI Components List](#),” on page 7-10.)

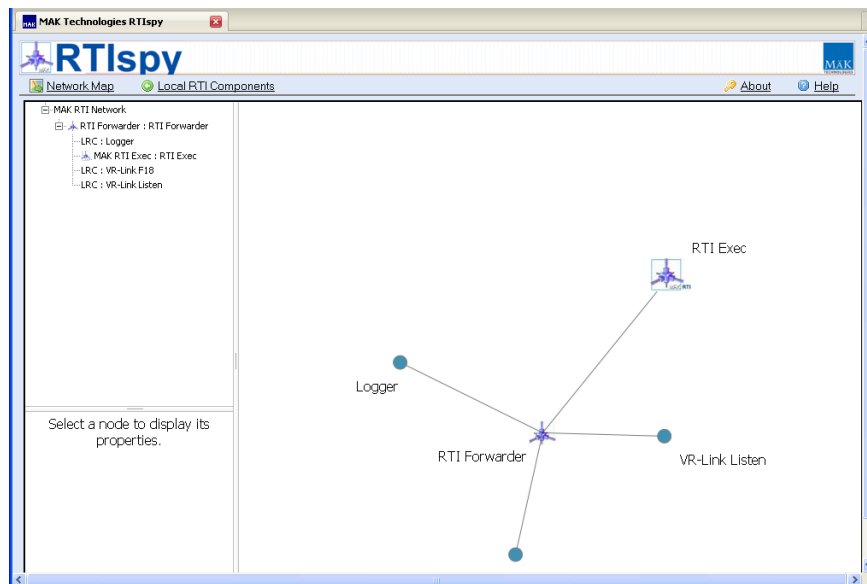


Figure 7-1. Network map, no components selected

The network map displays an icon for each forwarder, each federate, and the rtiexec node. The links between components represent the reliable connections (TCP) between them. (Components can communicate through other connectionless methods that are not represented in the network map (UDP).)

The upper left frame displays a tree list of components or nodes. The lower left frame displays information for a selected node.

The tree list is rooted with a notional RTI network node. Nested under the RTI network are the RTI Forwarders. Nested under each forwarder are the federates connected to that forwarder. The rtiexec is nested under one of the forwarders. You can expand and collapse the tree.

Selecting a node in the map highlights the tree list node and all of its connections in the network map. Selecting a node in the tree list causes it to be selected in the map and displays the node information (#1 in Figure 7-2). Selecting a node in the map selects the corresponding node in the tree list.

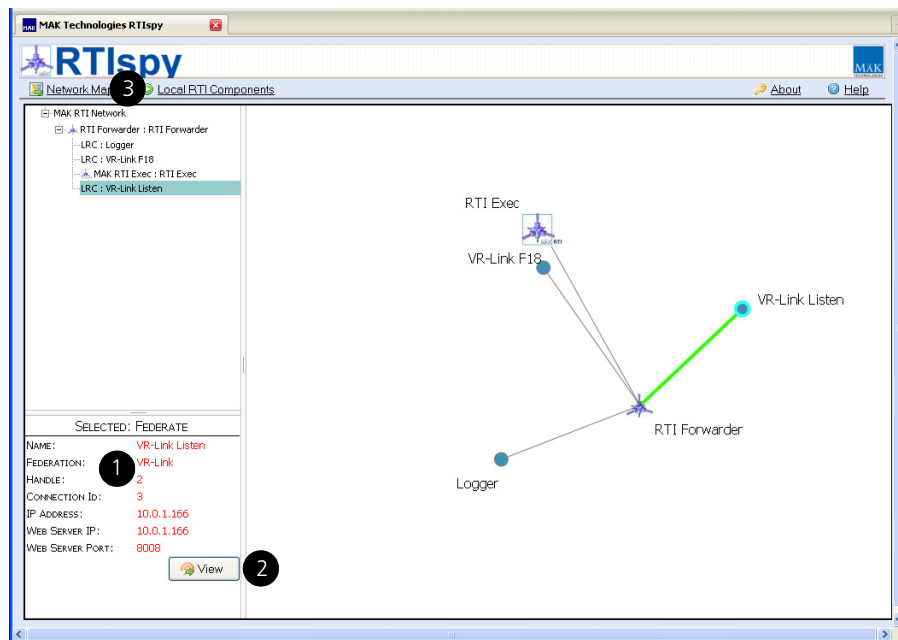


Figure 7-2. Network map with federate selected

You can also display node information in a popup window by hovering the mouse over the node in the map (Figure 7-3).

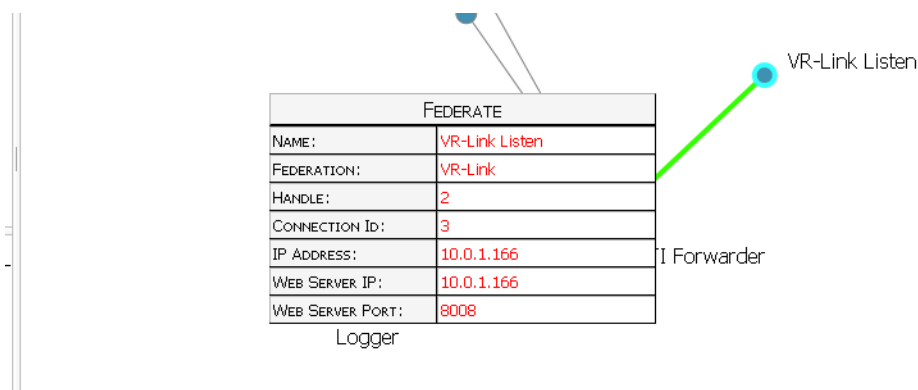


Figure 7-3. Popup information window

From the network map, you can get to pages for the rtiexec or any federate. When the rtiexec or a federate is selected, a view button is displayed in the Selected: Federate frame (#2 in Figure 7-2). Clicking the view button opens a window containing the page for the selected node. You can also open a federate or rtiexec page by double-clicking the node in the network map.

- To display the network map from any RTIspy page, select the network map link at the top of the page (#3 in Figure 7-2).



To view the network map, you must run the rtiexec and must run internal message reliable.

7.4.2. The RTI Network Component View

The RTI Network Component View lists the applications (federates, rtiexecs, and RTI Forwarders) running on the selected connection (Figure 7-4).

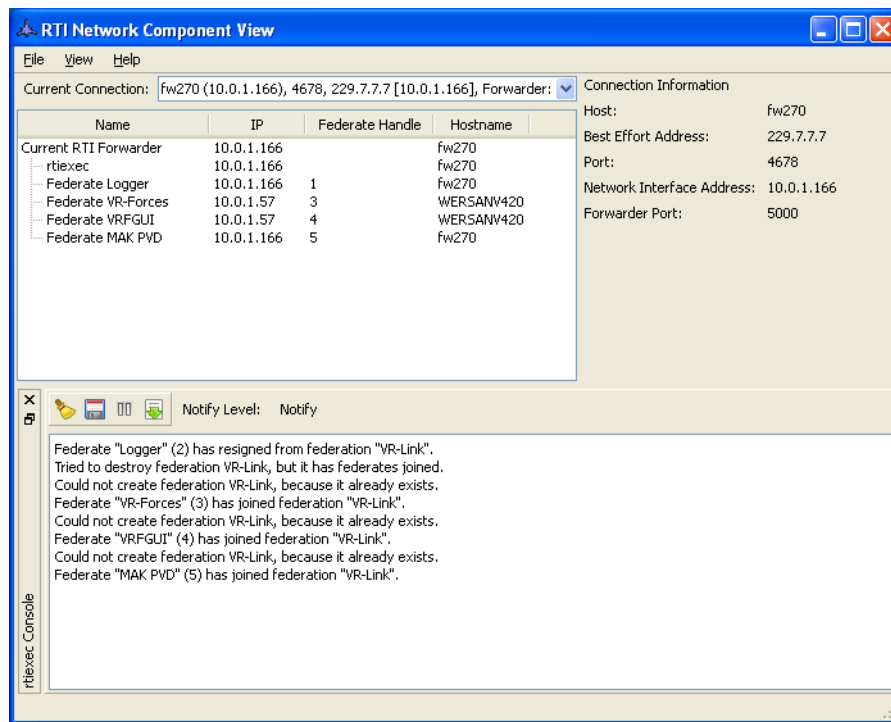


Figure 7-4. RTI Network Component View

To view network components:

1. Right-click the RTI Assistant icon in the system tray. A menu is displayed.
2. Choose **Network Component View**. The RTI Network Component View opens (Figure 7-4).

3. Select the connection whose components you want to view.

7.4.3. Viewing the MÄK RTI Components List

The Components list displays the components connected to an RTI web server [Figure 7-5](#). Each entry in the list is a link to a page for that component.

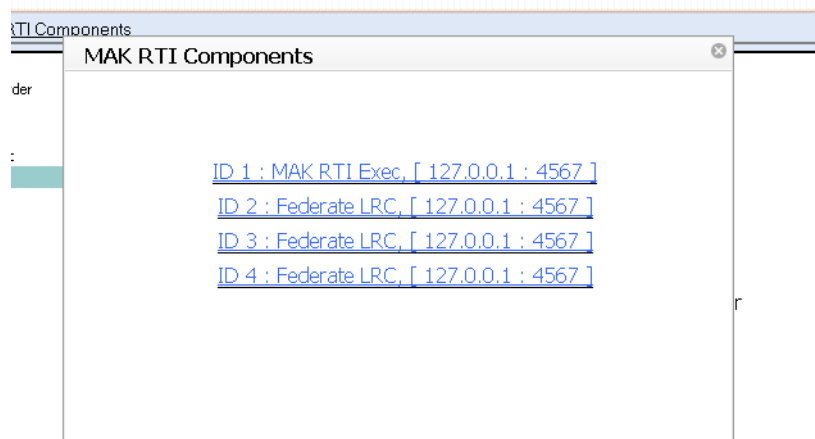


Figure 7-5. MÄK RTI Components list

- To display the MÄK RTI Components list, from any RTIspy page, click Local RTI Components ([Figure 7-6](#)).

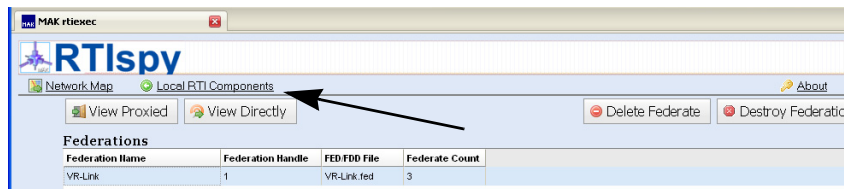


Figure 7-6. Local RTI Components link

7.5. Viewing Information about a Federation

The RTI Federations View and RTIspy display information about federations.

To display information about a federation in the RTI Federations View:

1. In the system tray, right-click the RTI Assistant icon. A menu is displayed.
2. Choose Federations View. The RTI Federations View opens.
3. In the Current Connection list, select the connection that you want to view.
4. Choose **View** → **Federation** → *federation*, where *federation* is one of the listed federations running on this connection. Information about the federation is displayed (Figure 7-7).

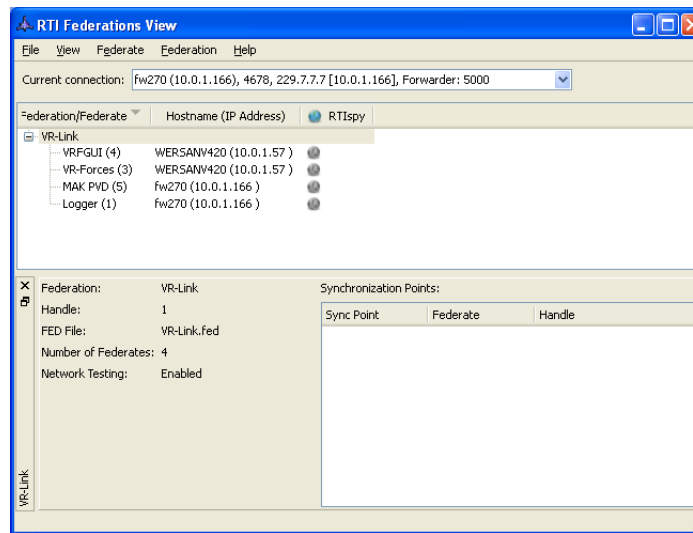


Figure 7-7. RTI Federation View

7.5.1. Viewing Federation Information in the RTIs Spy

The Federations window on the RTIs Spy rtiexec page ([Figure 7-8](#)) displays a list of federations connected to the RTI and connection information about all federates. It also serves as a portal to the current set of active LRCs.

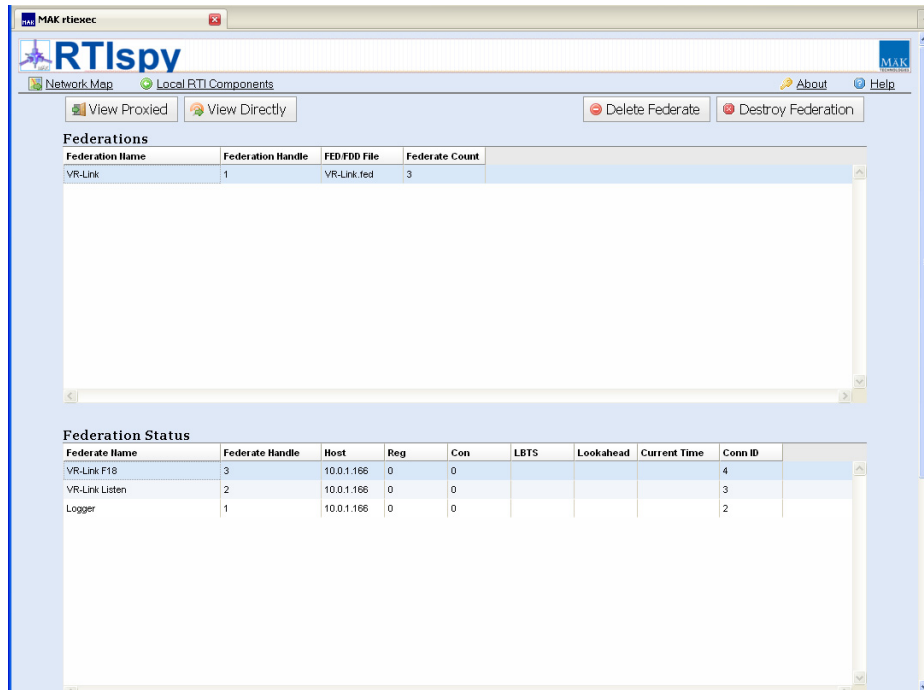


Figure 7-8. Federations window, rtiexec page

To view federation information in the RTIs Spy:

1. On the network map, select the rtiexec and click the View button, or in the Components List, click the listing for the rtiexec. (For details about the Components List, please see [“Viewing the MAK RTI Components List,”](#) on page 7-10.) The rtiexec page opens.
2. To view information about a federation, select it in the Federations window. The federates in that federation are listed in the Federation Status window.



View a node by proxy when that federate is not directly reachable from the viewing location. For details, please see [“Running the RTIs Spy Web Servers,”](#) on page 6-4.

7.6. Viewing Time Management State Information

If time management is enabled, the RTI Federations View and the RTIspy can display time management state information for federates. You can display the following information:

- ♦ Regulating — Is the federate regulating?
- ♦ Constrained — Is the federate constrained?
- ♦ LBTS — The Lower Bound Time Stamp on messages that the federate may generate. This view of LBTS is the time to which a regulating federate constrains all other constrained federates, rather than the minimum LBTS by which the federate is itself constrained. When the federates are sorted by the LBTS value, the minimum LBTS value indicates the LBTS of messages that any federate may receive (except the smallest federate that uses the next smallest value).
- ♦ Lookahead — The federate's current lookahead setting.
- ♦ Current Time — The federate's current view of simulation time. This is the time of the federate's last Time Advance Grant, and is unaffected when the federate enters the Time Advancing state.



- ♦ In the RTI Federations View, you can configure which time management state information is displayed. In the RTIspy all information is displayed.
 - ♦ Displaying time management information adds some overhead to RTI performance.
-

To display time management state information:

1. In the system tray, right-click the RTI Assistant icon. A menu is displayed.
2. Choose Federations View. The RTI Federations View opens.
3. In the Current Connection box, select the connection that you want to view.
4. Choose **View** → **Federate Columns** → **Customize**.
5. Select the time management columns that you want to display.
6. Click OK. The columns are added to the display ([Figure 7-11](#)).



You can also display or hide columns individually by selecting them on the Federate Columns sub-menu.

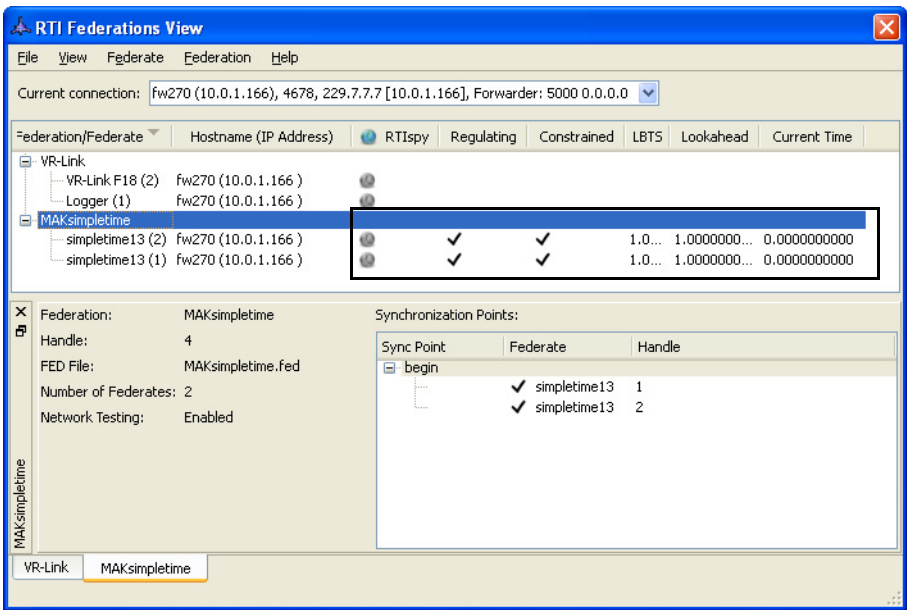


Figure 7-9. Time management state information

7.6.1. Viewing Time Management State Information in the RTIs Spy

To view time management state information in the RTIs Spy:

1. In the network map or the Components List, select the rtiexec. The rtiexec page opens.
2. Select the federation for which you want to view time management information. Time management status is displayed in the Federation Status window (Figure 7-10).

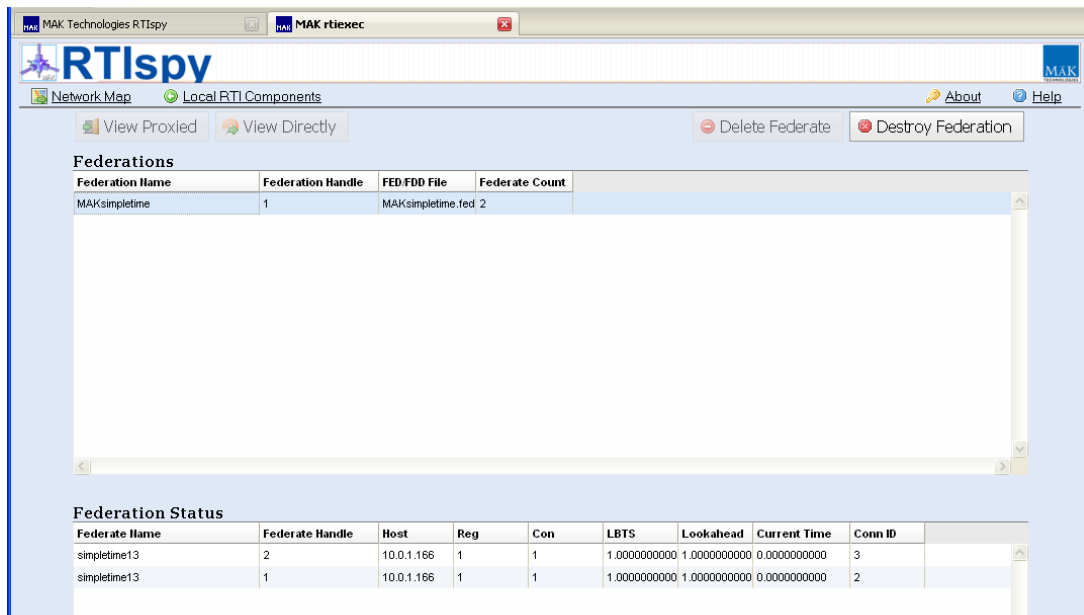


Figure 7-10. Time management status in the RTIs Spy

7.7. Viewing Synchronization Points

You can view the status of a federation's synchronization points in the RTI Federations View and RTIsPy.

To view synchronization points:

1. In the system tray, right-click the RTI Assistant icon. A menu is displayed.
2. Choose Federations View. The RTI Federations View opens.
3. In the Current Connection box, select the connection that you want to view.
4. Choose **View** → **Federation** → *federation_name*. The window expands to show information about the federation and a list of synchronization points (Figure 7-11).

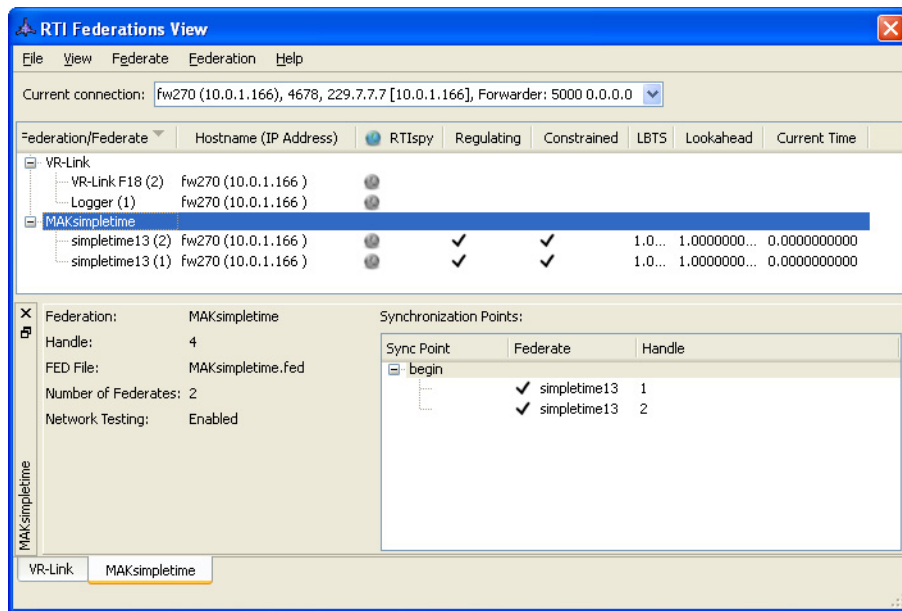


Figure 7-11. Synchronization points

5. For each synchronization point, you can expand and collapse the list of federates for that point.

To view synchronization points in the RTIsPy GUI:

1. In the network map, select the rtiexec and click the View button, or in the Components List, click the listing for the rtiexec. (For details about the Components List, please see “[Viewing the MÄK RTI Components List](#),” on page 7-10.) The rtiexec page opens.
2. Scroll to the bottom of the page to see synchronization points.

7.8. Viewing Federation-Wide Notification Messages

The RTI Assistant displays Fatal and Warn messages received from all federates running on the same machine. The messages are displayed in a popup window when they are received (Figure 7-12). Thereafter, you can view them in the Local Component Notification History window. An exclamation point is added to the RTI Assistant icon until the next time the notification history is viewed (Figure 7-13). You can clear, pause, and save the notification history. (For details, please see “Common Operations in RTI Assistant Log Windows,” on page 5-8.)

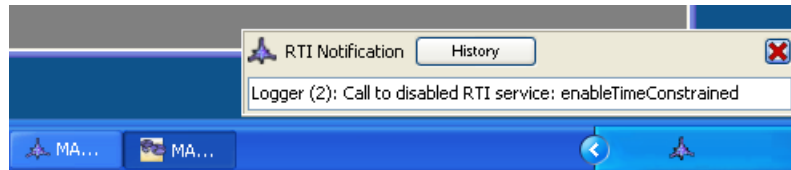


Figure 7-12. RTI notification popup window

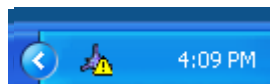


Figure 7-13. RTI Assistant icon message prompt

To display the RTI Notification History:

1. Right-click the RTI Assistant icon on the notification area of the task bar.
2. On the popup menu, choose Local Notification History. The Local Component Notification History window opens (Figure 7-14).

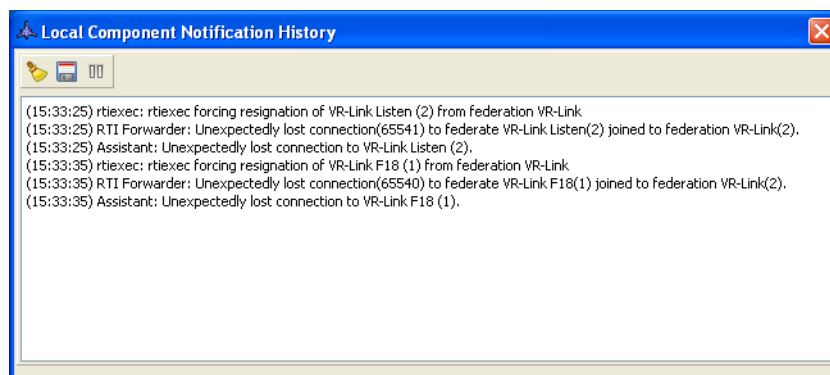


Figure 7-14. Local Component Notification History window

7.9. Configuring Fault Tolerance Strategies

Fault tolerance is a set of strategies for responding to problems within a federation, such as a federate dropping out without resigning, or unusual delays in connecting to the RTI.

7.9.1. Responding to Abnormal Termination

The MÄK RTI supports automatic removal of abnormally terminated federates and their orphaned objects. The RTI detects terminated federates as follows:

- It detects broken TCP connections between the LRC and the rtiexec (that is, by its RTI Forwarder component).
- It monitors a heartbeat message sent by each LRC to the federation execution.

For RTI 1.3 and 1516, the removal of a terminated federate from a federation is treated approximately as if the federate resigned either with an action of `RTI::RELEASE_ATTRIBUTES` or `RTI::DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES`. For HLA Evolved, a federate that is abnormally terminated is resigned with the action specified by the `automaticResignAction` in the FDD file.

To detect broken TCP connections, an LRC must establish a TCP connection to the rtiexec's RTI Forwarder (`RTI_tcpForwarderAddr`). When the rtiexec detects that this TCP connection has been lost, the federate is forcibly removed from its federation, if it is joined.

The heartbeat mechanism has each LRC send a heartbeat message to its federation executive once it has joined. If the federation executive does not receive a heartbeat message within a specified time-out interval, the federate is forcibly removed from the federation. To configure the heartbeat interval, set `RTI_enableFederateHeartbeat` to 1, and edit the `RTI_federateHeartbeatInterval` parameter, as follows.

```
RTI_federateHeartbeatInterval heartbeat
```

where *heartbeat* specifies the frequency with which each federate LRC sends a heartbeat message. The default heartbeat is 10 seconds.

To configure the timeout interval, set `RTI_enableFederateHeartbeat` to 1, and edit the `RTI_federateTimeoutInterval` parameter, as follows:

```
RTI_federateTimeoutInterval timeout
```

where *timeout* specifies the time period in which the federation must receive a heartbeat or it will forcibly remove the federate. The default is 25 seconds.



The asynchronous IO thread in the asynchronous IO or asynchronous message processing modes does not generate heartbeats. Heartbeats are generated during the call to `tick()`, `evokeCallback()`, or `evokeMultipleCallbacks()`. Therefore the federate must call these services more frequently than the timeout interval.

Handling Orphaned Objects

The object attributes owned by a federate that has terminated abnormally are considered to be orphaned. HLA does not address how they should be treated. The MÄK RTI fault tolerance behavior attempts to process these objects as if the federate resigned directing the RTI to either delete the objects or release the attributes. The `RTI_deleteOrphans` parameter specifies how the RTI handles orphaned objects. If `RTI_deleteOrphans` is enabled and the federate owns the `privilegeToDelete` attribute, the orphaned objects are deleted from the federation. If the federate does not own the object's `privilegeToDelete` attribute, the federate's owned attributes become orphaned and they cannot be recovered. If the `RTI_deleteOrphans` parameter is disabled, the federate's owned attributes become orphaned even if the federate owns the `privilegeToDelete` attribute. The RTI does not support ownership transfer of orphaned attributes. A federation can choose to allow orphaned attributes so that attributes owned by other federates can still be updated.

To specify deletion of orphaned objects, enable the `RTI_deleteOrphans` parameter, as follows:

```
RTI_deleteOrphans mode
```

where `mode` specifies whether or not the objects for which a terminated federate owns the `privilegeToDelete` will be automatically deleted.

- ♦ 0 — disabled (all owned attributes are orphaned)
- ♦ 1 — enabled.

Default: 1.

7.9.2. Reconnecting to the RTI Forwarder

You can configure an LRC to try to re-establish its connection to the RTI Forwarder if the connection breaks. This feature may allow the federate to continue running despite a large class of network layer errors.

To reconnect to the RTI Forwarder, set the `RTI_reconnectEnabled` parameter to 1, and specify a value for `RTI_federateReconnectPause` and `RTI_rtiExecReconnectPause`. If reconnect is enabled, when a broken connection is detected, the federate waits for the time specified by `RTI_federateReconnectPause` to allow the network to stabilize, then it tries to re-establish the broken connection. If an rtiexec detects a broken connection to the forwarder, the rtiexec waits for the time specified by `RTI_rtiExecReconnectPause` before it tries to re-establish the connection. The value for `RTI_rtiExecReconnectPause` must be lower than the value of `RTI_federateReconnectPause`. If the rtiexec and some number of federates are disconnected from the forwarder, it is important that the rtiexec re-establish its connection before any federates try to reconnect; otherwise the federates will not succeed. A connection will be dropped if the rtiexec or federate does not re-establish it within the time: $(2 \times \text{ReconnectPause}) + \text{response\_interval}$. (For details about specifying the response interval, please see “[Specifying the Response Interval](#),” on page 7-20.)

If the federate reconnects, it usually continues to operate in the federation with no visible effects to the application. If the federate cannot reconnect, the LRC throws an `RTIInternalError` exception to indicate that the federate has lost its connection to the federation (as it does when `RTI_reconnectEnabled` is disabled). During the reconnection period some reliable messages may be lost, because the federate is temporarily disconnected from the RTI.

7.9.3. Specifying the Response Interval

You can control the amount of time (in seconds) that an LRC waits to receive a response from the rtiexec when processing the `createFederation()`, `destroyFederation()`, or `joinFederation()` service calls. You set the response interval with the `RTI_responseInterval` parameter. The default interval is 3.0 seconds.

A federate's LRC can sometimes experience unusually long delays when communicating with the rtiexec to process the `createFederation()` or `joinFederation()` service calls. As a result, the federate will not be able to complete its connection to the rtiexec and will report, “Could not connect to rtiexec”. The service call throws an `RTIInternalError` exception. The solution to this problem is to increase the amount of time that the LRC waits for a response.



Other connection issues may cause the LRC to report the same type of connection error, for example, mismatched UDP ports.

The syntax for the `RTI_responseInterval` is:

```
(setqb RTI_responseInterval interval)
```

7.9.4. Processing Discovery of Unknown Objects

By default, the RTI does not perform discovery processing for updates of unknown objects. The RID parameter, `RTI_processUnknownUpdatesForDiscovery`, allows you to control whether updates from unknown objects can cause discovery. By default this option is turned off to improve efficiency. Turning it on provides fault tolerance for dropped register messages when internal messages are sent using UDP. The syntax is:

```
(setqb RTI_processUnknownUpdatesForDiscovery mode)
```

where *mode* is:

- ♦ 0 — do not process unknown updates for discovery
- ♦ 1 — process unknown updates for discovery.

Default: 1.



`RTI_processUnknownUpdatesForDiscovery` should almost always be disabled when `RTI_internalMsgReliableWhenUsingRtiexec` is enabled. `RTI_processUnknownUpdatesForDiscovery` was designed specifically to mitigate problems encountered when reliable messaging is disabled. So, in almost all cases, this setting provides no benefit when `RTI_internalMsgReliableWhenUsingRtiexec` is enabled. Enabling both parameters may lead to unexpected results in some configurations. For example, an object may have an update delivered after it has been deleted, resulting in its mistaken rediscovery.

7.9.5. Choosing Silent Failure Instead of Exceptions

You can have your function calls fail silently rather than throwing an exception when they call a service that is not activated in the MÄK RTI. Use the `RTI_throwExceptionForDisabledService` parameter in the RID file. The syntax is:

```
(setqb RTI_throwExceptionForDisabledService 0)
```

where:

- ♦ 0 — do not throw exceptions
- ♦ 1 — throw exceptions.

The default is to throw exceptions.

7.10. Configuring Save/Restore

You can configure the number of seconds the federation execution waits for a save/restore operation to complete before indicating a failure with the `RTI_saveRestoreTimeout` parameter. The default is 120 seconds.

The heartbeat mechanism is not necessary when reliable transport is enabled and a TCP connection is established by the federate. The detection of a broken TCP connection is sufficient and better than the heartbeat. However, a broken TCP connection is often not detected until the federate sends a message. If a federate is often in a quiescent state in which it is not sending any messages, enabling the heartbeat is one way to ensure early detection of the broken TCP connection. As long as the federate is ticking the RTI, it sends the heartbeats automatically.

7.11. Viewing Notification Messages from the *rtiexec*

The RTI Network Component View has a console that can display output from the *rtiexec* to which it is connected (Figure 7-15). You can hide or show the console. You can clear, pause, and save the log. (For details, please see “Common Operations in RTI Assistant Log Windows,” on page 5-8.) You can also load the most recent log entries.

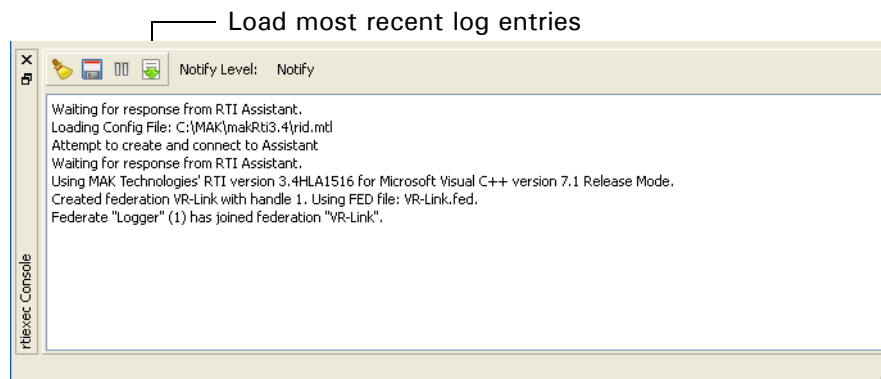


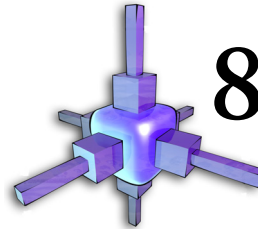
Figure 7-15. Console window

- To hide or show *rtiexec* console output, in the RTI Network Component View, choose **View** → **rtiexec Console**.

7.11.1. Displaying Recent Log Entries

When you switch connections, the *rtiexec* Console clears the console and begins displaying messages as they are received. You can display log entries generated before you selected the current connection. When you request recent log entries, the RTI Assistant sends as many log entries as it can fit into a message.

- To display recent log entries, on the *rtiexec* Console window, click the Load Most Recent Log Entries icon (Figure 7-15).



Managing Federates

This chapter explains how to view information about federates and how to configure them.

Introduction	8-2
Configuring Federate Connections in the RID File	8-2
Displaying Federate Information in the RTIspy	8-3
Displaying Federate LRC Details.....	8-4
Displaying Federate Objects and Interactions.....	8-5
Displaying the Object Instances Page	8-5
Displaying Object Attribute Information	8-6
Undiscovering Objects.....	8-7
Displaying Federate Interactions.....	8-7
Displaying FOM Object and Interaction Classes	8-8
Displaying a Class's Attributes or Parameters	8-8
Displaying the Classes a Federate is Publishing and Subscribing To	8-9
Displaying the Federate LRC Event Log.....	8-10
Viewing LRC Diagnostic Messages	8-11
Enabling LRC Diagnostic Data Logging in the RID File.....	8-11
Enabling LRC Diagnostic Data Logging from the RTI Assistant	8-12
Displaying LRC Diagnostic Messages	8-13
Using the RTIspy to Debug an Application	8-14

8.1. Introduction

The RTIspy and RTI Assistant have options for viewing information about federates and managing them. You can:

- Display information about federates
- Display information about objects and interactions
- View event logs
- Resign federates from a federation.

This chapter describes these procedures and how to configure federates.

8.1.1. Configuring Federate Connections in the RID File

By default, federates connect to a federation by specifying a connection configuration in the RTI Assistant. To configure connection settings in the RID file, instead of using the RTI Assistant, specify values for the following parameters:

- To configure a federate to try to connect to the rtiexec, set `RTI_useRtiExec` to 1.
- To configure federates for lightweight mode, set the RID file parameter `RTI_useRtiExec` to 0.
- To specify full compliance mode, set `RTI_forceFullCompliance` to 1.
- Specify the port with `RTI_udpPort`.
- Specify the best effort address with `RTI_destAddrString`.
- Specify the RTI Forwarder address with `RTI_tcpForwarderAddr`.
- Set the `RTI_internalMsgReliableWhenUsingRtiexec`, `RTI_fomDataReliableWhenUsingRtiexec`, `RTI_mcastDiscoveryEnabled`, and `RTI_forceFomDataReliable` parameters to your preference.
- To force the RTI to use the values specified for these parameters instead of those in the connection configuration, set `RTI_configureConnectionWithRid` to 1.

8.2. Displaying Federate Information in the RTIspy

The RTIspy can display the following information about a federate:

- ♦ Federate identification information
- ♦ Federate object and interaction information
- ♦ Configuration settings (RID file)
- ♦ FOM objects and interactions
- ♦ LRC event log.



In any RTIspy table, you can sort columns in ascending or descending order by clicking the column heading.

To display federate information, use one of the following methods:

- ♦ In the Components list, click the listing for the federate you want to view (“[Viewing the MÄK RTI Components List](#),” on page 7-10).
- ♦ In the network map, select the federate you want to view and click the View button (“[Viewing Network Components](#),” on page 7-7).
- ♦ On the federations page, select the federate in the Federation Status list (“[Viewing Federation Information in the RTIspy](#),” on page 7-12). Then, click the View Proxied button (#1 in [Figure 8-1](#)) to open a proxy connection to that federate in the current page, or click the View Directly button (#2 in [Figure 8-1](#)) to open a direct connection to that federate in a new page.

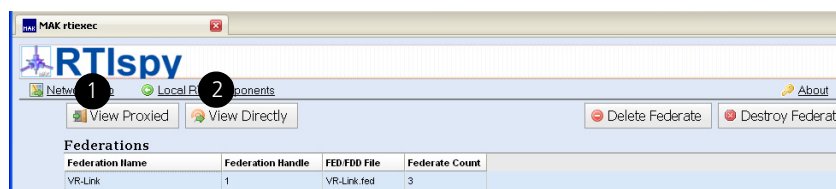


Figure 8-1. View federate buttons

The default view of the federate is the Object Instances page.

8.2.1. Displaying Federate LRC Details

- To display application information about a federate, choose **Display** → **Federate Information**. The Federate Information window opens (Figure 8-2).

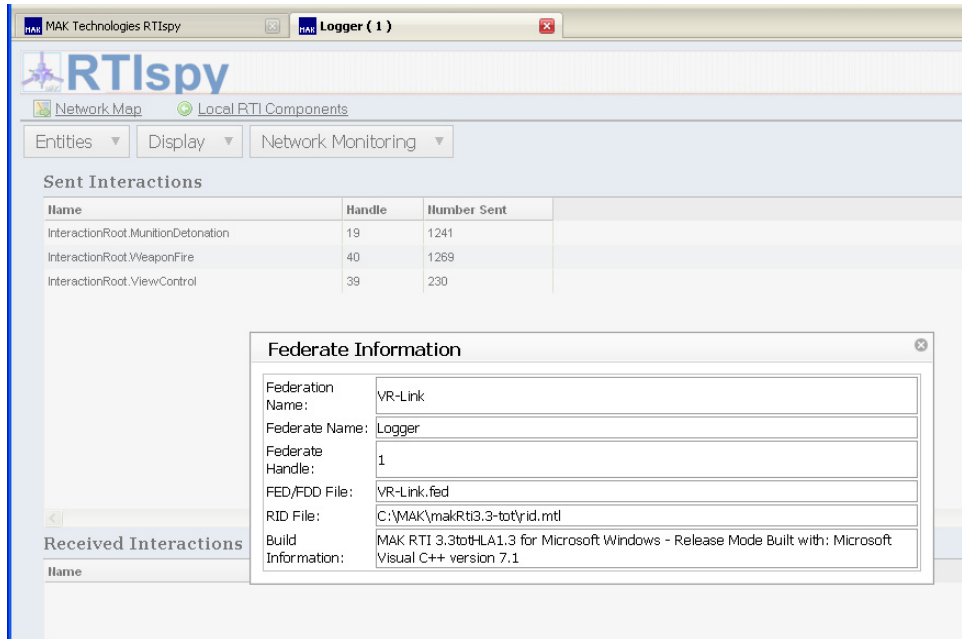


Figure 8-2. Federate Information window

8.3. Displaying Federate Objects and Interactions

The RTIsPy can display information about the objects and interactions a federate is publishing.

8.3.1. Displaying the Object Instances Page

The Object Instances page (Figure 8-3), is the default view for a federate. If you display other federate pages you can return to the Object Instances page.

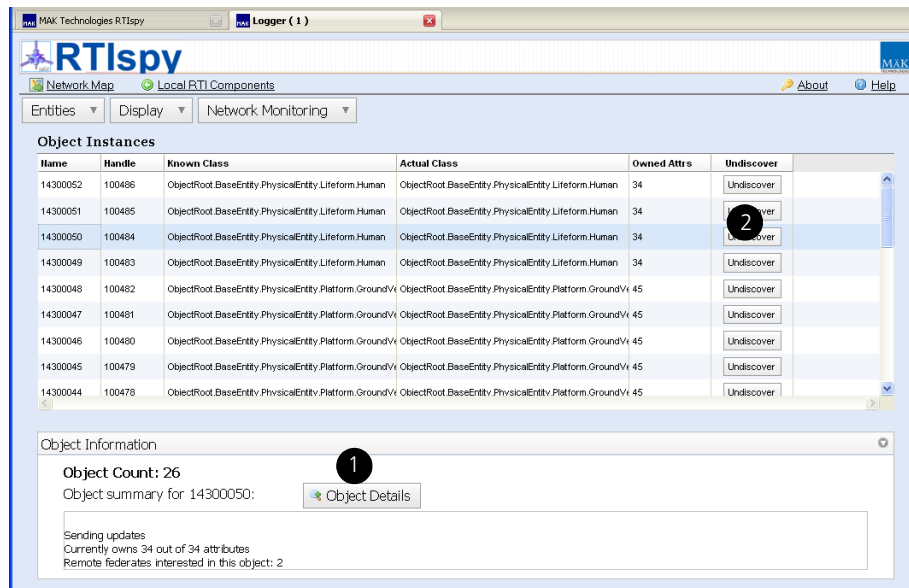


Figure 8-3. Federate Object Instances page

- To display the Object Instances page, from any LRC window, choose **Entities** → **Objects**.

8.3.2. Displaying Object Attribute Information

The Object Details window (Figure 8-4) shows you which attributes are being updated for the selected object. This is particularly useful when you are working with ownership management. The window also shows ownership information.

To display the attributes of an object:

1. In the Object Instances page, select the object for which you want additional information.
2. Click the Object Details button (#1 in Figure 8-3). The Object Details window opens (Figure 8-4).

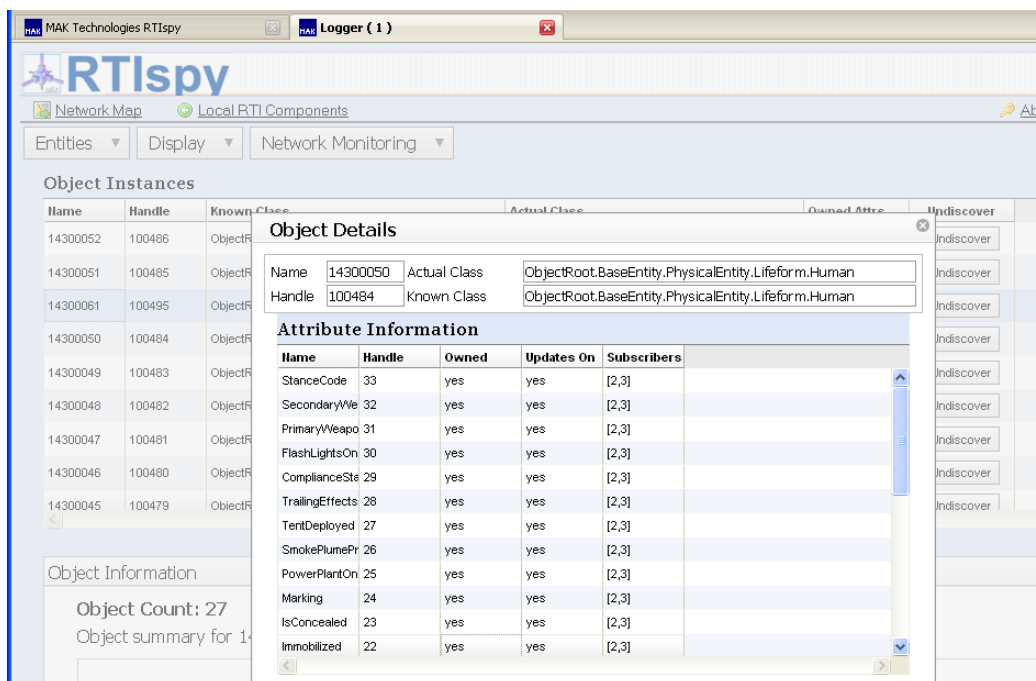


Figure 8-4. Object Details window

8.3.3. Undiscovering Objects

You can remove an object from the list of discovered objects. This action is a way to remove orphan objects left by a crashed federate. (For more information, please see [“Configuring Fault Tolerance Strategies,”](#) on page 7-18.) This action behaves as if the federate invoked the `localDeleteObjectInstance` service, but the LRC also invokes the `removeObjectInstance` service.



The object may be rediscovered if a federate that owns attributes still exists in the federation.

- To undiscover an object, in the Object Instances page, click the Undiscover button for the object (#2 in [Figure 8-3](#)).

8.3.4. Displaying Federate Interactions

- To view Federate interactions, on a federate page, choose **Entities** → **Interactions**. The browser displays sent and received interactions ([Figure 8-5](#)).

Sent Interactions		
Name	Handle	Number Sent
InteractionRoot.MunitionDetonation	19	1191
InteractionRoot.WeaponFire	40	1218
InteractionRoot.ViewControl	39	223

Received Interactions		
Name	Handle	Number Received

Figure 8-5. Interactions page

8.4. Displaying FOM Object and Interaction Classes

The RTIsPy GUI lets you view a list of the FOM's objects and interactions.

- To display FOM object classes, on a federate page, choose **Display** → **FOM Objects**.
- To display FOM interaction classes, on a federate page, choose **Display** → **FOM Interactions**.

8.4.1. Displaying a Class's Attributes or Parameters

The tree view on the left side of the Federate FOM view shows the current set of object (Figure 8-6) or interaction (Figure 8-7) classes.

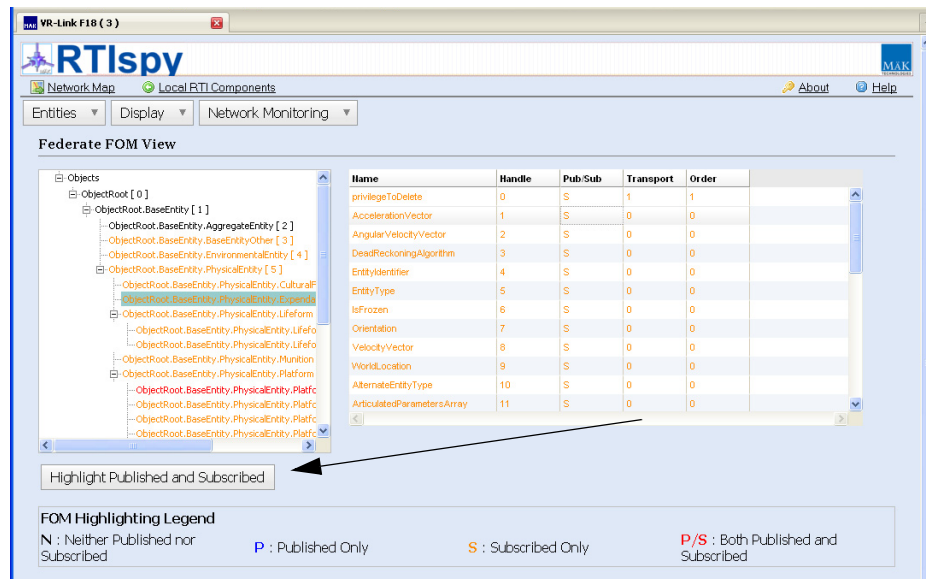


Figure 8-6. Federate FOM View, object classes

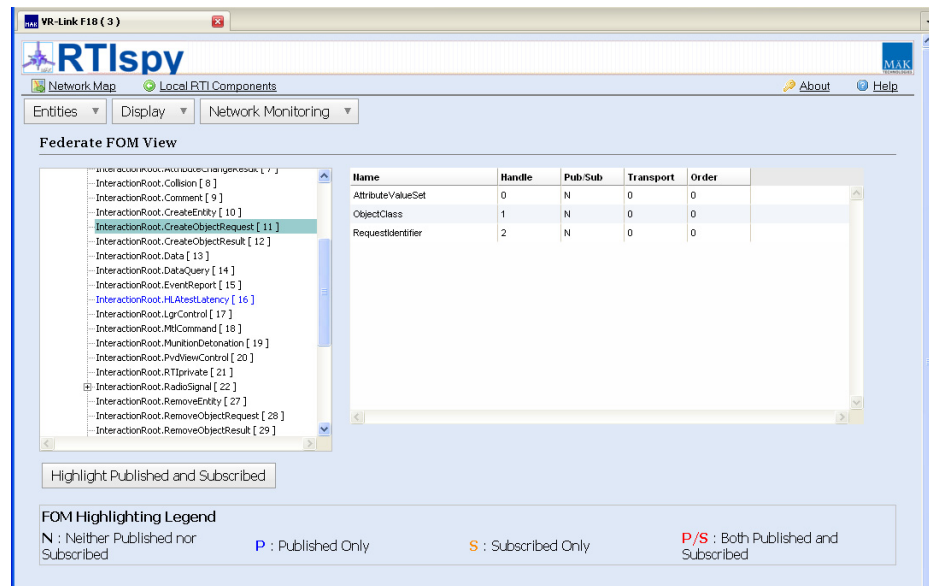


Figure 8-7. Federate FOM View, interaction classes

- To display the list of attributes or parameters for a class, select the class in the tree view.

8.4.2. Displaying the Classes a Federate is Publishing and Subscribing To

The RTIsPy can highlight the classes that a federate is publishing or is subscribing to. A key to the color scheme for highlighting published and subscribed classes is at the bottom of the window.

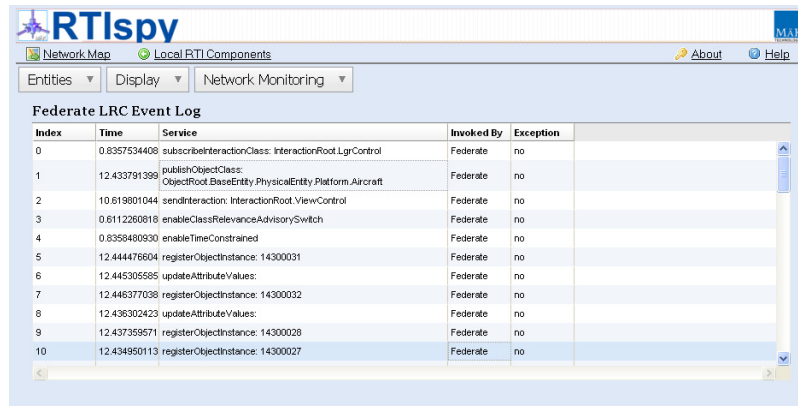
- To highlight a federate's current publication and subscription interests, click the “Highlight Published and Subscribed” button (Figure 8-6).

8.5. Displaying the Federate LRC Event Log

The event log (Figure 8-8) shows a list of actions performed by the federate on the RTI and a list of events generated in the federate by the RTI. Since RTI events occur rapidly, monitoring the event log is a relatively expensive operation.

The Invoked By column indicates whether a message was invoked by a federate or the RTI. RTI-invoked messages are printed in purple to help distinguish them without the need to view the Invoked By column.

- To display the monitored federate's LRC event log, choose **Display** → **LRC Log**.



Index	Time	Service	Invoked By	Exception
0	0.8357534408	subscribeInteractionClass: InteractionRoot.LgrControl	Federate	no
1	12.433791399	publishObjectClass: ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft	Federate	no
2	10.619801044	sendInteraction: InteractionRoot.ViewControl	Federate	no
3	0.6112260818	enableClassRelevanceAdvisorySwitch	Federate	no
4	0.8358480830	enableTimeConstrained	Federate	no
5	12.444476604	registerObjectInstance: 14300031	Federate	no
6	12.445305585	updateAttributeValues:	Federate	no
7	12.446377038	registerObjectInstance: 14300032	Federate	no
8	12.436302423	updateAttributeValues:	Federate	no
9	12.437359571	registerObjectInstance: 14300028	Federate	no
10	12.434950113	registerObjectInstance: 14300027	Federate	no

Figure 8-8. LRC Event log

8.6. Viewing LRC Diagnostic Messages

The RTI can capture diagnostic messages sent by LRCs. You have two options for enabling message logging, as follows:

- ♦ Enable logging in the RID file. If you enable logging in the RID file, all messages are logged to a specified file from the time the federate starts until it exits. You can view these messages by opening the log file in a text editor. You can also view the log at any time while the federate is running and save the current snapshot of messages to a file.

i

When logging is enabled in the RID file, if there is more than one federate running on a PC, a separate log file of the specified name is created in the directory in which the executable is located. If you are running more than one executable from the same directory, for example the VR-Link talk and listen applications, data for both federates is written to the same log file unless you set `RTI_reuseLogFile` to 0.

- ♦ Enable logging from the RTI Assistant. If you enable logging from the RTI Assistant, no messages are logged until you enable logging, so startup messages are not captured. The messages are not saved unless you explicitly save them. You can view the log at any time while the federate is running and save the current snapshot of messages to a file. Any messages that are sent between the time you save a snapshot and shut down the federate are not saved.

i

- ♦ Once you start logging messages, either globally or for a particular LRC, you cannot turn off the logging process. However, you can change the notification level to reduce the number of messages logged.
 - ♦ You can view log files only for federates on the local machine.
-

8.6.1. Enabling LRC Diagnostic Data Logging in the RID File

The `RTI_logfileName` parameter enables or disables logging of LRC diagnostic data. It is disabled by default to improve performance. To enable logging, specify a name for the log file. To disable logging, specify an empty string ("") or comment out the parameter.

8.6.2. Enabling LRC Diagnostic Data Logging from the RTI Assistant

If logging is not enabled in the RID file, you can enable it from the RTI Assistant. You must enable it individually for each federate whose messages you want to log. The log is not saved automatically.

To enable logging of LRC diagnostic data from the RTI Assistant:

1. Right-click the RTI Assistant icon. A menu is displayed.
2. Choose the federate for which you want to enable logging. A submenu is displayed.
3. On the submenu, choose **Enable Logging**. Depending on your RTI Assistant preferences, the log may be displayed, you may be prompted to display it, or there may be no action. For details about setting this preference, please see [“Configuring the RTI Assistant’s View Log Prompt,”](#) on page 5-6.



When logging is enabled from the RTI Assistant, the LRC does not create a permanent log file. The logged text is not saved to a file unless you explicitly save it. For details, please see [“Saving Console Output to a File,”](#) on page 5-8.

8.6.3. Displaying LRC Diagnostic Messages

When you display the LRC log, you get the current snapshot of the information that has been logged. You can clear, pause, and save the log. (For details, please see [“Common Operations in RTI Assistant Log Windows,”](#) on page 5-8.) You can also change the notification level for the log.

To display a federate’s message log:

1. Right-click the RTI Assistant icon. A menu is displayed.
2. Select the federate whose log you want to display. A submenu is displayed.
3. Choose **Show Log**. The Log window opens ([Figure 8-9](#)).

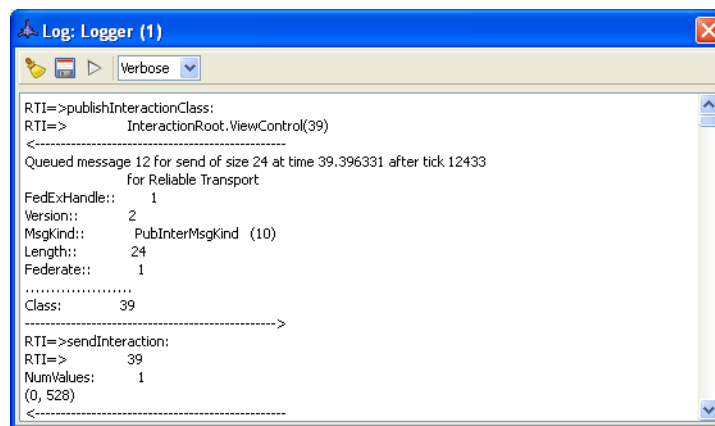


Figure 8-9. LRC message log

Changing the Notification Level for the LRC Log

- To change the notification level for the Log window, select an option in the drop-down list.

8.7. Using the RTIspy to Debug an Application

One of the difficulties in debugging HLA applications is finding out where a problem lies — in your application, in the RTI, in the network, in the FOM, and so on. By providing a view of each federate's interaction with the RTI, the RTIspy can help you isolate, monitor, and debug a problem.

Suppose that remote federates are not seeing objects that your application has registered for. How could you use the RTIspy to debug this problem?

To debug your federate:

1. Start your federate, a remote federate, and the RTIspy.
 - a. Check the Object Instances page for your federate (Figure 8-10). It lists all objects that have been registered by the federate and all objects that have been discovered. Make sure that it has registered the object in question.
 - b. Check the Object Instances page for the remote federate to see if it has discovered the object. If it has not, we need to find out why.

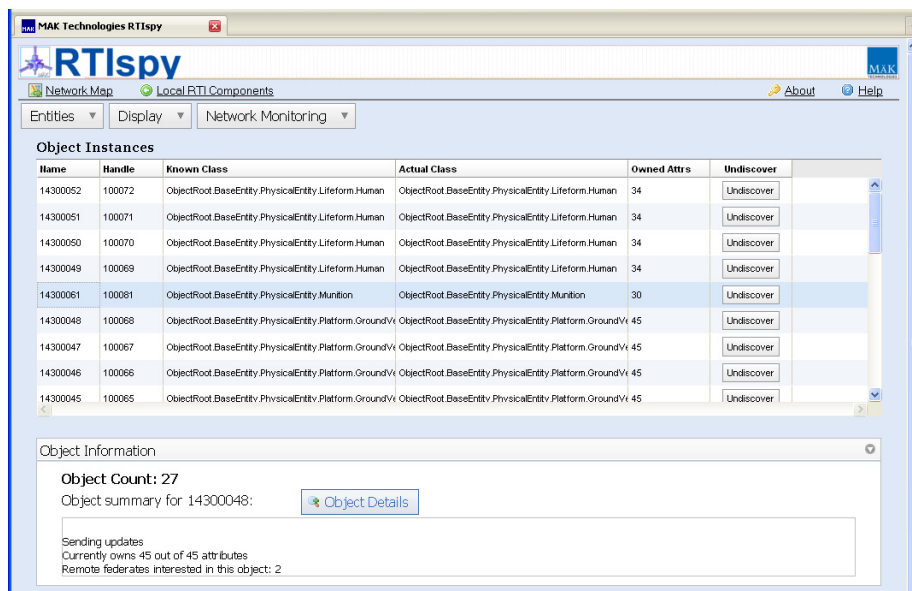


Figure 8-10. Object Instances page

2. Check the network settings for both federates.
 - a. For each federate, choose **Display** → **RID**. The Federate RID File Settings window displays the RID settings for the federate (Figure 8-11).
 - b. Make sure that the federates are using the same port and destination address. If they are, that is not the cause of the problem.

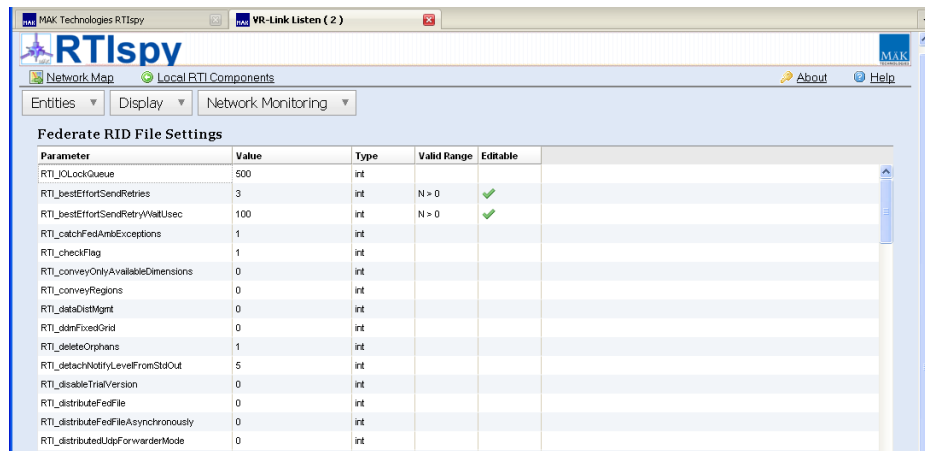


Figure 8-11. Configuration window

3. See if the local federate is publishing the object and if the remote federate is subscribed to it.
 - a. For each federate, choose **Display** → **FOM Objects**. The Federate FOM View page opens (Figure 8-12).
 - b. In the page for the local federate, click the “Highlight Published and Subscribed” button. In the class hierarchy, all objects published are highlighted.
 - c. Make sure the local federate’s object’s FOM class is being published.
 - d. Make sure the remote federate is subscribed to the object’s FOM class. If either federate is not publishing or subscribing to the correct objects, you need to update the federate to subscribe to or publish the object or its base class.

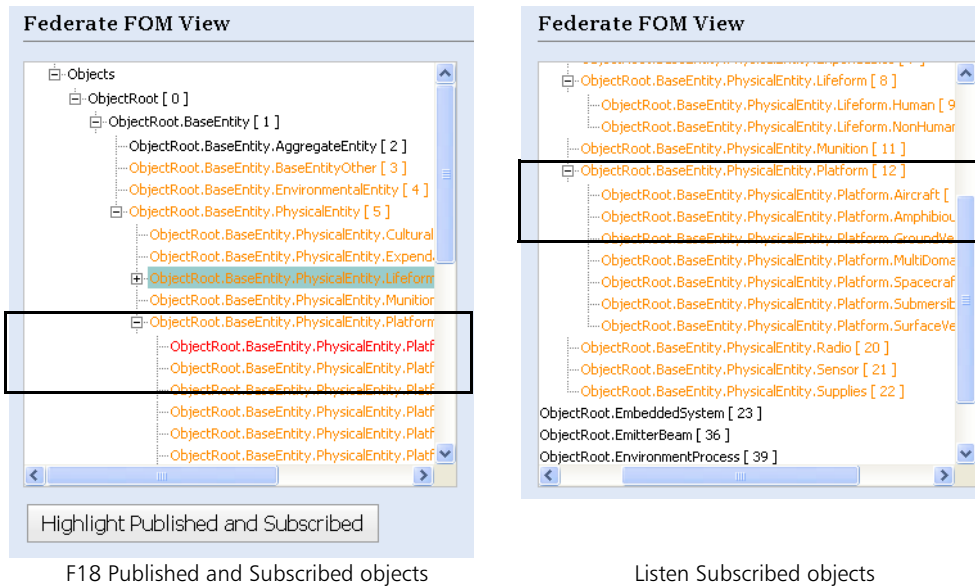
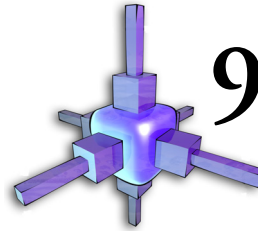


Figure 8-12. Federate FOM view, Objects

4. If none of these steps identifies a problem, there could be a problem in the RTI. If you have developed an RTI plug-in, debug it. Otherwise, contact MÄK technical support.



Monitoring the Network

This chapter explains how to monitor network components and the federates in a federation.

Introduction	9-2
Testing Latency Between Federates.....	9-3
Enabling Latency Testing	9-4
Running Latency Tests	9-4
Interpreting the Results of Latency Tests.....	9-5
Running Latency Tests Manually.....	9-8
Running Continuous Latency Tests.....	9-8
Configuring Latency Tests.....	9-9
The RTIspy Network Monitoring Tool.....	9-10
Enabling Network Monitoring.....	9-10
Logging Network Monitoring Statistics	9-10
Displaying Object and Interaction Statistics	9-11
Displaying RTI Messages.....	9-12
The RTI API Call Statistics Page	9-13
Examples of Using the RTIspy Network Monitoring Tool	9-14
Finding Bottlenecks Caused by Processing Callbacks.....	9-14
Identifying the Cause of Slow-Downs Under Time Management	9-15
Comparing the Costs and Benefits of RTI Features	9-16

9.1. Introduction

The HLA API provides many options for optimizing performance. Since each of the RTI options offers trade-offs between CPU usage, network overhead, and functionality, it can be very difficult to predict the final effect of the various options on overall network and simulation performance. For example, time management provides synchronization between federates, but imposes additional overhead in terms of message processing and increased complexity at the RTI level. Choosing data distribution management (DDM) over declaration management can greatly reduce the number of updates and interactions sent by the RTI, but adds processing overhead to each RTI call, which may result in an update or interaction message being sent, and in fact, may increase the number of messages sent under certain conditions.

To help you assess the results of using different performance optimizations, you can enable and run latency tests between federates and you can use the RTIspy's network monitoring tools to view object and interactions statistics, the frequency of RTI message types, and API call statistics.

9.2. Testing Latency Between Federates

The MÄK RTI can run latency tests among federates. You must have at least two federates in the federation to run latency tests. When you start latency testing for a federate, the RTI automatically runs a latency test from this initiating federate to all other federates in the federation. The results are displayed in the Latency Test window (Figure 9-1). You can run additional tests manually or run tests continually.

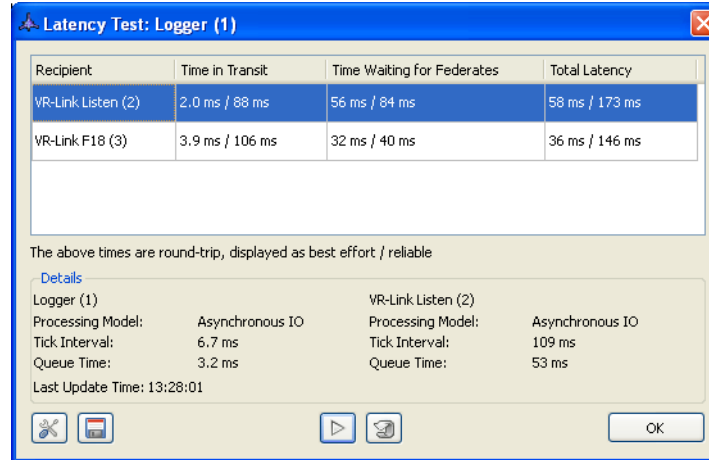


Figure 9-1. Latency Test window

You can configure:

- The number of interactions run in each test
- The time between interactions
- The size of the interaction message.

Each test does the following for each interaction that it sends:

1. Records the start time.
2. Sends an interaction from the initiating federate to a recipient federate. The recipient returns the interaction.
3. Waits until the amount of time calculated as `start_time + time_between_interactions` has elapsed.

While the initiator is sending interactions, it is also processing the return interactions and recording individual results. After the last interaction is sent to a given recipient federate and the wait time has expired (plus some extra time), a complete result for the initiator:recipient pair is computed and displayed.

9.2.1. Enabling Latency Testing

Latency tests are run between federates. Latency testing is enabled at the federation level. If you are running in lightweight mode, enable or disable latency testing with the **RTI_enableNetworkTesting** parameter. If the rtiexec is running, you can enable and disable latency testing in the RTI Federations View.

To enable or disable latency testing in the RTI Federations View:

1. In the system tray, right-click the RTI Assistant icon. A menu is displayed.
2. Choose **Federations View**. The RTI Federations View opens.
3. In the Current Connection box, select the connection that you want to view.
4. In the list of federations, select the federation that you want to test.
5. Choose **Federation** → **Network Testing Enabled**. A check mark next to the menu command indicates whether or not testing is enabled.

9.2.2. Running Latency Tests

You can run latency testing when the rtiexec is running and in lightweight mode. You must enable latency testing before you can run a test.



You must have at least two federates in the federation to run latency tests.



Latency Testing through the Federations View is provided for rtiexec connections only. Lightweight connections need to use the RTI Assistant Menu to run the latency testing

To run a latency test from the RTI Federations View:

1. In the system tray, right-click the RTI Assistant icon. A menu is displayed.
2. Choose **Federations View**. The RTI Federations View opens.
3. In the Current Connection box, select the connection that you want to view.
4. If latency testing is not enabled, choose **Federation** → **Network Testing Enabled**.
5. Select the federate that you want to test.
6. Choose **Federate** → **Run Latency Test**. The Latency Test window opens ([Figure 9-1](#)). The federate name is listed in the title bar.

You can run latency tests from the RTI Assistant menu. This procedure is available in both lightweight mode and when the rtiexec is running. Latency testing must be enabled, as described in [“Enabling Latency Testing,”](#) on page 9-4.

To run a latency test from the RTI Assistant:

1. Right-click the RTI Assistant icon.
2. On the popup menu, choose the federate you want to test. A secondary menu is displayed.
3. On the secondary menu, click **Run Latency Test**.

9.2.3. Interpreting the Results of Latency Tests

The Latency Test window displays the results for tests between the initiating federate and each recipient in a table in the results window. It displays additional details in the Details group box. The Latency Test window displays results for best effort and reliable mode latency. If results are not available for a particular transport type, a dash is displayed (Figure 9-2). The data reported in the latency test table is as follows:

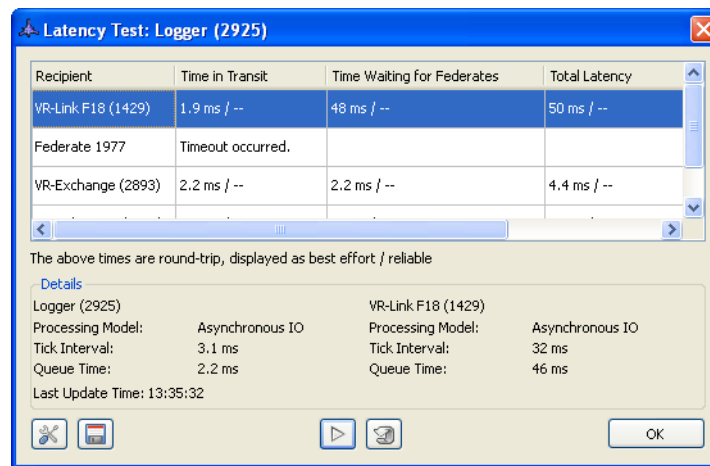


Figure 9-2. Latency test results, best effort only

- ♦ **Time in Transit.** The time in transit is the total latency of the message's round trip minus the time spent in queue waiting to be read, if any. The comparison of time in transit to total latency can help you understand the effect of using different processing modes and tick rates in your federates. For details, please see "[Understanding the Time in Transit Latency Test Results](#)," on page 9-6.
- ♦ **Time Waiting for Federates.** The time waiting for federates is the amount of time a message spends in a queue waiting for a federate to retrieve it. In other words, it is the difference between the time in transit and total latency.
- ♦ **Total Latency.** Total latency is the round trip time – the time from when the initiator sends the interaction to when the recipient's response is received.

i

Each of the values reported is the average of the values for the individual interactions sent in the test.

Understanding the Time in Transit Latency Test Results

The total latency of a message is the sum of the transit time of the message and any time that it has spent in a queue waiting for a federate to read it. The transit time is a function of network topology and hardware. The time spent in queue depends on the queuing strategy of the RTI and how frequently a federate ticks the network for new messages. Therefore, the time in transit values in the Latency Test window must be evaluated in the context of the processing model being used. (The MÄK RTI's processing models are described in [“The MÄK RTI Process Model,”](#) on page 2-3.) The process models are:

- ♦ Synchronous
- ♦ Asynchronous
- ♦ Asynchronous message processing
- ♦ Asynchronous callbacks.

In synchronous mode, and asynchronous callbacks mode, there is no internal RTI queue. Therefore, there should be no difference between the time in transit and total latency time. The latency test does not report time in transit or time waiting for federates ([Figure 9-3](#)).

The asynchronous and asynchronous message processing modes use message queues to store messages until a federate ticks the RTI and retrieves the message for processing. The total latency time does not tell you how much time a message has been in transit and how much time it has spent sitting in a queue. If you want to improve the performance of your federation, you need to separate these issues so that you know where to focus your efforts. Therefore, during latency testing, the RTI records the time when a message is placed on a queue and when it is removed so that it can report transit time and waiting time separately.

In asynchronous mode, if there is a significant discrepancy between your federate's time in transit and total latency (as shown by the time waiting for federates value), increasing the frequency with which your federate ticks the RTI may improve performance.

In asynchronous message processing mode, the RTI does some message processing before it queues the message. As a result of this preprocessing, when the federate ticks the RTI and retrieves the message it has less work to do. You can compare the latency results of asynchronous mode to asynchronous message mode to see if your federates improve latency by changing the processing model. Increasing the tick rate may still provide improved performance in this mode.

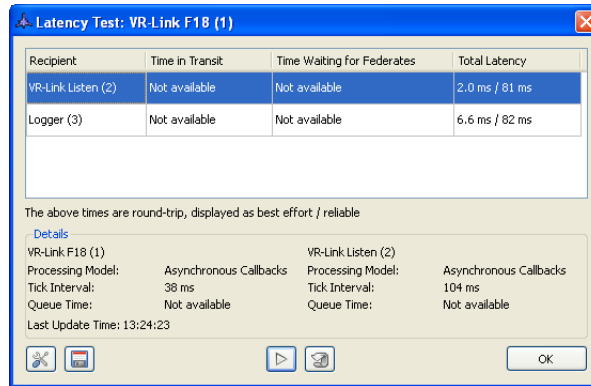


Figure 9-3. Latency test, asynchronous callbacks mode

Displaying Latency Test Details for a Federate

The Details section of the Latency Test window, (Figure 9-3) shows the following information:

- ♦ The timestamp of when the report was received and displayed
 - ♦ The federate name (handle) of the initiator and recipient
 - ♦ The processing model of the initiator and recipient, which can be:
 - Synchronous
 - Asynchronous
 - Asynchronous message processing
 - Asynchronous callbacks
 - ♦ The average frequency with which each federate calls tick()
 - ♦ The average amount of time that a message was queued at each federate.
- To display details about a federate, select it in the Latency Test window. The Details group box is updated.

9.2.4. Running Latency Tests Manually

After the RTI automatically runs its first latency test, you can run additional tests manually.

- To run a latency test manually, in the Latency Test window, click the Run icon (Figure 9-4).

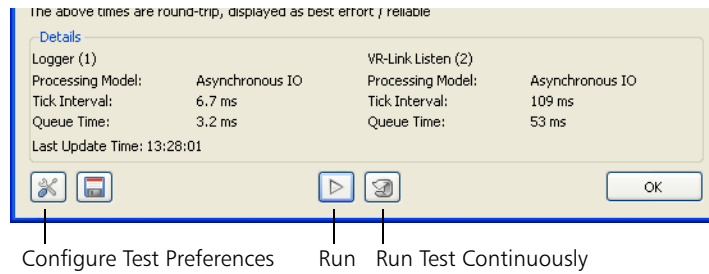


Figure 9-4. Latency Test window buttons

9.2.5. Running Continuous Latency Tests

You can have the RTI run continuous latency tests. There is no configurable break between each test run. The results of each test replace the results of the previous test in the Latency Test window.



If a federate drops out of an exercise, its last test results remain in the Latency Test window. They do not update, but there is no indication that the federate is no longer active.

- To run latency tests continuously, in the Latency Test window, click the Run Test Continuously icon (Figure 9-4).

9.2.6. Configuring Latency Tests

You can configure latency test parameters in the Latency Test window or in the RTI Preferences dialog box.

To configure latency test parameters:

1. Display latency test configuration options in one of the following ways:
 - On the RTI Assistant menu, choose **Preferences**. Then select the Latency Testing page (Figure 9-5).
 - In the Latency Test window, click the Configure Test Preferences icon (Figure 9-4). The Latency Test Configuration dialog box opens (Figure 9-6).

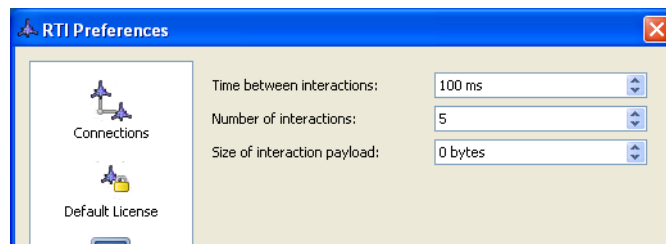


Figure 9-5. RTI Preferences dialog box, Latency Testing page

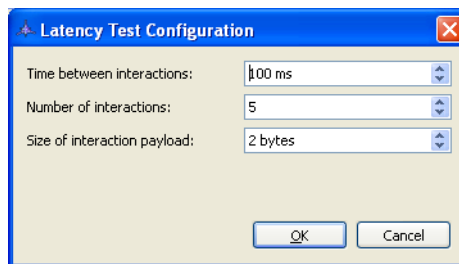


Figure 9-6. Latency Test Configuration dialog box

2. Edit any of the following parameters:
 - Time between interactions — the amount of time, in milliseconds, to wait between sending test interactions
 - Number of interactions — the number of interactions to send per latency test
 - Size of interaction payload — the size, in bytes, of the interaction message to send.

9.3. The RTIspy Network Monitoring Tool

The RTIspy Network Monitoring Tool is a window into the “black box” of the RTI. It records each message sent and each API call made. You can use this information to help weigh the costs and benefits of different RTI features. It can also help you locate the bottlenecks in your simulations, whether they are due to the underlying network, processing in the RTI, processing in Federate Ambassador callbacks, or the simulation execution itself.

The RTIspy Network Monitoring Tool exposes the internal processing of the RTI in the following ways:

- The API Call Statistics page can show you which RTI Ambassador or Federate Ambassador calls might be affecting execution.
- You can observe how much data is sent across the network to represent the simulation’s objects and interactions. This can help identify possible problems and optimizations.
- You can log the statistics to a file, allowing deeper analysis after a simulation run.

9.3.1. Enabling Network Monitoring

The `RTI_enableNetworkMonitoring` parameter enables or disables the RTIspy Network Monitoring Tool. Network monitoring is enabled by default.

Enabling network monitoring may affect the performance of the RTI.

9.3.2. Logging Network Monitoring Statistics

The `RTI_logNetworkMonitorStatistics` parameter sets the initial state of network statistic logging for the RTIspy Network Monitoring Tool. It is disabled by default. The RTIspy Network Monitoring Tool must be enabled for this parameter to be relevant.

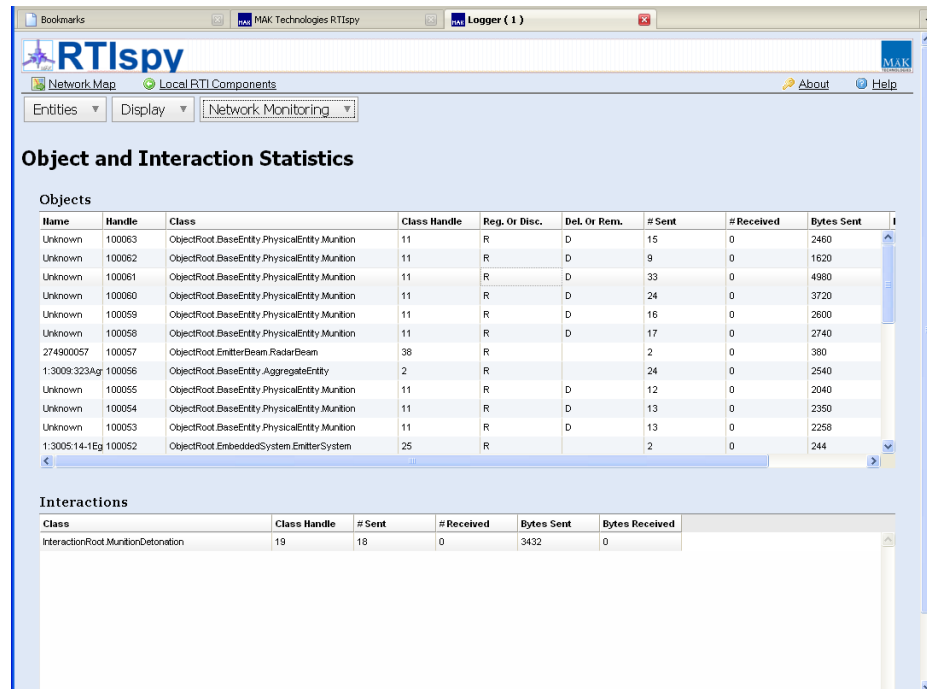
Logging network statistics imposes a performance penalty above and beyond basic network monitoring. Log files may grow quickly in large federations since each RTI operation is monitored.

If you enable logging, the RTIspy Network Monitoring Tool creates three log files, one for each page. The log files are named using the federate name and handle. The file extensions indicate the page from which the file was generated.

9.3.3. Displaying Object and Interaction Statistics

The Objects and Interaction Statistics page (Figure 9-7) lists the objects and interactions sent and received by a federate. You can sort the list using any of the columns to make it easier to distinguish the entries, for example sorting by Reg. Or Disc. Or Disc separates the Objects list into registered objects versus discovered objects.

- To display a list of all objects and interactions that have been detected by the monitored federate, choose **Network Monitoring** → **Objects and Interactions**.



Object and Interaction Statistics

Objects

Name	Handle	Class	Class Handle	Reg. Or Disc.	Del. Or Rem.	# Sent	# Received	Bytes Sent
Unknown	100063	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	15	0	2460
Unknown	100062	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	9	0	1620
Unknown	100061	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	33	0	4980
Unknown	100060	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	24	0	3720
Unknown	100059	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	16	0	2600
Unknown	100058	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	17	0	2740
274900057	100057	ObjectRoot.EmitterBeam.RadarBeam	38	R		2	0	380
1:3009:323Ag	100056	ObjectRoot.BaseEntity.AggregateEntity	2	R		24	0	2540
Unknown	100055	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	12	0	2040
Unknown	100054	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	13	0	2350
Unknown	100053	ObjectRoot.BaseEntity.PhysicalEntity.Munition	11	R	D	13	0	2258
1:3005:14-1Eg	100052	ObjectRoot.EmbeddedSystem.EmitterSystem	25	R		2	0	244

Interactions

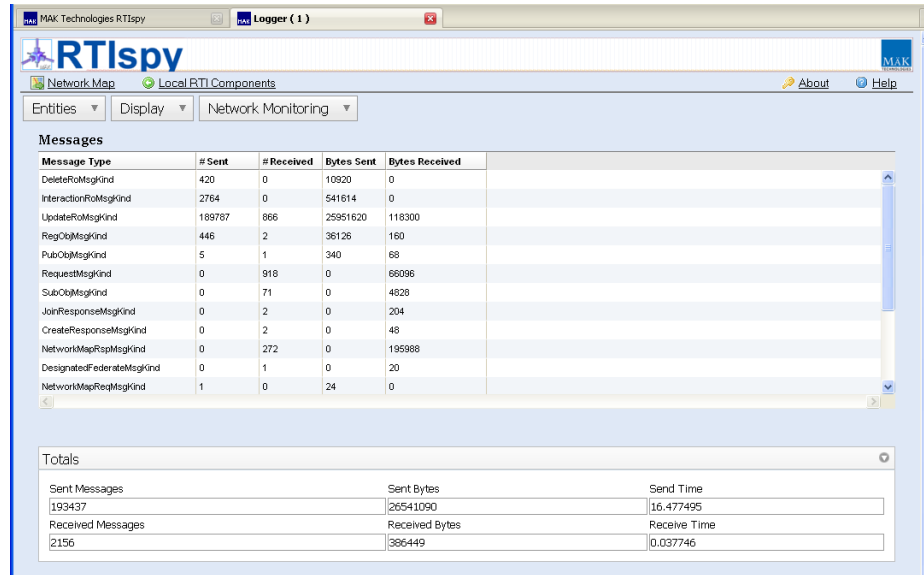
Class	Class Handle	# Sent	# Received	Bytes Sent	Bytes Received
InteractionRoot.MunitionDetonation	19	18	0	3432	0

Figure 9-7. Object and Interaction Statistics page

9.3.4. Displaying RTI Messages

The Messages page (Figure 9-8) lists the RTI message types and shows how often each message type has been sent and received and how large the messages are. It also provides summary statistics on the number of messages sent and received and their sizes.

- To display the complete list of RTI messages sent and received by the federate, choose **Network Monitoring** → **Messages**.



The screenshot shows the RTIsPy application window with the 'Messages' page selected. The page displays a table of message statistics and a 'Totals' summary section.

Message Type	# Sent	# Received	Bytes Sent	Bytes Received
DeleteFoMsgKind	420	0	10820	0
InteractionFoMsgKind	2764	0	541614	0
UpdateFoMsgKind	189787	866	25951620	118300
RegObjMsgKind	446	2	36126	160
PubObjMsgKind	5	1	340	68
RequestMsgKind	0	918	0	66096
SubObjMsgKind	0	71	0	4828
JoinResponseMsgKind	0	2	0	204
CreateResponseMsgKind	0	2	0	48
NetworkMapRespMsgKind	0	272	0	195988
DesignatedFederateMsgKind	0	1	0	20
NetworkMapReqMsgKind	1	0	24	0

Totals		
Sent Messages	Sent Bytes	Send Time
193437	26541090	16.477495
Received Messages	Received Bytes	Receive Time
2156	386449	0.037746

Figure 9-8. Messages page

9.3.5. The RTI API Call Statistics Page

The RTI API Call Statistics page (Figure 9-9) displays a list of all the calls made to the RTI Ambassador and the calls that the federate has made to the Federate Ambassador. Observing the amount of time spent on calls that the federate makes can help you locate bottlenecks in your application's use of the RTI and in your callbacks.

- To display the API Call Statistics page, choose **Network Monitoring** → **API Call Statistics**.

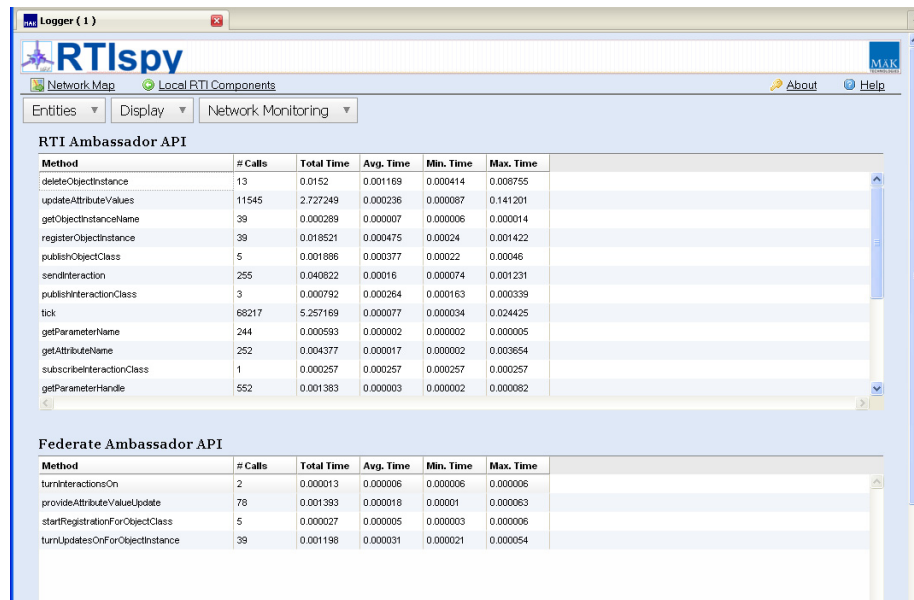


Figure 9-9. API Call Statistics page

9.4. Examples of Using the RTIspy Network Monitoring Tool

The examples in this section show you how you can use the RTIspy Network Monitoring Tool to help resolve common problems.

9.4.1. Finding Bottlenecks Caused by Processing Callbacks

If you suspect that your application is spending too much time processing callbacks, the RTIspy Network Monitoring Tool can help you find the bottlenecks.

To locate bottlenecks caused by processing callbacks:

1. Choose **Network Monitoring** → **API Call Statistics** (Figure 9-10).

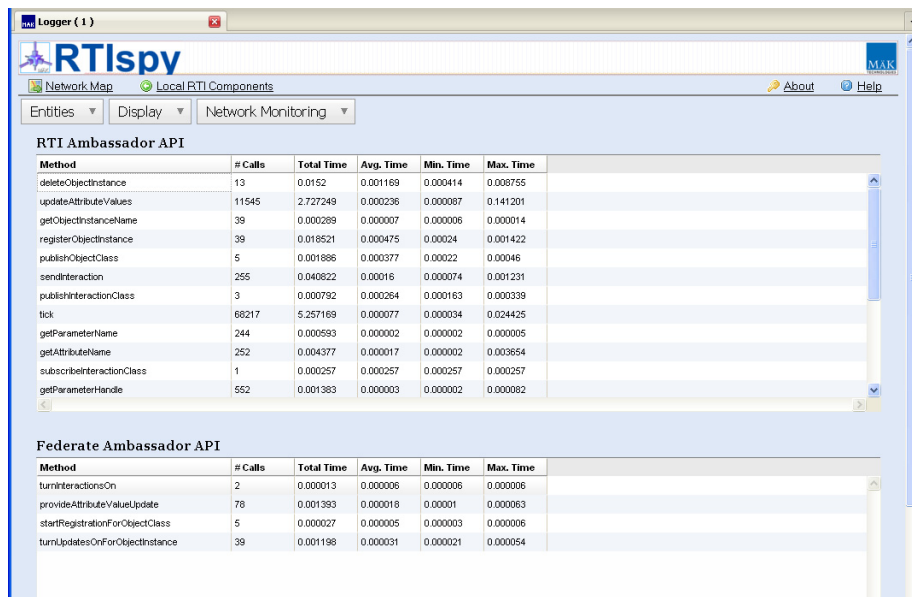


Figure 9-10. API Call Statistics view

2. To locate any callbacks that are consuming resources, look at the Federate Ambassador API callback list and sort it by Total Time.
3. Review your callback code and look for optimizations. You might consider moving the processing of the callback into a separate thread to isolate the network I/O and simulation processing portions of your code (at the cost of greatly increased complexity).

9.4.2. Identifying the Cause of Slow-Downs Under Time Management

Suppose your simulation uses time management to synchronize its federates. If the federates advance through time at different rates, there may be an apparent slow-down in the federates that take larger time steps. For example, if one set of federates requests time advances of 10 seconds per step and a second set requests time steps of 1 millisecond per step, then the 10 second per step federates, which on their own would advance at a rate far greater than real-time, will appear to be slowing down. It may not be obvious what has caused the slowdown. The RTIsPy Network Monitoring Tool and the RTI Federations View can help you to identify the problem.

To locate slowdowns caused by time management:

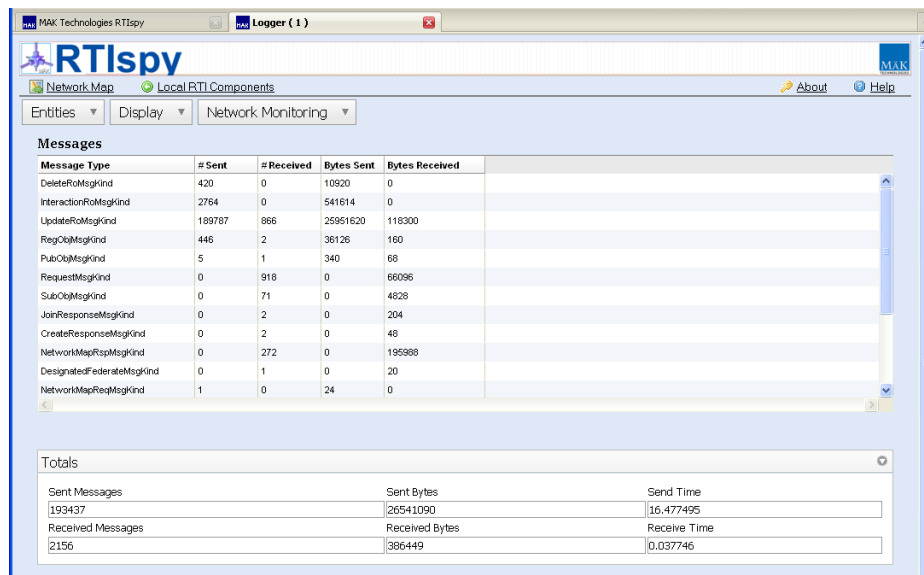
1. In the web page for one of the 10 second per step federates, open the API Call Statistics page.
2. Sort the RTI Ambassador API list Total Time. (Click the heading for the column.) This will show that almost all of the federate's time is spent processing tick() calls. This is a sign that your simulation is blocked and waiting for an RTI event to proceed – in this case a Time Advance Grant.
3. In the RTI Federations View, choose **View** → **Federation** → *federation_name*. The federation toolbar opens. You should see the 1 millisecond per-step federates advancing through time while the 10 second per step federates advance approximately every 10 seconds. This is because they are constrained by the 1 millisecond federates and can only advance when the 1 millisecond federates have stepped 10,000 times.

9.4.3. Comparing the Costs and Benefits of RTI Features

The RTIs Spy Network Monitoring Tool can help you to weigh the costs and benefits of various RTI features. Some of the more advanced RTI features, such as MOM or DDM, offer a great deal of functionality, but for some simulations, the costs in terms of additional message passing and RTI processing outweighs the benefits. For example, the use of DDM requires the RTI to exchange additional messages for default region information even if simple DM calls are being used.

To examine the benefits of RTI features:

1. Run the rtiexec and a simple federate without DDM enabled.
2. Choose **Network Monitoring** → **Messages** (Figure 9-11).

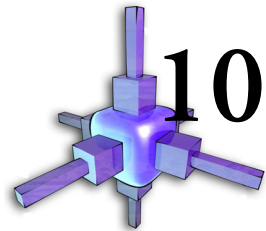


Message Type	# Sent	# Received	Bytes Sent	Bytes Received
DeleteReqMsgKind	420	0	10920	0
InteractionReqMsgKind	2764	0	541614	0
UpdateReqMsgKind	199787	866	25951620	118300
RegObjMsgKind	446	2	36126	160
PubObjMsgKind	5	1	340	66
RequestMsgKind	0	918	0	66096
SubObjMsgKind	0	71	0	4828
JoinResponseMsgKind	0	2	0	204
CreateResponseMsgKind	0	2	0	48
NetworkMapRspMsgKind	0	272	0	195388
DesignatedFederateMsgKind	0	1	0	20
NetworkMapReqMsgKind	1	0	24	0

Totals		
Sent Messages	Sent Bytes	Send Time
193437	26541090	16.477495
Received Messages	Received Bytes	Receive Time
2156	386449	0.037746

Figure 9-11. Messages page

3. Run the federate with DDM enabled.
4. Even though DDM has no effect on a simple simulation, you will still see additional messages passed such as AssocPubRegMsgKind, which are required by the DDM implementation.



Managing Network Traffic

This chapter describes a variety of ways to configure how the RTI manages network traffic. It does not cover all aspects of network configuration. For a detailed discussion of packet forwarding strategies, including distributed forwarding, please see Chapter 11, *Configuring Message Forwarding*.

Choosing Best Effort or Reliable Transport	10-2
Packet Bundling for Reliable and Best Effort Messages.....	10-3
Automatically Locating the RTI Forwarder and rtiexec	10-3
Configuring Sender-Side Filtering (Smart TCP Forwarding).....	10-4
Configuring Sender-Side Filtering for rtiexec Messages.....	10-6
Compressing RTI Messages.....	10-6
Enabling Support for Large Attributes and Parameters.....	10-7
Configuring the Network Buffer Sizes.....	10-7
Choosing the Network Device to Use	10-8
Controlling Congestion for Best Effort Sockets.....	10-8
Configuring Asynchronous Processing	10-8
Buffering Automatic Update Requests (1516 and Evolved)	10-9
Filtering by Multicast Groups Using Declaration Management.....	10-9
Assigning DM Multicast Groups to Network Interfaces	10-10

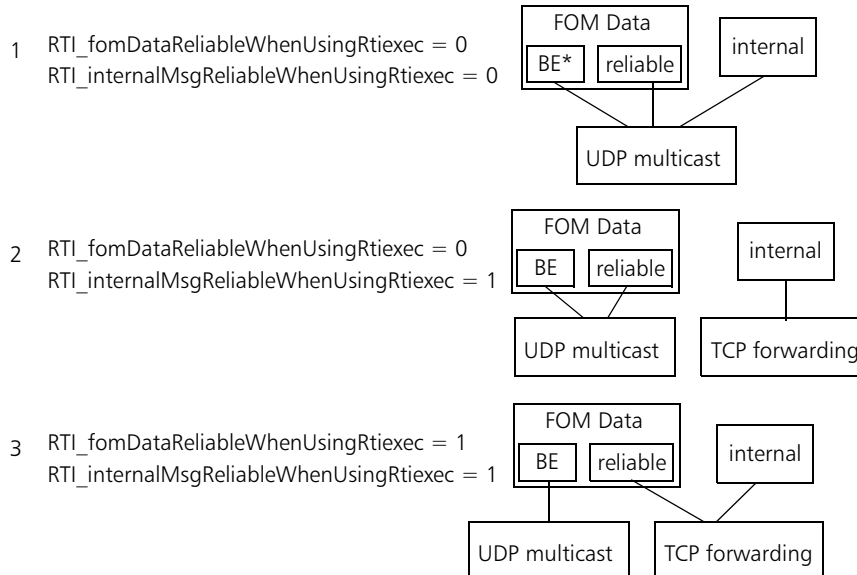
10.1. Choosing Best Effort or Reliable Transport

This section explains how to choose which type of network transport to use.

The type of message transport to be used for FOM data and internal messages is set by the `RTI_fomDataReliableWhenUsingRtiexec` and `RTI_internalMsgReliableWhenUsingRtiexec` parameters. If these parameters are set to 0, all messages are sent best effort (example 1 in [Figure 10-1](#)).

If you want the RTI's internal bookkeeping messages, such as federate subscription, to be sent using reliable transport, set `RTI_internalMsgReliableWhenUsingRtiexec` to 1 (example 2 in [Figure 10-1](#)).

In the FED file, you can specify how you want object attributes and interactions to be sent – reliable transport or best effort. If you want this data sent as stated in your FED file, set `RTI_fomDataReliableWhenUsingRtiexec` to 1 (example 3 in [Figure 10-1](#)).



*BE = best effort

Figure 10-1. Configuring the transport type



[Figure 10-1](#) does not illustrate use of an RTI Forwarder to forward UDP multicast traffic over a WAN. For details, please see “[Configuring UDP and Centralized TCP on a WAN](#),” on page 11-4.

10.2. Packet Bundling for Reliable and Best Effort Messages

The MÄK RTI supports message bundling. The RTI bundles messages into a single TCP packet or UDP datagram, up to the size limit configured via the `RTI_enablePacketBundling` parameter. Message bundling can significantly reduce the amount of bandwidth consumed by packet headers and the processor time spent waiting for network I/O calls. The recommended maximum bundle size is 1492 bytes for WAN traffic, or the maximum transmission unit (MTU) of the local network, if you are just running on a LAN.



Bundles are flushed at the end of every tick(), even if the bundle is not full.

The syntax for `RTI_enablePacketBundling` is as follows:

```
setqb RTI_enablePacketBundling size
```

where *size* is the maximum size of a bundled packet. *size* should be $\leq$ MTU of the network. If *size* is 0 (zero), then packet bundling is disabled. Default: 0.

10.3. Automatically Locating the RTI Forwarder and rtiexec

Using multicast discovery, the RTI can automatically locate an RTI Forwarder or `rtiexec` running on the LAN (that is, without specifying `RTI_tcpForwarderAddr`).

➤ To enable multicast discovery set `RTI_mcastDiscoveryEnabled` to 1.

With multicast discovery enabled, federates send a beacon to the multicast group selected by the `RTI_mcastDiscoveryAddrString` parameter. If a RTI Forwarder is running on the LAN, it responds with the correct address. The federate will continue trying to discover a forwarder until it reaches the limit specified by the `RTI_mcastDiscoveryTries` parameter. At that point, it defaults to best effort only.

You can use multicast discovery to ensure that only one RTI Forwarder and only one `rtiexec` are running on a LAN. This is particularly useful in cases where the `rtiexec` may run on different hosts from execution to execution.

10.4. Configuring Sender-Side Filtering (Smart TCP Forwarding)

Sender-side filtering (also called smart TCP forwarding) minimizes bandwidth by taking advantage of subscription information down to the level of each attribute instance, including taking HLA's DDM into account. If an attribute is not in scope for a receiving federate, it can be omitted from the transmission. Subscriptions are tracked at the federate level and a routing mask is appended to each update and interaction message. The routing mask is created based on the maximum number of federates permitted, as specified by the `RTI_maxNumFederates` parameter.

As you increase the number of federates permitted, the update and interaction message size increases. This leads to a possible trade-off between the cost of the small increase in message size and the lowered bandwidth achieved because entire transmissions may be filtered out.

Sender-side filtering is beneficial only if federates subscribe to a subset of the attributes being published. If a federate subscribes to all data, sender-side filtering can decrease performance. [Figure 10-2](#) compares TCP forwarding to sender-side filtering. When you use TCP forwarding without filtering, all federates receive all messages, all of which must travel over the LAN and WAN to the RTI Forwarder, then back to the other federates. When you use sender-side filtering, all federates still send updates through the RTI Forwarder. However, they receive updates based on their subscriptions. In the figure, M1A2 federates only receive updates from other M1A2 federates, in this case one update each. The F18 federate only receives updates from other fixed-wing entities, in this case, none. The COM federate receives all updates.



- ♦ If you use sender-side filtering, the rtiexec can support only one federation at a time.
- ♦ Sender-side filtering is not compatible with fixed grid DDM.

To configure sender-side filtering, edit the `RTI_smartForwardingLevel` and `RTI_maxNumFederates` parameters in *rid.mtl*.

```
(setqb RTI_smartForwardingLevel 0)
(setqb RTI_maxNumFederates 100)
```

`RTI_smartForwardingLevel` specifies the TCP forwarding mode, where:

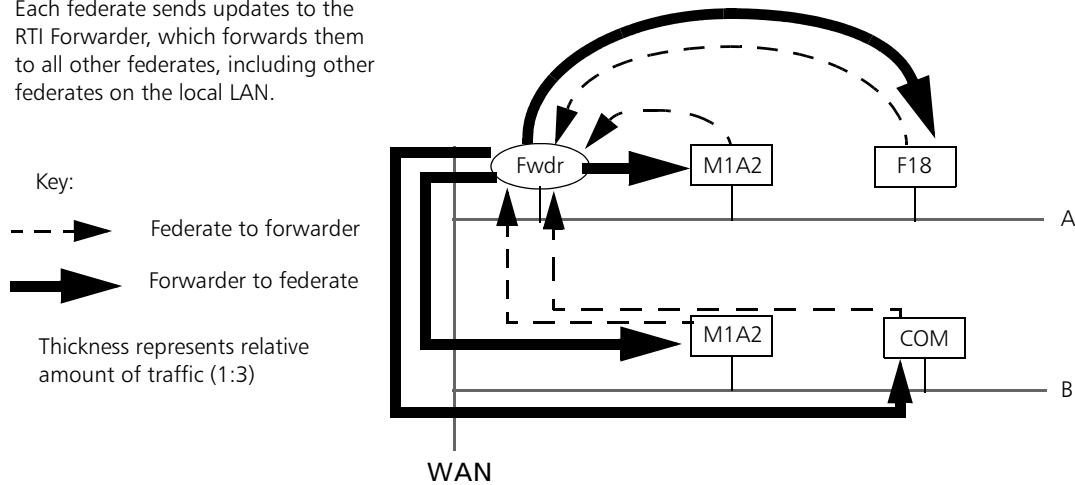
- ♦ 0 — basic TCP forwarding (no subscription-based routing or filtering)
- ♦ 1 — enable subscription based routing
- ♦ 2 — enable subscription based routing and disable send for updates and interactions that have no subscribers.

Default: 0.

RTI_maxNumFederates specifies the maximum number of federates allowed in a federation at any one time. This parameter only applies when sender-side filtering is enabled. Each update and interaction message has a bitmask of at least this number of bits appended, so increasing the number of federates increases the overhead. Default: 100.

TCP forwarding

Each federate sends updates to the RTI Forwarder, which forwards them to all other federates, including other federates on the local LAN.



Sender-side filtering

Each federate sends updates through the RTI Forwarder. Only updates of interest are sent to receiving federates.

- M1A2s subscribe only to ground entity updates.
- F18 subscribes only to fixed-wing entity updates.
- COM subscribes to all updates.

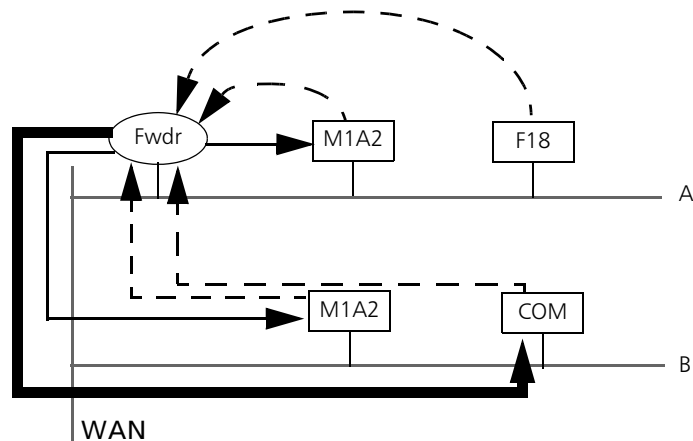


Figure 10-2. TCP forwarding compared to sender-side filtering

10.4.1. Configuring Sender-Side Filtering for rtiexec Messages

By default, the RTI Forwarder sends internal messages only to the rtiexec (but not to federates), and FOM data only to federates (but not to the rtiexec). This filtering can significantly reduce the number of internal message transmissions, reducing both bandwidth and network overhead. In those federations in which internal RTI messages consume a significant portion of network resources this can lead to a substantial, federation-wide performance improvement. You enable or disable this message filtering with the `RTI_enableFedexMsgRouting` parameter.

10.5. Compressing RTI Messages

The RTI supports compression and decompression of RTI messages (including the header and payload). You can enable compression separately for UDP (best effort) and TCP (reliable). The amount of compression applied can also be controlled with separate configuration parameters.



While you can connect a socket with compression enabled to a non-compressing socket, the RTI messages will be unintelligible to each endpoint. The RID consistency checker cannot determine if there is a mismatch in the status of TCP compression between federates, because if there is a mismatch it will not be able to establish a connection over which to exchange RID file consistency messages.

To configure compression, use the following parameters in *rid.mtl*:

- ♦ **RTI_enableTcpCompression.** Enables compression of RTI message payloads transmitted over TCP connections. Consistency checking is enforced on this parameter.
- ♦ **RTI_tcpCompressionLevel.** Specifies the amount of compression applied to RTI message payloads over TCP connections as follows:
 - 1 — minimal compression.
 - 2 through 8 — increasing levels of compression.
 - 9 — maximal compression.
 - 10 — maximal compression of packet bundles (super bundling). This is only valid when packet bundling is enabled.
- ♦ **RTI_enableUdpCompression.** Enables compression of RTI message payloads transmitted in UDP datagrams. Consistency checking is enforced on this parameter.
- ♦ **RTI_udpCompressionLevel.** Specifies the amount of compression applied to RTI message payloads in UDP datagrams, using the same options as `RTI_tcpCompressionLevel`.



If you want to use the RTISpy API to override the socket compression implementation, please contact MÄK technical support for assistance.

10.6. Enabling Support for Large Attributes and Parameters

The `RTI_use32BitsForValueSize` parameter enables or disables support for sending update and interaction messages that have attributes and parameters that are larger than 64 KB. The default message format uses 16-bit integers (maximum message size 64 KB) to represent the size of attribute and parameter data elements. When you enable this parameter, the message format is modified to use 32-bit integers for each value. Any resulting large messages (> 64KB) must be sent using TCP. The fragmentation and reassembly of messages is only supported by the RTI reliable connection that uses TCP. The best effort connection does not support data fragmentation other than what is provided by the underlying UDP/IP, and it is limited to 64K packets. You can send messages using best effort when this parameter is enabled, but the messages must be smaller than 64K.

You must adjust the network buffer sizes to accommodate the largest message that will be transmitted.

10.7. Configuring the Network Buffer Sizes

You can configure the TCP and UDP network receive buffers, the send buffer, and the TCP buffer size. If you enable big messages, you must adjust the network buffer sizes to accommodate the largest message that will be transmitted.

The `RTI_socketReceiveBufferSize` parameter configures the size of the TCP and UDP network receive buffers. Increasing the size can increase the RTI's tolerance of peak network loading. This buffer size should be large enough to hold several of the largest messages. Default: 50K.

```
(setqb RTI_socketReceiveBufferSize buffer_size)
```

The `RTI_socketSendBufferSize` parameter configures the socket send buffer size. This buffer size should be large enough to hold the largest message to be sent. Default: 50K.

```
(setqb RTI_socketSendBufferSize buffer_size)
```

The `RTI_tcpBufferSize` sets the size of the internal buffer into which TCP packets get read. This buffer size should be large enough to hold the largest message to be received. Default: 50K.

```
(setqb RTI_tcpBufferSize buffer_size)
```

10.8. Choosing the Network Device to Use

To set the network device to use with multicast traffic, use the `RTI_mcastDevAddrString` parameter. The syntax is:

```
(setqb RTI_mcastDevAddrString "IP_address")
```

where *IP_address* is the IP address of the ethernet device you want to use with multicast traffic.

10.9. Controlling Congestion for Best Effort Sockets

When `RTI_enableBestEffortSendRetry` is enabled, the RTI tries to provide limited congestion control for best effort sockets. To configure congestion control, set `RTI_bestEffortSendRetries` and `RTI_bestEffortSendRetryWaitUsec`. `RTI_bestEffortSendRetries` specifies the number of send attempts before discarding the message.

`RTI_bestEffortSendRetryWaitUsec` specifies the number of microseconds to wait after each send attempt for a blocked socket.

`RTI_bestEffortSendRetries` and `RTI_bestEffortSendRetryWaitUsec` are advanced network configuration parameters. When used together, they provide limited congestion control for best effort sockets. `RTI_bestEffortSendRetries` specifies the number of send attempts before discarding the message. `RTI_bestEffortSendRetryWaitUsec` specifies the number of microseconds to wait after each send attempt for a blocked socket.

10.10. Configuring Asynchronous Processing

The `RTI_processingModel` parameter specifies the processing model, as follows:

- ♦ 0 (zero) — synchronous mode.
- ♦ 1 — asynchronous I/O (default). The RTI processes network reads and writes in a secondary thread. Asynchronous I/O can improve performance in federations that often produce bursts of network traffic and may reduce the number of dropped messages.
- ♦ 2 — asynchronous message processing. The RTI processes messages in the asynchronous thread, but it does not invoke any resulting callbacks. This processing mode may improve the response and efficiency of some distributed algorithms (for example, time management). However, it requires the caching of callback parameters, which results in additional memory allocation and data copy operations.
- ♦ 3 — asynchronous callbacks. The RTI performs federate callbacks in the asynchronous thread. Asynchronous callbacks may offer performance benefits to certain classes of federate. However, asynchronous callbacks significantly increase the complexity of federates, because the federate designer must ensure that all calls to the Federate Ambassador are fully thread-safe.

10.11. Buffering Automatic Update Requests (1516 and Evolved)

HLA 1516 and HLA Evolved federates can instruct the RTI to buffer automatic update requests temporarily, to accumulate automatic requests from multiple federates. This can reduce the number of automatic provideAttributeUpdate callbacks required for changes in publication and subscription. To enable this feature, set `RTI_messageThrottling` and `RTI_autoProvideDelay` to 1.

10.12. Filtering by Multicast Groups Using Declaration Management

You can assign object and interaction classes to separate multicast groups to enable basic filtering by multicast groups using declaration management (DM) services. Use the RID file parameters, `RTI-addDMObjectMulticastAddr` and `RTI-addDMInteractionMulticastAddr`.

The syntax for the parameters is:

```
(RTI-addDMObjectMulticastAddr "ObjectClassName" "MulticastAddress"
  "NestingOperator" "interface")
(RTI-addDMInteractionMulticastAddr "InteractionClassName"
  "MulticastAddress" "NestingOperator" "interface")
```

where:

- *ObjectClassName* is a fully qualified FOM class of the form `"ObjectRoot.BaseEntity.Platform"`.
- *MulticastAddress* is a valid multicast address of the form `"229.7.7.9"`.
- *NestingOperator* is a quoted string containing either `exclusive` or `inclusive`. The nesting operator values determine whether the assignment of the multicast address will apply to the specific class exclusively or to that class and all derived classes.
- *interface* specifies a network interface. For details, please see [“Assigning DM Multicast Groups to Network Interfaces,”](#) on page 10-10.

Organize entries in the order of base classes to leaf classes. Exclusive entries must be placed after all inclusive entries from which the class inherits.

In the following example, the address 229.7.7.9 is assigned to all classes derived from `"ObjectRoot.BaseEntity.Platform"` except for `"ObjectRoot.BaseEntity.Platform.GroundVehicle"`, which is assigned 229.7.7.10:

```
(RTI-addDMObjectMulticastAddr "ObjectRoot.BaseEntity.
  Platform" "229.7.7.9" "inclusive" "0.0.0.0")
(RTI-addDMObjectMulticastAddr "ObjectRoot.BaseEntity.
  Platform.GroundVehicle" "229.7.7.10" "exclusive" "0.0.0.0")
```

Using these parameters will result in the following behavior:

- ♦ A federate subscription causes the LRC to join all multicast groups specified for the subscribed class and its derived classes. For example, given the previous example, a subscription to `ObjectRoot.BaseEntity.Platform` would result in the LRC joining multicast groups 229.7.7.9 and 229.7.7.10 on 0.0.0.0, the default interface.
- ♦ The LRC transmits object attributes to the multicast group specified by the class of the object instance (not the class where the attributes are defined). Given the previous example, the LRC transmits a `WorldLocation` attribute (defined in class *BaseEntity*) of an object instance of class `ObjectRoot.BaseEntity.Platform.GroundVehicle` to the multicast group 229.7.7.10. It would transmit the same attribute of an object instance of class `ObjectRoot.BaseEntity.Platform` to the multicast group 229.7.7.9.
- ♦ The LRC transmits interactions to the multicast group specified by the interaction class.



The functionality described in this section is separate from any filtering that the DDM services might perform.

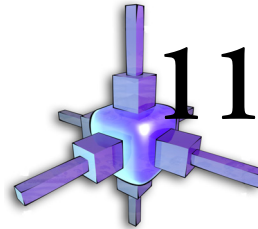
10.12.1. Assigning DM Multicast Groups to Network Interfaces

The `RTI-addDMObjectMulticastAddr` parameter has a parameter that assigns DM multicast groups to specific network interfaces. If you have multi-homed machines, this parameter allows you to segment RTI traffic onto multiple physical networks without modifying your system's routing tables. For example, if a multi-homed machine has network interfaces with address 10.0.1.211 and 192.168.1.211, you can send all HLA object traffic to the 10.0.1.x network as follows:

```
(RTI-addDMObjectMulticastAddr "ObjectRoot" "224.0.0.1" "inclusive"
 "10.0.1.211")
```

The `RTI-addDMInteractionMulticastAddr` parameter has a similar parameter. For example, if a multi-homed machine has network interfaces with address 10.0.1.211 and 192.168.2.211 you can send all HLA interaction traffic to the 192.168.1.x network using the following setting:

```
(RTI-addDMInteractionMulticastAddr "InteractionRoot" "224.0.0.2"
 "inclusive" "192.168.2.211")
```



Configuring Message Forwarding

The MÄK RTI offers a variety of options for configuring how messages are forwarded to federates. The first part of this chapter discusses networking on LANs and WANs using UDP or centralized forwarding. The rest of the chapter describes how to configure distributed forwarding. Use of distributed forwarding can significantly reduce network traffic over a LAN or WAN.

Configuring UDP and Centralized TCP Forwarding on a LAN	11-3
Configuring Best Effort (UDP) on a LAN.....	11-3
Configuring Centralized TCP Forwarding on a LAN	11-4
Configuring UDP and Centralized TCP on a WAN	11-4
Configuring Centralized TCP Forwarding	11-5
Configuring UDP Over a WAN without Distributed Forwarding	11-6
Distributed Forwarding Topologies on a LAN	11-7
Forwarder Groups	11-7
Singleton Forwarders.....	11-9
Advanced Distributed Forwarding on a LAN	11-10
Distributed Forwarding Topologies on an WAN.....	11-11
Optimizing Distributed Forwarding across a WAN	11-12
Configuring Federates for Distributed Forwarding.....	11-15
Configuring RTI Forwarders for Distributed Forwarding.....	11-16
Configuring RTI Forwarders Using the RTI Assistant	11-17
Creating a Routes File	11-18
Optional RTI Forwarder Configuration Parameters.....	11-20
Configuring Single-Portal Forwarder Groups	11-21
Configuring Load-Balanced Forwarder Groups	11-23
Configuring Distributed UDP Forwarding.....	11-24
Configuring Routers or Firewalls for WAN Federations	11-26

Configuring Message Forwarding

Special Distributed Forwarder Networks.....	11-26
Configuring Hierarchical Forwarder Networks.....	11-27

11.1. Configuring UDP and Centralized TCP Forwarding on a LAN

As discussed in [“Network Topologies for RTI Messages,”](#) on page 2-7, the MÄK RTI supports sending messages over a LAN via best effort, centralized forwarding, and distributed forwarding. For details about configuring distributed forwarding, please see [“Distributed Forwarding Topologies on a LAN,”](#) on page 11-7 and the sections that follow it.

11.1.1. Configuring Best Effort (UDP) on a LAN

To configure best effort communications, set `RTI_fomDataReliableWhenUsingRtiexec` or `RTI_internalMsgReliableWhenUsingRtiexec`, or both, to 0, as described in [“Choosing Best Effort or Reliable Transport,”](#) on page 10-2. Optionally, specify the following parameters:

- ♦ `RTI_udpPort`. Specifies the port number to be used for best effort messages sent among federates, and TCP messages sent between federates and the `rtiexec`.
Default: 4000.
- ♦ `RTI_destAddrString`. Specifies the IP address to which best effort federation execution traffic is sent. This will typically be a broadcast address or multicast address.
Default: “229.7.7.7” (a multicast address). A value of “0.0.0.0” is a special value that indicates that the default broadcast address should be used. If any of your federates are running on platforms on which multicast is not supported, you must use a broadcast address rather than a multicast address.

11.1.2. Configuring Centralized TCP Forwarding on a LAN

By default, the RTI uses reliable transport and starts an RTI Forwarder when you start the `rtiexec`. In other words, the configuration instructions in this section are the default behavior of the RTI when you run an `rtiexec` using a connection configuration in the RTI Assistant.

To configure centralized TCP forwarding:

1. Enable use of reliable transport for internal messages or FOM data or both, by setting `RTI_fomDataReliableWhenUsingRtiexec` or `RTI_internalMsgReliableWhenUsingRtiexec`, or both, to 1.
2. Set `RTI_tcpForwarderAddr` to the IP address of the machine that is running the RTI Forwarder.

You must run the RTI Forwarder if you are using reliable transport (set `RTI_useRtiExec` to 1).

For example, if you want to use reliable transport for your internal messages and FOM data, and the machine that you are running the `rtiexec` on has the IP address 192.168.x.x, then your `rid.mtl` file should include the following:

```
(setqb RTI_useRtiExec 1)
(setqb RTI_internalMsgReliableWhenUsingRtiexec 1)
(setqb RTI_fomDataReliableWhenUsingRtiexec 1)
(setqb RTI_tcpForwarderAddr "192.168.x.x")
```



You can set the size of the buffer into which TCP packets get read. To set the buffer size, use the `RTI_tcpBufferSize` variable. The default size is 20000 bytes.

11.2. Configuring UDP and Centralized TCP on a WAN

You can use best effort or reliable transport over a WAN. The MÄK RTI supports various packet forwarding strategies for reducing the amount of bandwidth consumed in a WAN environment. The RTI Forwarder supports centralized TCP forwarding, distributed TCP forwarding, and distributed UDP forwarding. For details about configuring distributed forwarding on a WAN, please see [“Distributed Forwarding Topologies on a LAN,”](#) on page 11-7 and the sections that follow it.

11.2.1. Configuring Centralized TCP Forwarding

You can use the MÄK RTI's centralized TCP forwarding scheme in a WAN environment. This configuration is relatively simple to set up: as you would in a LAN, just set `RTI_tcpForwarderAddr` to the address of the RTI Forwarder.

Although centralized TCP forwarding is easy to configure, it is not particularly efficient. When you only have one RTI Forwarder, all TCP messages in the federation must travel through that central RTI Forwarder, even when both the sender and ultimate recipient are located on one LAN, and the RTI Forwarder is located on another. Obviously, this results in many more messages being sent across WAN links than are necessary. Therefore, the preferred method for running federations over a WAN is distributed forwarding.

Configuring Routers or Firewalls to Support WAN-based Federations

When you run a federation across a WAN, it is possible, and even likely, that the remote networks will be protected by firewalls or routers that perform Network Address Translation (NAT). The RTI Forwarder must be accessible to all other federate hosts or forwarders via the designated distributed forwarder port. Therefore, you will probably need to do some additional configuration to allow traffic to get through the firewalls to your RTI Forwarders. There are several options for doing this, including NAT with port forwarding, using a demilitarized zone (DMZ), virtual private networks (VPNs), static routing, and virtual LANs. You should consult with your network administrator and the administrators of other networks in your exercise to determine the best method to use. This section explains the port forwarding approach.

If you are using centralized TCP forwarding, you only need to configure one firewall – the one on the RTI Forwarder's LAN, as follows:

1. Make sure that all remote federates are configured to send TCP packets to the firewall's IP address, rather than the IP address of the RTI Forwarder's machine itself (which will not be reachable directly). In other words, set `RTI_tcpForwarderAddr` to the IP address of the firewall protecting the RTI Forwarder's LAN.
2. Configure that firewall so that it will forward TCP packets received on the RTI's TCP port to the machine that hosts the RTI Forwarder. (If you are using centralized TCP forwarding, then the RTI's TCP port will be the same as the RTI's UDP port).

If you are using distributed forwarding, please see [“Configuring Routers or Firewalls for WAN Federations,”](#) on page 11-26 for configuration details.

11.2.2. Configuring UDP Over a WAN without Distributed Forwarding

The preferred method of supporting best effort traffic over a WAN is to have a local RTI Forwarder listen to local multicast traffic, then forward those packets to remote RTI Forwarders via TCP. If you do not want to use the distributed forwarding method for UDP messages, you can use the strategy described in this section.

In this configuration, multiple copies of each message get sent over the WAN. The MÄK RTI supports the specification of multiple destination addresses for best effort packets. This is often necessary when communicating across a WAN, since most routers do not forward multicast packets.



If you are running two or more federates on the same computer and that computer is not available by multicast, then you must use the RTI Forwarder for best effort. This is because unicast UDP packets are only available to one application per IP:port pair. In other words, the procedure described in this section does not work if you run multiple federates on a computer.

To use this method, set the `RTI_distributedUdpForwarderMode` parameter to 0. Use `RTI_addDestAddrString` to add one or more destination addresses. All best effort data will be sent to each registered address. Configure a separate RID file for each LAN, such that a unicast address for each remote federate (or computer if a computer hosts multiple federates) is added as a destination. The default destination address (`RTI_destAddrString`) is automatically added to the list of destination addresses for LAN best effort communication, so do not add local federates.

For example, suppose two LANs have federates at the following IP addresses:

- ♦ LAN 1:
207.86.232.1
207.86.232.2
207.86.232.3
- ♦ LAN 2:
208.86.232.1
208.86.232.2
208.86.232.3

Add the following entries to the RID files:

- ♦ LAN 1 RID file
(RTI-addDestAddrString "208.86.232.1")
(RTI-addDestAddrString "208.86.232.2")
(RTI-addDestAddrString "208.86.232.3")
- ♦ LAN 2 RID file
(RTI-addDestAddrString "207.86.232.1")
(RTI-addDestAddrString "207.86.232.2")
(RTI-addDestAddrString "207.86.232.3")

11.3. Distributed Forwarding Topologies on a LAN

For very large federations that send large amounts of reliable data, a single RTI Forwarder may eventually limit performance of the federation, because it is responsible for sending a copy of each message to each federate. Running multiple RTI Forwarders can relieve this problem by allowing the burden of message distribution to be balanced among the forwarders. This preserves the benefits of employing RTI Forwarders while reducing the possibility of a forwarder overload.

If you use multiple RTI Forwarders, you can configure them in forwarder groups or as individual (singleton), fully connected forwarders. You can combine these two methods, where appropriate. For a discussion of when each alternative may be advantageous, please see [“Advanced Distributed Forwarding on a LAN,”](#) on page 11-10.

i

If you run multiple forwarders on a LAN-only configuration, you must disable distributed UDP forwarding. (Set `RTI_distributedUdpForwarderMode` to 0.) There is no benefit to distributed UDP forwarding in this situation because multicast and broadcast over a LAN are more efficient.

11.3.1. Forwarder Groups

You can set up RTI Forwarders to share LAN traffic by configuring them into a forwarder group. A forwarder group consists of a set of RTI Forwarders and all of the federates associated with them.

In a forwarder group, each federate is assigned to a primary forwarder. Federates send messages only to their primary forwarder. The primary forwarder relays the messages to all of the other federates in the forwarder group. Federates receive messages from all of the forwarders in the group. To achieve a good load sharing arrangement, you must assign federates to primary forwarders in a way that assures proper load dispersal. The forwarders in a group can also communicate with each other, primarily to exchange internal messages and messages destined for the `rtiexec` (which only maintains a connection to a single forwarder in the group).

[Figure 11-1](#) shows a forwarder group consisting of three RTI Forwarders, each of which serves several federates. The solid bi-directional lines represent connections between federates and their primary forwarders. The dashed uni-directional lines represent connections from the other (ancillary) forwarders. Federate B2 (shaded) sends a message to its primary RTI Forwarder (Forwarder B), which in turn sends it to all the other federates in the group (shaded lines).

i

The primary and ancillary forwarders are different for each federate. Forwarder A is the primary forwarder for federate A1, but is an ancillary forwarder to federate B1.

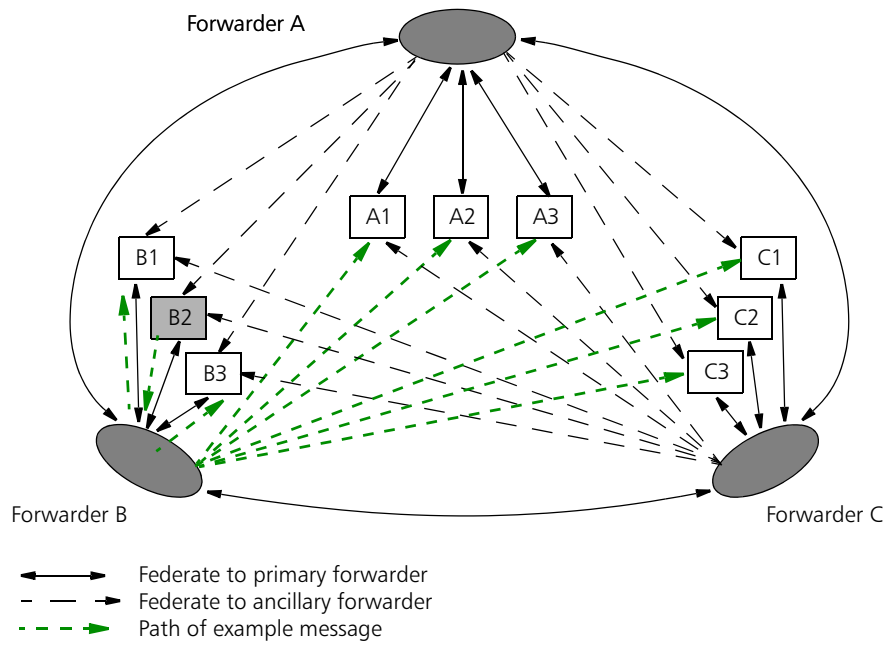


Figure 11-1. Distributed forwarding on a LAN with a forwarder group

11.3.2. Singleton Forwarders

If you do not want to create forwarder groups, you can connect federates to individual forwarders. These singleton forwarders can then be connected in a fully-connected configuration as shown in (Figure 11-2). This is known as a distributed singleton forwarder network. Each forwarder and its associated federates are essentially treated as a forwarder group that has one forwarder. The difference between this configuration and the forwarder group configuration is that the federates do not connect to any ancillary forwarders. So a messages sent by a federate (shaded federate and lines in Figure 11-2) must be relayed by its forwarder to the other forwarders, which then send it to the federates they are connected to.

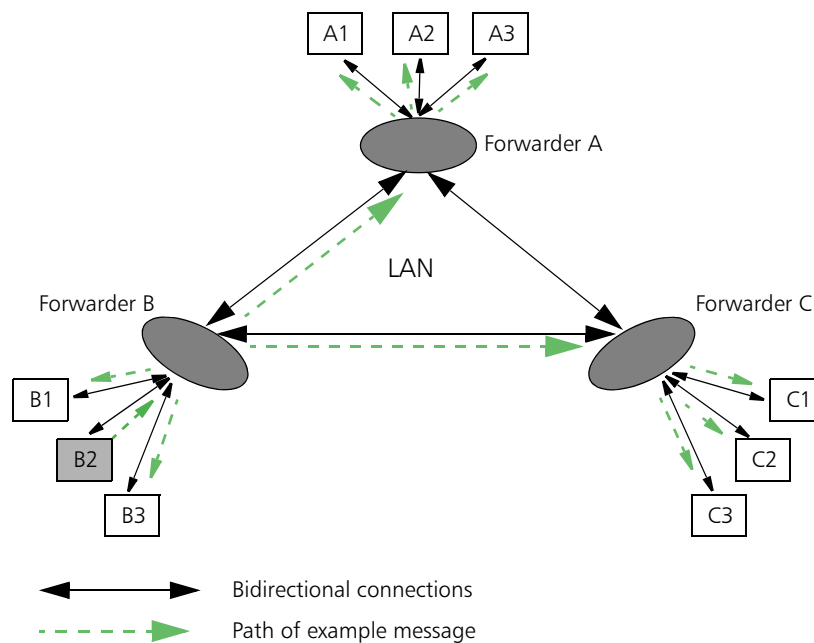


Figure 11-2. Distributed Singleton Forwarder Network on a LAN

11.3.3. Advanced Distributed Forwarding on a LAN

Forwarder groups and singleton forwarder networks both spread LAN traffic among multiple RTI Forwarders, thereby reducing the potential bottleneck of a centralized forwarder setup. In choosing which topology to use for your LAN, consider the following:

- Since group forwarders relay messages directly to the other federates on the LAN, group forwarding reduces latency (in most cases) by eliminating the hop between forwarders that is necessary if the distributed singleton forwarder network is used.
- Depending on the number of federates in a federation, a singleton forwarder network can have advantages over a forwarder network comprised of a single forwarder group.

Consider the case of a federation with a very large number of federates, for example 3000. If all of these are in a single forwarder group of 30 forwarders, in which each RTI Forwarder acts as the primary forwarder for 100 federates, each message from a federate must be retransmitted 2999 times by a primary forwarder in order to reach all 3000 federates. Since the primary forwarder likely only has one network interface over which to send these messages, the average latency is calculated as:

$$(t_{\text{msg}} * (\# \text{ of federates} / 2))$$

where t_{msg} is the time required to transmit the message once over the LAN.

In contrast, consider a distributed singleton forwarder network and the time required to distribute a message. The primary forwarder only has to relay the message 128 times (99 copies to the other federates it is connected to and 29 copies to the other RTI Forwarders). Since individual forwarders can probably transmit simultaneously if they are on a switched Ethernet backbone, all 30 forwarders could theoretically be distributing the message simultaneously to the federates connected to them. Potentially, the average latency is reduced to:

$$((t_{\text{msg}} * (\# \text{ of forwarders} / 2)) + (t_{\text{msg}} * (\# \text{ of federates per forwarder} / 2)))$$

The number of federates per forwarder is an order of magnitude less than the total number of federates. The time it takes for each forwarder to distribute the message to all federates connected to it is the largest contributor to the overall latency. Therefore, the latency should be reduced by an order of magnitude.

There is another way to configure large single-LAN federations. You can use one of the distributed forwarding methods used for sending reliable traffic over a WAN. Since the only difference between reliable communication over a LAN and over a WAN is the availability of bandwidth, it may be advantageous to use multiple forwarder groups on a LAN. Each group would consist of multiple RTI Forwarders that share the load and then interconnect over the LAN in a load-balanced configuration. This allows RTI Forwarders in disparate groups to handle message relay simultaneously within each group, thus potentially reducing latency even further.

There is no one correct topology. You must consider the unique circumstances of your federation when determining the best strategy for optimizing communication.

11.4. Distributed Forwarding Topologies on an WAN

The simplest distributed forwarding configuration for a WAN is to use individual (singleton), fully connected forwarders at each LAN. The forwarder at each LAN forwards messages to its local federates and between forwarders on remote LANs. The forwarders handle both reliable (TCP) and best effort (UDP) messages.

Similar to the centralized forwarder strategy, the sending federate passes the message only to the RTI Forwarder to which it is connected and the forwarder re-sends the message to all other federates connected directly to it (its federation subset). However, the RTI Forwarder also sends a copy of the message to all remote RTI Forwarders whose federation subsets contain federates that require the message. The remote forwarders distribute the message to all interested federates in their subset. (Figure 11-3) shows a distributed forwarding network. This figure is essentially identical to Figure 11-2. The only difference is that now the RTI Forwarders are on separate LANs.

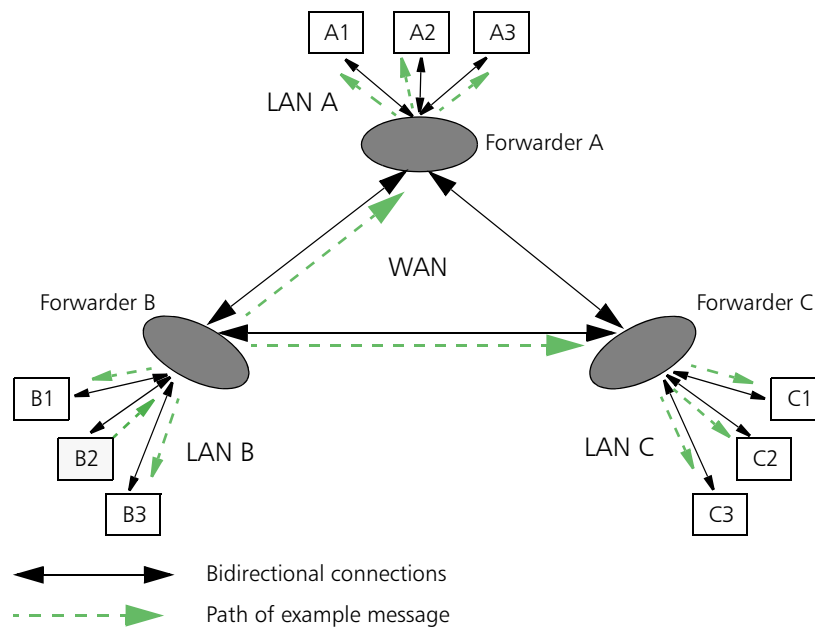


Figure 11-3. Distributed Singleton Forwarder Network on a WAN

11.4.1. Optimizing Distributed Forwarding across a WAN

Forwarding across a WAN can be optimized by adding more forwarders. To optimize distributed forwarding across a WAN, you would create one (or more) forwarder groups on each LAN. Each group can consist of one or more RTI Forwarders and individual groups can have different numbers of group members. A forwarder cannot be assigned to more than one group. A forwarder that is not explicitly assigned to any group is considered a singleton group (that is, a group consisting of a single member). There are two options available to interconnect the groups, single-portal and load-balance.



The simple configuration with a single forwarder at each LAN is a special case of interconnecting forwarder groups in which each LAN has a singleton group.

Single-Portal Interconnections

Using single-portal interconnections, one forwarder in each group is designated as the ‘portal’ forwarder. The portal forwarder acts as the entryway into the forwarder group. However, each forwarder in the group is responsible for forwarding its messages out of the group. To achieve this behavior, all local forwarders connect to the portal forwarder in foreign forwarder groups. Each forwarder then forwards messages from the federates they service to each foreign portal, which in turn relays the messages to all the federates in the portal forwarder’s local forwarder group.

The portals also handle messages destined for the rtiexec and the distribution of internal messages to the other non-portal forwarders within their groups. [Figure 11-4](#) illustrates a single-portal forwarder network. The solid lines represent bi-directional connections between two portals; the dashed lines represent connections from non-portal forwarders to portal forwarders. Non-portal forwarders send messages to the portals they are connected to, but the portals do not send messages directly to non-portals, only to the other portals. In a forwarder group network it is the groups that are considered fully-connected, not each individual forwarder.

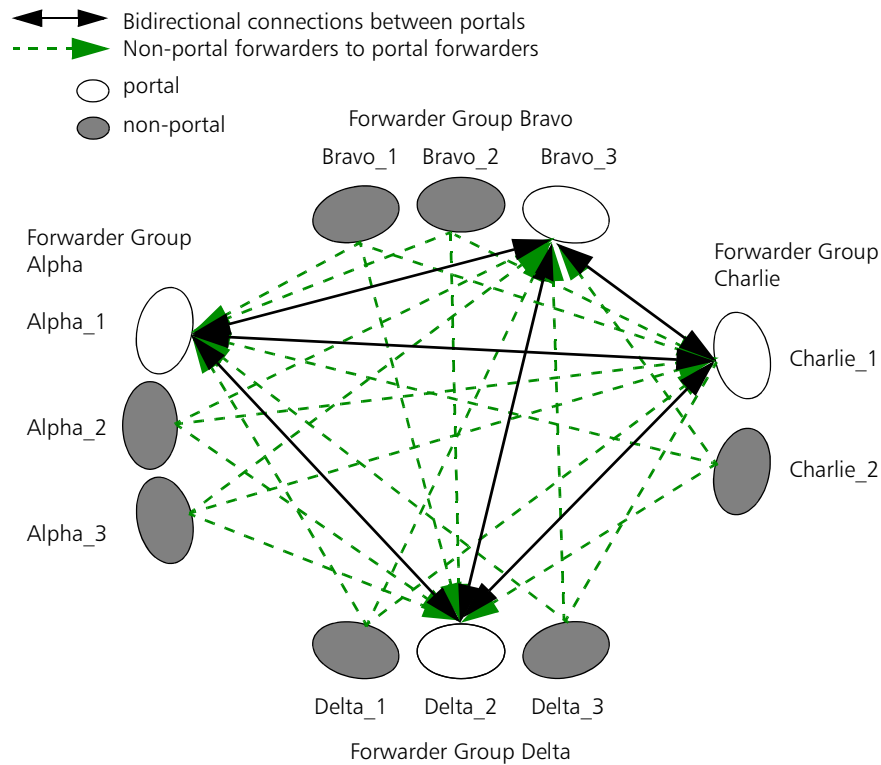


Figure 11-4. Distributed forwarder network (single-portal)

From a federate's perspective, the single-portal WAN configuration is similar to the LAN-based group forwarder strategy. The sending federate passes the message to its primary forwarder, which relays the message to all other federates in its forwarder group (also known as its federation subset). The RTI Forwarder also sends a copy of the message to a portal forwarder in each foreign forwarder group whose federation subsets contain federates that require the message. Those portal forwarders distribute the message to all interested federates in their group.

Load-Balanced Interconnections

In a load-balanced forwarder network, multiple forwarders (preferably all forwarders) in each group act as portals. There are multiple entry points into the group. Again however, each forwarder is responsible for forwarding messages out of the group. So instead of all local forwarders connecting to just a single portal forwarder in foreign groups, each local forwarder selects a different portal forwarder in each foreign group. In this arrangement, each local forwarder sends messages from the federates it services to a portal in each foreign forwarder group which in turn relays the messages to all the federates in its group. Use of multiple portals distributes the load of traffic sent between groups among several forwarders in each group, thus eliminating the potential bottleneck created by the single-portal scheme. Figure 11-5 illustrates a load-balanced configuration. In an ideal configuration, all groups have an equal number of forwarders and each forwarder has a bidirectional connection with a single forwarder in each foreign group.

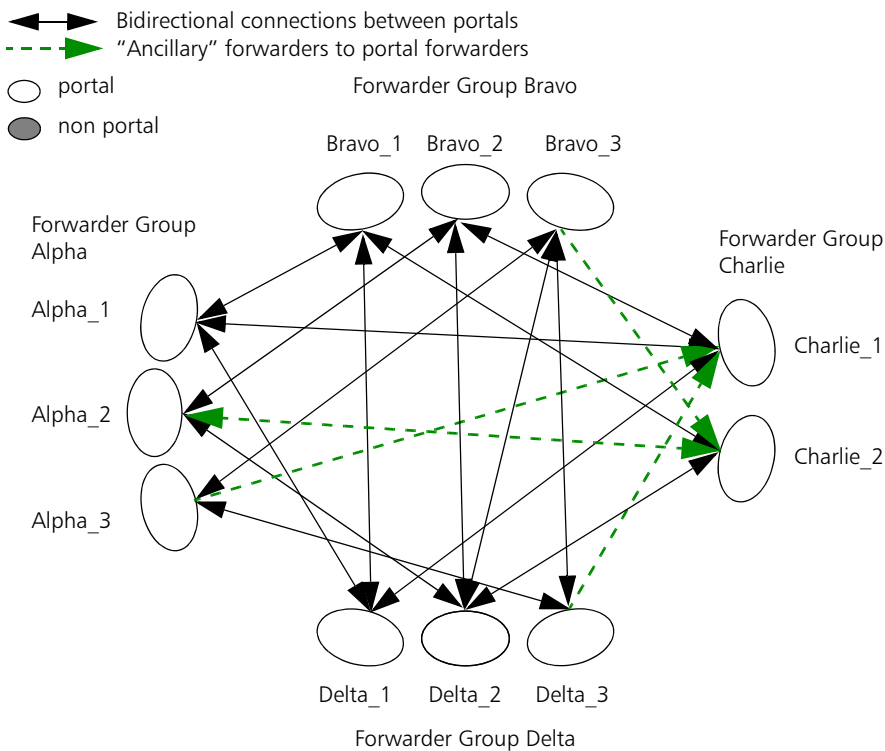


Figure 11-5. Distributed Forwarder Network (load-balance)

In [Figure 11-5](#), there is an unequal number of forwarders in one group. One or more forwarders in this smaller forwarder group must maintain connections with more than one forwarder in the larger foreign forwarder groups. However, a forwarder with multiple connections can have only one bidirectional connection to each foreign group. The other connections to a foreign group are treated like ancillary connections, from which it only receives messages. It sends messages only to the bidirectional connection, but receives them from both. The configuration of a forwarder forming multiple connections to a single foreign group is illustrated by forwarder group Charlie, which has only two members (compared to three for the other groups). The ancillary connection from each of the other groups is automatically distributed evenly between the two portals in the group so as to avoid overloading one of the portals.

11.5. Configuring Federates for Distributed Forwarding

You configure federates for distributed forwarding by editing the RID file on each computer on which federates are running.

To configure federates to use distributed forwarding:

1. Enable reliable transport, as described in “[Choosing Best Effort or Reliable Transport](#),” on page 10-2.
2. Specify the primary forwarder. In the RID file, set the `RTI_tcpForwarderAddr` parameter to point to the RTI Forwarder to use as the primary forwarder.
3. If you are using forwarding groups, specify the forwarders in the forwarding group. Add an `RTI-addForwarderGroupAddrString` parameter to the RID file for each RTI Forwarder in the forwarder group. If your network does not use forwarder groups, do not include this parameter in the RID file.
4. Set `RTI_distributedUdpForwarderMode` to 0, which disables distributed UDP forwarding.



When you use forwarder groups, each forwarder serves as a primary for a subset of the federates. As a result, the configuration will be different for federates with different primary forwarders.

11.6. Configuring RTI Forwarders for Distributed Forwarding

If you use a fully-connected forwarder network across a WAN, you can configure RTI Forwarders through the RTI Assistant. For other network topologies, RTI Forwarders are configured in the routes file (*routes.mtl*). You must copy the routes file to each computer that is running an RTI Forwarder. For a given distributed forwarding topology, the routes file is the same whether it is used for a LAN or WAN.

The RTI Forwarder determines how to function as follows:

- ♦ If you specify its connection using the RTI Assistant, it uses the selected connection configuration. It switches between distributed forwarding and centralized TCP forwarding dynamically as other forwarders connect to and disconnect from the network. It ignores the routes file even if it exists.
- ♦ If you specify use of the connection configured in the RID file by specifying the `-M` command-line option, or by setting `RTI_configureConnectionWithRid` to 1:
 - If there is a routes file, the RTI Forwarder uses distributed forwarding
 - If there is no routes file, it uses centralized TCP forwarding.

11.6.1. Configuring RTI Forwarders Using the RTI Assistant

To configure multiple forwarders using the RTI Assistant, you specify the network address of one of the forwarders in the Advanced Forwarder Options group box of the connection configuration dialog box. As you add RTI Forwarders, they dynamically share their network addresses to make connections.

To configure distributed forwarder connections:

1. When creating or editing a connection configuration, click the Advanced button. The Add New [Local rtiexec] Connection dialog box expands (Figure 11-6).

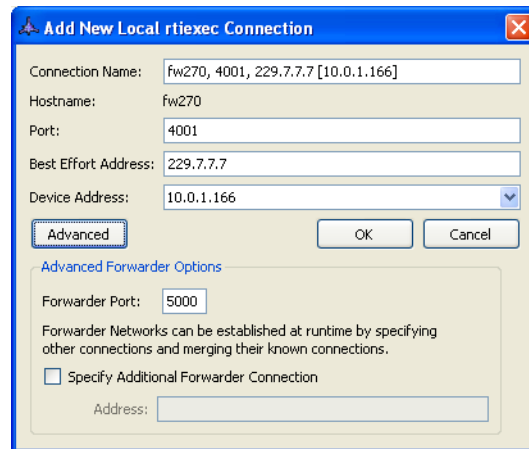


Figure 11-6. Expanded Add New Local rtiexec Connection dialog box

2. Optionally, specify a port for the RTI Forwarder.
3. Select Specify Additional Forwarder Connection.
4. Type the address of the RTI Forwarder that you want this forwarder to connect to. (It can be any forwarder on the network.)
5. Click OK.

11.6.2. Creating a Routes File

The routes file must specify each RTI Forwarder that will be part of the network (using `RTI_addForwarder`). The `RTI_distributedForwarderPort` parameter, in *rid.mtl*, specifies the port on which each RTI Forwarder will accept incoming connection attempts from other forwarders.

If you do not use forwarder groups, these are the only parameters required to construct a fully-connected forwarder network. A fully-connected network is one in which each forwarder maintains a connection to every other forwarder in the network, as illustrated in [Figure 11-4](#).

The list of forwarders should be identical in each routes file. The name:IP address pairs must be identical (that is, for a given IP address, the associated name must always be the same).

The syntax of the `RTI_addForwarder` parameter is:

```
(RTI_addForwarder "ForwarderName" "IP address | hostname")
```

where:

- *ForwarderName* is an arbitrary name assigned to a forwarder. It cannot contain spaces.
- *IP address* is the IP address of the computer on which the forwarder is running.
- *hostname* is the host name of the computer on which the forwarder is running.

Both input values must be enclosed by double-quotes (" ").



The use of hostnames rather than IP addresses in the routes file is strongly discouraged. Although specifying hostnames may work in some limited cases, in most WAN situations issues such as network address translation (NAT) traversal or VPN tunneling will lead to incorrect or unstable routing networks.

The following example is a routes file with three forwarders:

```
;; Forwarder List
;; List all Distributed Forwarder endpoint IPs here
(RTI_addForwarder "A" "192.168.1.1")
(RTI_addForwarder "B" "192.168.1.2")
(RTI_addForwarder "C" "192.168.1.3")
```

Table 11-1 describes all of the routes file parameters. Some of them are discussed in more detail in [“Optional RTI Forwarder Configuration Parameters,”](#) on page 11-20.

Table 11-1: Routes file parameters

Parameter	Description
RTI_forwarderInterconnectSetupTimeout	Specifies how long, in seconds, RTI Forwarders should try to initiate connections to other forwarders identified in the forwarder list. Default: 15.0 seconds.
RTI_addForwarder	Specifies the IP address (or hostname) of an RTI Forwarder to be added to the forwarder list.
RTI_addInterfaceAddr	Specifies the IP address of the host interface on which to try to initiate connections to other RTI Forwarders. Only necessary for multi-homed hosts.
RTI_allowMulticastForwarding	Specifies the forwarder name of an RTI Forwarder enabled to handle distributed UDP forwarding duties on its LAN. A separate entry for each RTI Forwarder that will act as the distributed UDP forwarder for its LAN is necessary. It is required only when forwarder groups are used and RTI_distributedUdpForwarderMode is 2.
RTI_forwarderGroupInterconnectMethod	Specifies the type of distributed forwarder network that will be created. Acceptable values are: ♦ “single-portal” ♦ “load-balance” ♦ “manual”. For information about single-portal and load-balanced networks, please see “Configuring Single-Portal Forwarder Groups,” on page 11-21 and “Configuring Load-Balanced Forwarder Groups,” on page 11-23. For information about manual configuration of forwarder networks, please see “Special Distributed Forwarder Networks,” on page 11-26.
RTI-addForwarderGroup	Specifies the name of a Forwarder Group that will be included in the distributed forwarder network.
RTI-addForwarderToGroup	Adds the named RTI Forwarder to the specified Forwarder Group.

Routes File Search Order

The search order for the routes file is as follows:

1. Load *routes.mtl* from the working directory.
2. Try to load the routes file from the directory specified by the `RTI_CONFIG` environment variable.
3. Use the default settings.

You can specify a different name for the routes file with the `RTI_forwarderRoutesFile` parameter.

11.6.3. Optional RTI Forwarder Configuration Parameters

This section describes in more detail some of the routes file parameters used to configure distributed forwarding.

Configuring the Connection Port and Timeout

During startup, each forwarder tries to connect to other forwarders. Then it waits for 15 seconds (by default) for other forwarders to connect to it. After the initial startup time elapses, an RTI Forwarder completes its initialization and starts handling incoming messages from federates. The forwarder accepts incoming connection attempts from late-starting forwarders for as long as it is executing.

In *rid.mtl*, specify the port used to connect to a forwarder with the `RTI_distributedForwarderPort` parameter. In *routes.mtl*, specify the amount of time spent in the initial startup period with the `RTI_forwarderInterconnectSetupTimeout` parameter.

Setting the timeout to 0.0 causes the RTI Forwarder to wait indefinitely for all forwarders to connect to the forwarder network. You can manually end the network establishment phase regardless of the configured timeout by typing `abort` or `a` and pressing **Enter** at the RTI Forwarder console.

UDP Forwarding

If you use distributed UDP forwarding to handle multicast traffic between LANs, you must add an `RTI_allowMulticastForwarding` parameter for the RTI Forwarder on each LAN that will monitor and relay those messages. The syntax is:

```
(RTI_allowMulticastForwarding "ForwarderName")
```

The *ForwarderName* string cannot contain spaces and must be enclosed by double-quotes (" "). Each name must be identical to a name specified in one of the `RTI_addForwarder` parameters.

11.6.4. Configuring Single-Portal Forwarder Groups

If you are using forwarder groups to manage network traffic, you must configure the group membership rosters and the interconnection between groups in the forwarder network.

The `RTI_forwarderGroupInterconnectMethod` parameter specifies how the RTI Forwarders will assemble themselves into a network. The syntax of the parameter is:

```
(setqb RTI_forwarderGroupInterconnectMethod {"single-portal" |
"load-balance" | "manual"})
```

where:

- `single-portal` specifies a single portal interconnection method, as described in [“Single-Portal Interconnections,”](#) on page 11-12. This is the default interconnect method.
- `load-balance` specifies a load balanced interconnection method, as described in [“Load-Balanced Interconnections,”](#) on page 11-14.
- `manual` specifies that interconnections are configured manually as described in [“Special Distributed Forwarder Networks,”](#) on page 11-26.

The value string must be enclosed in double-quotes (“ ”).

To configure the group membership rosters, create a group using the `RTI-addForwarder-Group` parameter. Then populate it with RTI Forwarders using the `RTI-addForwarderTo-Group` parameter. The RTI Forwarders must have already been specified in the forwarder list created by the `RTI_addForwarder` entries. The syntax of the two parameters is:

```
(RTI-addForwarderGroup "GroupName")
(RTI-addForwarderToGroup "GroupName" "ForwarderName" "URL"
acceptRemoteConnections)
```

where:

- `ForwarderName` must be identical to a name specified in one of the `RTI_addForwarder` parameters and it must appear before the group configuration section of the routes file.
- `URL` is currently not supported and should be left as an empty string (“”).
- `acceptRemoteConnections` specifies that the forwarder is a portal (1) or not a portal (0).

The `GroupName` and `ForwarderName` strings cannot contain spaces and must be enclosed by double-quotes (“ ”).

If none of the forwarders in a group is explicitly assigned to act as the portal for the group, the forwarder in the group whose name occurs first alphabetically acts as the portal.

The following example configuration shows the creation of four groups interconnected using the single-portal method (illustrated in [Figure 11-4](#)):

```
;; Forwarder Groups
(RTI-addForwarderGroup "Alpha")
```

```
(RTI-addForwarderToGroup "Alpha" "Alpha_1" "" 1)
(RTI-addForwarderToGroup "Alpha" "Alpha_2" "" 0)
(RTI-addForwarderToGroup "Alpha" "Alpha_3" "" 0)

(RTI-addForwarderGroup "Bravo")
(RTI-addForwarderToGroup "Bravo" "Bravo_1" "" 0)
(RTI-addForwarderToGroup "Bravo" "Bravo_2" "" 0)
(RTI-addForwarderToGroup "Bravo" "Bravo_3" "" 1)

(RTI-addForwarderGroup "Charlie")
(RTI-addForwarderToGroup "Charlie" "Charlie_2" "" 0)
(RTI-addForwarderToGroup "Charlie" "Charlie_1" "" 0)

(RTI-addForwarderGroup "Delta")
(RTI-addForwarderToGroup "Delta" "Delta_1" "" 0)
(RTI-addForwarderToGroup "Delta" "Delta_2" "" 1)
(RTI-addForwarderToGroup "Delta" "Delta_3" "" 0)

(setqb RTI_forwarderGroupInterconnectMethod "single-portal")
```

There is no portal explicitly identified in group Charlie. Charlie_1 is chosen by default to be the portal because it is first in its group alphabetically (even though it is listed last in the roster for the group.)

11.6.5. Configuring Load-Balanced Forwarder Groups

To optimize load balancing of network traffic, the forwarder groups must be configured such that each of the forwarders in the group is a portal. [Figure 11-5](#) depicts such a network. The configuration of that example is given below:

```
;; Forwarder Groups
(RTI-addForwarderGroup "Alpha")
(RTI-addForwarderToGroup "Alpha" "Alpha_1" "" 1)
(RTI-addForwarderToGroup "Alpha" "Alpha_2" "" 1)
(RTI-addForwarderToGroup "Alpha" "Alpha_3" "" 1)

(RTI-addForwarderGroup "Bravo")
(RTI-addForwarderToGroup "Bravo" "Bravo_1" "" 1)
(RTI-addForwarderToGroup "Bravo" "Bravo_2" "" 1)
(RTI-addForwarderToGroup "Bravo" "Bravo_3" "" 1)

(RTI-addForwarderGroup "Charlie")
(RTI-addForwarderToGroup "Charlie" "Charlie_2" "" 1)
(RTI-addForwarderToGroup "Charlie" "Charlie_1" "" 1)

(RTI-addForwarderGroup "Delta")
(RTI-addForwarderToGroup "Delta" "Delta_1" "" 1)
(RTI-addForwarderToGroup "Delta" "Delta_2" "" 1)
(RTI-addForwarderToGroup "Delta" "Delta_3" "" 1)

(setqb RTI_forwarderGroupInterconnectMethod "load-balance")
```

The RTI Forwarders automatically determine the necessary connections for each individual forwarder and assemble themselves into the optimal forwarder network.

If it is not feasible to have every forwarder in a group act as a portal (for instance if some forwarders in a group are behind a firewall and no port forwarding is available) they should be marked as non-portals (`acceptRemoteConnections = 0`) in their `RTI-addForwarderToGroup` entry. The other RTI Forwarders will automatically distribute connections to the affected group among the portals in that group. The non-portal forwarders will still connect to other portals in foreign forwarder groups, and these connections are accounted for when determining the optimal interconnect routing to ensure that every portal has a roughly equal number of connections to foreign forwarder groups to handle.

11.6.6. Configuring Distributed UDP Forwarding

The RTI Forwarder can send both TCP and UDP messages across a WAN. (UDP messages are sent between forwarders over a TCP connection.). The federates on a LAN send messages to the multicast address on their LAN. When you enable distributed forwarding for UDP messages, a single RTI Forwarder running on the LAN monitors the multicast messages and sends them to all the remote forwarder groups using the TCP network. The portal forwarder in that remote group then multicasts the messages to federates on its local LAN.

To configure distributed UDP forwarding:

1. Set the RID parameter `RTI_distributedUdpForwarderMode` to 2.
2. Configure distributed forwarding as you would for TCP.
3. Specify a single forwarder on each LAN that will be allowed to monitor multicast traffic by adding `RTI_allowMulticastForwarding` entries in the routes configuration files.



Only one RTI Forwarder on each LAN can monitor traffic. Otherwise there will be duplication of messages. (For details, please see “[Optional RTI Forwarder Configuration Parameters](#),” on page 11-20).

Sending Packets Directly to Remote RTI Forwarders Using UDP Unicast

An alternative way of supporting distributed UDP forwarding is to bypass the local RTI Forwarder and have each federate send a message directly to all of the remote RTI Forwarders via UDP unicast. To use this strategy, set the `RTI_distributedUdpForwarderMode` parameter to 1.



You must use distributed forwarding to communicate with multiple federates on a single remote host that cannot be reached via multicast. Adding an additional destination address to the host is not sufficient, because unicast UDP packets are only available to one application on a single IP:port pair. This UDP unicast limitation also means that the RTI Forwarder must run on a dedicated host so that it is assured of receiving any remote messages.

Assume LANs A, B, and C ([Figure 11-7](#)). Lan A has one federate. LANs B and C have multiple federates and each have a RTI Forwarder. Each federate’s RID file must list the multicast address for the local LAN (`RTI_destAddrString`) and the address of each remote forwarder (`RTI_addDestAddrString`) (or machine for LANs that do not run a forwarder).

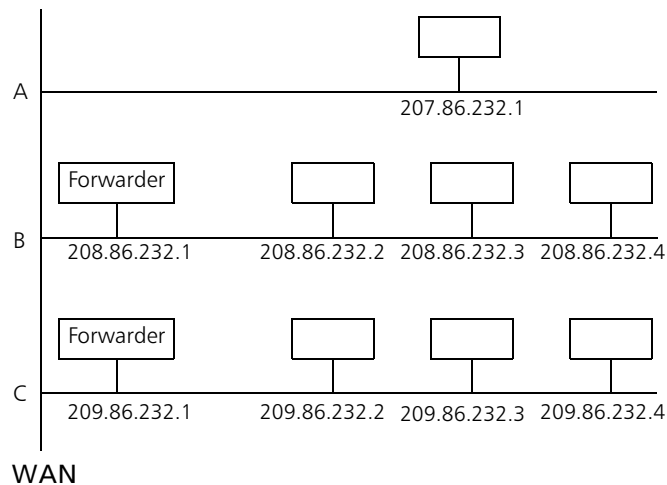


Figure 11-7. UDP forwarding

For the federate on LAN A, there are no other local federates; its only destinations are the federates on LANs B and C. Even so, there are some messages that can be sent by an LRC that are processed by the sending LRC (for example, `requestAttributeValueUpdate`). Therefore, the primary destination address must still be either a broadcast or multicast address (which will loop back the message). Then the addresses of the two RTI Forwarders at LAN B and C would be added as additional destinations.

The RID file for LAN A might have the following entries:

```
(setqb RTI_destAddrString "229.7.7.7")
(RTI-addDestAddrString "208.86.232.1")
(RTI-addDestAddrString "209.86.232.1")
```

The RID files for federates on LAN B should be configured with a multicast address as the primary destination for the other federates on the LAN. Then the address of the federate on LAN A and the UDP Forwarder on LAN C should be added as additional destinations.

```
(setqb RTI_destAddrString "229.7.7.7")
(RTI-addDestAddrString "207.86.232.1")
(RTI-addDestAddrString "209.86.232.1")
```

The RID files for federates on LAN C should be configured similarly, but use LAN B's UDP Forwarder address.

```
(setqb RTI_destAddrString "229.7.7.7")
(RTI-addDestAddrString "207.86.232.1")
(RTI-addDestAddrString "208.86.232.1")
```

11.7. Configuring Routers or Firewalls for WAN Federations

As explained in “[Configuring Routers or Firewalls to Support WAN-based Federations](#),” on page 11-5, when you use distributed forwarding, you will probably need to configure your system to allow traffic to get through firewalls to your RTI Forwarders.

If you are using distributed forwarding, you will typically be running one copy of the RTI Forwarder application on each LAN. But if you tried to configure your *routes.mtl* file with the IP addresses of the various machines that are running the RTI Forwarders (as described in “[Configuring RTI Forwarders for Distributed Forwarding](#),” on page 11-16), those addresses would be unreachable, since those machines are behind firewalls. To configure distributed forwarders:

1. Configure your *routes.mtl* file so that it uses the IP addresses of the various LANs' firewalls.
2. Configure your firewalls so that they will forward TCP packets received on the RTI's distributed forwarder port to the machines that host the RTI Forwarders. The port number used for messages destined for distributed RTI Forwarders is indicated by the `RTI_distributedForwarderPort` parameter in *rid.mtl*.

11.8. Special Distributed Forwarder Networks

The distributed forwarder networks described in previous sections can all be described as fully-connected in that each singleton forwarder or forwarder group connects to all other singleton forwarders or groups. Where forwarder groups are used, each group connects to every other group. The appropriate connections between individual forwarders are determined automatically at run-time when single-portal or load-balanced interconnects are selected in the routes file.

However, certain situations may dictate more complex network layouts. Manual configuration of forwarder interconnections permits you to construct an intricate forwarder network.

Such situations are likely to involve forwarders that cannot communicate with all other forwarders to produce a fully-connected network, whether because of infrastructure restrictions or security concerns. In place of a fully-connected network, you can set up a hierarchical network.

11.8.1. Configuring Hierarchical Forwarder Networks

In order to create a hierarchical forwarder network, every connection between individual forwarders must be specified in the *routes.mtl* file and the routing of messages to and from adjacent nodes must be specified for each forwarder.



Large forwarder networks can be quite complex. If you plan to configure a hierarchical network, we recommend that you contact MÄK technical support for assistance.

To set up a hierarchical forwarder network, set the `RTI_forwarderGroupInterconnectMethod` parameter to `manual`.

Specify connections between adjacent forwarder nodes by including an `RTI_addForwarderConnection` entry in the routes file for each connection. The syntax of this parameter is:

```
(RTI_addForwarderConnection "ForwarderName1" "ForwarderName2")
```

The *ForwarderName* strings cannot contain spaces and must be enclosed by double-quotes (" "). The names must be identical to a name specified in one of the `RTI_addForwarder` parameters. The `RTI_addForwarder` parameters must be specified before the connections between them are specified.

The routing of messages through a forwarder node is specified by including an `RTI_addForwarderRoute` entry in the routes file for each adjacent node. The syntax of this parameter is:

```
(RTI_addForwarderRoute "LinkNode" "Source" "NextHop")
```

where:

- ♦ *LinkNode* is the RTI Forwarder through which the message passes
- ♦ *Source* is the RTI Forwarder from which the message originates
- ♦ *NextHop* is the destination RTI Forwarder for this network hop.

The *LinkNode*, *Source*, and *NextHop* strings must be valid forwarder names. There can be multiple entries between source and link nodes with different next hop nodes, and likewise there can be multiple entries dictating different sources through a link node to any particular next hop node.

Each of these entries configures a routing rule at the link node such that any message received by the link node from the source node will be forwarded to all the next hop nodes for which a matching routing rule exists at the link node. It is not necessary to establish a route beyond the next hop from any link node. However forwarder routes are uni-directional, so to establish a complementary return path for messages, a separate `RTI_addForwarderRoute` entry with the source and next hop values swapped is required.

Consider the simple hierarchical network illustrated in [Figure 11-8](#). Forwarders A and B serve as link nodes between all other forwarders. Consequently routes must be configured to direct messages from Forwarder C through Forwarder A to Forwarders B and D, and both reverse routes must be configured (from B through A to C and from D through A to C). Likewise routes from Forwarder D through Forwarder A to Forwarder C and B (and the reverse) must be created, as well as routes from Forwarders E and F through Forwarder B and their reverse routes. No explicit entry is necessary to spell out the path from Forwarder C to Forwarder E since the entries covering the path from C through A to B and from A through B to E will allow the messages to be delivered end-to-end. Since in this example Forwarders C, D, E, and F are leaf nodes, none of the routes go through them. In other words, leaf nodes cannot be link nodes.

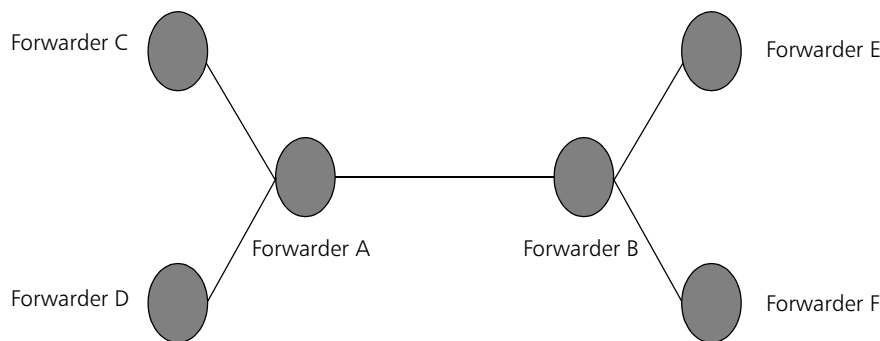


Figure 11-8. Hierarchical Forwarder Network

The necessary entries in the routes file to create this simple network are:

```

(RTI_addForwarderConnection "A" "B")
(RTI_addForwarderConnection "A" "C")
(RTI_addForwarderConnection "A" "D")
(RTI_addForwarderConnection "B" "E")
(RTI_addForwarderConnection "B" "F")

(RTI_addForwarderRoute "A" "C" "B")
(RTI_addForwarderRoute "A" "B" "C")

(RTI_addForwarderRoute "A" "C" "D")
(RTI_addForwarderRoute "A" "D" "C")

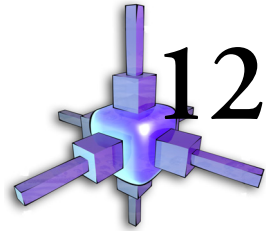
(RTI_addForwarderRoute "A" "D" "B")
(RTI_addForwarderRoute "A" "B" "D")

(RTI_addForwarderRoute "B" "E" "A")
(RTI_addForwarderRoute "B" "A" "E")

(RTI_addForwarderRoute "B" "E" "F")
(RTI_addForwarderRoute "B" "F" "E")

(RTI_addForwarderRoute "B" "F" "A")
(RTI_addForwarderRoute "B" "A" "F")

```



Configuring HLA Services

For a listing of all configuration parameters, with brief descriptions, please see Appendix A, *RID File Parameters Reference*.

Enabling and Disabling HLA Services.....	12-2
Configuring MOM Services	12-2
Configuring Time Management Services	12-2
Configuring Internal Messages.....	12-3
Configuring the Delay Time for Processing Internal Messages.....	12-3
Transmitting Internal Messages Across Ticks	12-4

12.1. Enabling and Disabling HLA Services

MOM, time management, and DDM services may incur additional overhead in the RTI. Since many federations do not use these services, or only some of them, the default MÄK RTI configuration disables them to optimize RTI performance. If your federation requires full HLA compliance, you can enable all of these services by setting `RTI_forceFullCompliance` to 1. If your federation uses a subset of the HLA services, you can enable or disable them independently.

12.1.1. Configuring MOM Services

- To enable MOM services, set `RTI_momServiceAvailable` to 1.
- To generate and log MOM exceptions, set `RTI_momExceptionReports` and `RTI_momExceptionLogging` to 1.
- To set the interval at which the MOM Federate object is updated, set `RTI_momFederateUpdateInterval`. The default (0) means that the MOM federate object is not updated.

Enabling MOM Service Reports

You can instruct the RTI to enable MOM service reports even if the federate does not enable them using the RTI Ambassador service call. This feature may be useful for managing federations that have federates that do not automatically enable service reports. To enable MOM service reporting, set the `RTI_momServiceReporting` parameter to 1.

12.1.2. Configuring Time Management Services

Time management requires the use of the `rtiexec`.

- To enable time management services, set `RTI_timeMgmt` to 1.

12.2. Configuring Internal Messages

The configuration options for internal messages help you manage internal network traffic. The implementation of the HLA services requires messages to be exchanged between the RTI components. In order to adhere to the HLA Interface Specification, these message exchanges can sometimes be redundant and also cause message bursts. A slight relaxation of the specification allows the RTI to regulate these internal messages in order to improve performance. You can use the configuration options in this section to regulate the internal message traffic.

12.2.1. Configuring the Delay Time for Processing Internal Messages

The RTI can delay the processing of requests by setting `RTI_enableMessageThrottling` to 1. To configure the delay, set `RTI_transmitDelay` to the amount of time (in seconds) that the RTI should delay before processing requests.

During federation startup it is typical for federates to perform declaration and object management services that require the RTI to exchange a significant number of internal messages. Often, these messages are redundant, simply satisfying duplicate requests from each joining federate. For example, the LRC of a federate that has registered an object will resend the register information for each subscription message that it receives (that matches the registered object). The LRC can condense these duplicate responses (for example, register messages) by delaying the processing of requests (for example, subscription messages) allowing a single response to satisfy multiple requests. Another example is the processing of request attribute values. By delaying the invocation of the federate ambassador provide attribute value service, a single invocation can handle the multiple requests typically occurring during federation startup.



In the 1516 APIs, the user supplied tag is dropped.

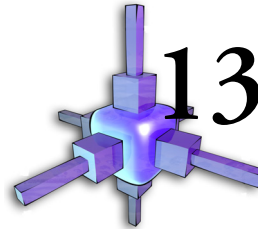
12.2.2. Transmitting Internal Messages Across Ticks

The RTI can send out internal messages at a throttled rate. This is enabled by setting `RTI_enableMessageThrottling` to 1. To configure the transmission rate, set `RTI_transmitRate` to the rate, in messages per second, that the LRC should transmit all required internal messages across multiple ticks (based on the time since the last tick()). The default setting is 0.0, which means the LRC should send all messages during a single tick.

During federation startup it is typical for federates to perform declaration and object management services that require the RTI to exchange a significant number of internal messages. Often, the LRC has to respond to messages that it receives by retransmitting messages about its internal state (for example, object registrations and class publications). The LRC retransmits all the necessary messages during a single tick. This message retransmission process can cause a spike in the message bandwidth. To smooth out these message spikes, the LRC can be throttled to retransmit messages at a specified rate across subsequent ticks.



The throttle only affects the retransmission of internal messages during tick(). It does not affect messages required by services invoked by the federate (for example, updates and interactions).



Using Data Distribution Management

This chapter describes the approaches to Data Distribution Management (DDM) that are implemented by the MÄK RTI.

Configuring DDM Services	13-2
Enabling DDM	13-2
Configuring Distributed Region DDM	13-2
Configuring a Fixed Grid DDM	13-3
Configuring Implicit DDM	13-5
The Distributed Region Approach	13-7
Attribute Passelization	13-9
The Fixed Grid Approach	13-9
The Fixed Grid Representation	13-11
Federate Region Intersections with the Fixed Grid	13-12
Normalization of Federate Data for Range Bounds	13-14
The Fixed Grid Approach and DDM Services	13-15
Implicit DDM	13-17
Implicit DDM Implementation	13-18
Adding Implicit DDM to the FED or FDD	13-18
Implicitly Associating DDM Regions	13-20
Modifying Region Extents	13-20
Modifying Fixed Grid Region Extents	13-22

13.1. Configuring DDM Services

The MÄK RTI implements the following approaches to DDM:

- ♦ Distributed region
- ♦ Fixed grid
- ♦ Implicit DDM.

This section explains how to enable DDM and configure the different DDM approaches. The rest of this chapter provides conceptual background for the different approaches to DDM.



The HLA Evolved implementation of DDM is the same as the HLA 1516 implementation. Therefore, in this chapter, all references to HLA 1516 apply to HLA Evolved as well.

13.1.1. Enabling DDM

- To enable DDM services, set `RTI_dataDistMgmt` to 1.

13.1.2. Configuring Distributed Region DDM

Distributed region DDM is explained in “[The Distributed Region Approach](#),” on page 13-7. To use the distributed region approach, specify values for the `RTI_minChannelAddr` and `RTI_maxChannelAddr` parameters. They define a pool of multicast addresses that the LRC will assign to regions used for sending interactions and updating attributes. For optimal DDM filtering, each federate should define a separate pool of multicast address for its LRC to use.

The parameter `RTI_addressDelay` is used to temporarily bypass the use of an updated region’s multicast address when sending an interaction or attribute update. During the delay, the region change can propagate to subscribing federates and allow any that form new region overlaps to receive the messages.

13.1.3. Configuring a Fixed Grid DDM

Fixed grid DDM is explained in “[The Fixed Grid Approach](#),” on page 13-9. To use the fixed grid approach, you enable settings in *rid.mtl* and create a file that specifies the grid. The fixed grid configuration specifies the partitioning of the axes and the association of the axes with FOM dimensions. It specifies a routing space that must contain all the dimensions used to configure the fixed grid. In HLA 1516, a notional global routing space is assumed, but it may be specified as *HlaGlobalSpace*. A dimension may be specified as the subspace axis. If specified, its partitioning determines the number of application spaces. The remaining dimensions used to define the fixed grid are listed and only these dimensions may appear in the specification of the application spaces. The dimensions must be a subset of the routing space dimensions.

The RID parameters for fixed grid DDM are as follows:

- ♦ **RTI_ddmFixedGrid.** Enables or disables the fixed grid approach
- ♦ **RTI_ddmFixedGridFilename.** Specifies the file name (and path) of the fixed grid specification file
- ♦ **RTI_minChannelAddr.** Specifies the base multicast address used to initialize the fixed grid.

The syntax of the fixed grid specification file is as follows:

```
;;with subspace;;
([RoutingSpaceName]
  (SubspacePartition subspaceDimensionName partitionSize
    (dimName*) [defaultPartitionSize]
    (
      [ (index (descriptiveName (dimName partitionSize)* )]*
    )
  )
)
;;without subspace;;
([RoutingSpaceName]
  (
    (<dimName> [defaultPartitionSize])*
  )
)
```

where:

- ♦ *RoutingSpaceName* specifies the name of the routing space. For HLA 1.3, the routing space name is required to get dimension handles. For HLA 1516, the routine space name is optional. The default is an internal routing space named *HlaGlobalSpace*.
- ♦ *SubspacePartition* is a keyword. Its components are:
 - *subspaceDimensionName* specifies the FOM dimension
 - *partitionSize* specifies the number of partitions for the subspace axis
 - *dimName* specifies one or more dimension names of the remaining axes
 - *defaultPartitionSize* specifies the default partition size.

- ♦ Each index line specifies partitions for the application space, as follows:
 - *index* specifies an application space, beginning with 0. If an application space index is omitted, its axes are partitioned using the default partition size.
 - *descriptiveName* is for documentation only.
 - *dimName* specifies how the dimensions will be partitioned for each application space. If the dimension name is omitted, its axes is partitioned using the default value partition size.
- ♦ `;;` indicates a comment.

The following example fixed grid specification shows six application spaces in the subspace partition:

```
(RS1
  (SubspacePartition subspace 6 (one two) 1
    (
      (0 (level0 (one 1) (two 1)))
      (1 (level1 (one 2) (two 2)))
      (2 (level2 (one 4) (two 4)))
      (3 (level3 (one 8) (two 8)))
      (4 (level4 (one 16) (two 16)))
      (5 (level5 (one 32) (two 32)))
    )
  )
)
```

Fixed Grid Search Order

The fixed grid configuration file search order is as follows:

1. Try to load the grid file specified by `RTI_ddmFixedGridFilename` in the RID file.
2. Try to load `RTI_CONFIG/grid_file_name_from_RID_file`.
3. If the configuration file is not found, the RTI throws an exception.

13.1.4. Configuring Implicit DDM

Implicit DDM is explained in “[Implicit DDM](#),” on page 13-17. You can configure the Implicit DDM plug-in. Configuration options include the:

- ♦ Upper and lower bounds of the dimensions
- ♦ Update region size
- ♦ Update region margin
- ♦ Sensor range, which determines the subscription region size
- ♦ Weapon range, which determines the interaction region size.

The configuration file uses MÄK Technologies Lisp (MTL) format (as described in Appendix A, [RID File Parameters Reference](#)). The name of this file is specified by the RID file parameter `RTI_implicitDdmParamsFile`. The syntax for this parameter is

```
(setqb RTI_implicitDdmParamsFile "fileName")
```

where *fileName* is a quoted string containing the name of the implicit DDM configuration file.

[Table 13-1](#) describes the parameters in the implicit DDM configuration file:

Table 13-1: Implicit DDM parameters

Parameter	Description
<code>RTI_implicitDdmDefaultUpdateRange</code>	Specifies the maximum range, in meters, that the object can travel between successive location updates. This value is not used when fixed grid DDM is enabled. The default update range is used when a range is not specified for the object class using <code>RTI-setObjectUpdateRange</code>. This value is not used when fixed grid DDM is enabled. Default: 0.0.
<code>RTI_implicitDdmDefaultSensorRange</code>	Specifies the maximum range, in meters, within which the object can sense another object. This value is added to the update range to determine the furthest point an object can sense another object between successive location updates. The default sensor range is used when a range is not specified for the object class using <code>RTI-setObjectSensorRange</code>. Default: 100.0.
<code>RTI_implicitDdmRegionMargin</code>	Specifies a factor that increases the update region range so that the region does not have to be modified each time the object updates its location. The update range is multiplied by this factor so that the region is not modified unless the resulting region extent does not encompass the object’s actual update range. This value is not used when fixed grid DDM is enabled. Default: 5.0.

Table 13-1: Implicit DDM parameters

Parameter	Description
RTI_implicitDdmMinExtentLimit	Specifies the range over which the location values are normalized. Defaults: -12000000.0 and 12000000.0.
RTI_implicitDdmMaxExtentLimit	
RTI_implicitDdmOriginLat	Specifies the relative center location used to normalize location values. Defaults: 0.0 and 0.0.
RTI_implicitDdmOriginLong	
RTI-setObjectUpdateRange	Specifies an update range for an object class. Multiple instances of this parameter are permitted.
RTI-setObjectSensorRange	Specifies a sensor range for an object class. Multiple instances of this parameter are permitted.
RTI-setObjectWeaponRange	Specifies a weapon range for an object class. Multiple instances of this parameter are permitted.

The following is an example of an implicit DDM configuration:

```
;; All distances in meters
(setqb RTI_implicitDdmDefaultUpdateRange 0.0)
(setqb RTI_implicitDdmDefaultSensorRange 2000.0)
(setqb RTI_implicitDdmMinExtentLimit -12000000.0)
(setqb RTI_implicitDdmMaxExtentLimit 12000000.0)
(setqb RTI_implicitDdmRegionMargin 6.0)

;; Latitude and Longitude in degrees
;;(setqb RTI_implicitDdmOriginLat 10.0)
;;(setqb RTI_implicitDdmOriginLong 60.0)

(RTI-setObjectUpdateRange "BaseEntity.PhysicalEntity.Platform.Aircraft"
 328.8)
(RTI-setObjectSensorRange "BaseEntity.PhysicalEntity.Platform.Aircraft"
 160000.0)
(RTI-setObjectWeaponRange "BaseEntity.PhysicalEntity.Platform.Aircraft"
 56000.0)
```


13.2. The Distributed Region Approach

In the MÄK RTI distributed region approach, each federate's LRC exchanges its region information with remote LRCs. The RTI must exchange information about update regions and subscription regions. Update regions are regions used for sending updates and interactions. Subscription regions are regions used in DDM subscriptions. The individual LRCs perform matching between the update and subscription regions. An LRC uses the region matches to establish communication channels between publishers and subscribers. Publishers then send data using the established communication channels. Only subscribers that are listening to those channels receive the data.

The MÄK RTI establishes best effort communication channels by mapping the update regions to multicast groups. These mappings are also distributed among the LRCs. When an LRC's subscription region matches an update region (that is, it overlaps and has common attributes), the LRC joins the update region's multicast group. The data is sent once to the multicast address and only subscribers that have joined that multicast group receive the data. In [Figure 13-1](#), each update region has its own multicast address and subscription regions join the multicast group of overlapping update regions.

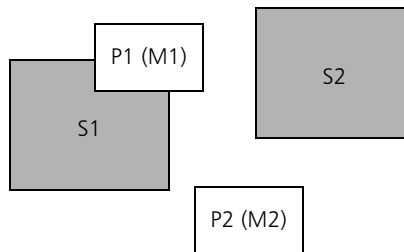


Figure 13-1. Distributed region using update regions

It is important to understand that multicast filtering is really just a very efficient receive side filtering. The actual bandwidth is not reduced. However, the processing at the receiver is greatly reduced as filtering occurs at the network layer rather than in the RTI layer. One caution with this approach is that the number of effective multicast groups is limited. Multicast groups will probably need to be shared among the communication channels. As a result, the RTI may receive messages that do not match the federate's subscriptions. The LRC handles these cases through an efficient channel identifier filter. Only messages with channel identifiers from matching update regions are processed.

The MÄK RTI also uses region matching information for reliable transmissions. Because both update and subscription region information is exchanged between the LRCs, an LRC can determine exactly which remote region subscriptions match its local update regions. The MÄK RTI uses this information to establish the destination mask used in sender-side filtering. The use of DDM matching to support smart forwarding results in true sender-side filtering. In fact, if no matching subscriptions are found on the sending side, the MÄK RTI can prevent updates from being sent for both reliable and best effort messages.

In Figure 13-2, P1 and P2 both have a subscription set of S1 and S2 while P3 has a subscription set of S1.

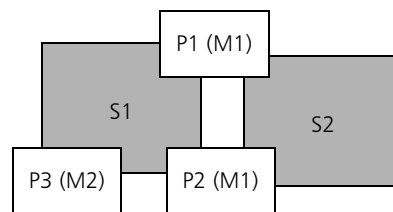


Figure 13-2. Distributed region using subscription regions

The advantage of distributed region matching is that the LRC can establish communication channels that carry relevant data only to receiving federates (in other words, perfect filtering). Additional unsubscribed attributes may be carried through the channel, but this multiple association problem exists in practically any approach, including the fixed grid approach. Also, the transmission of attributes and interactions is always sent only to a single channel (versus fixed grid where updates may have to be sent to multiple channels).

The disadvantage of distributed region matching is the considerable overhead of distributing the region information and performing matching between regions. With the distributed region approach, the overhead required to maintain regions becomes similar to object updates (with reliable attributes). A federation that naively uses regions and region updates can easily overwhelm any advantages of DDM implemented this way. Also, region matching scales poorly with the number of regions. Therefore, federates should limit the number of regions and the frequency that the regions are updated. An analysis of how using regions affects federation performance is provided in the paper SIW 99 1-054 “Characterizing Scenarios for DDM Performance and Benchmarking RTIs”.

A detailed description of the distributed region implementation for the MÄK RTI is provided in 02S-SIW-056 “Implementation of DDM in the MÄK RTI” which is available at http://www.mak.com/pdfs/wp_ddm.pdf.

13.2.1. Attribute Passelization

When a federate invokes an `updateAttributeValues` service, the LRC separates the submitted attributes into the designated region set channels formed from previous region associations. Therefore, the LRC may transmit more than one message for a given update even though all the attributes have the same transport type and order type. The LRC shares region sets across all object classes, but creates them separately for interaction classes. A federate should use separate regions when associating regions across multiple objects of different classes. Otherwise, the LRC will transmit the attributes from these objects using the same channel and filtering will be diminished.

13.3. The Fixed Grid Approach

A fixed grid approach to DDM allows the RTI to approximate the routing that would occur through update and subscription region overlap calculations, without having to exchange region information between LRCs. The fixed grid approach relies on an a priori configuration of multicast addresses into a fixed grid. Each axis of the fixed grid is associated with a FOM dimension. Each cell of the fixed grid is assigned a multicast address. The fixed grid takes the place of the opposing region in the DDM region overlap calculations. The federate regions are overlapped with the fixed grid and the overlapped grid cells establish the multicast addresses used for message routing. For subscription regions, the LRC joins each multicast address that is found in overlapped grid cells. For update regions, messages are sent using the multicast addresses found in overlapped grid cells.



The fixed grid approach is not fully compliant with either HLA 1.3 or HLA 1516. It relaxes some requirements and imposes some restrictions. (For details, please see “[The Fixed Grid Approach and DDM Services](#),” on page 13-15.)

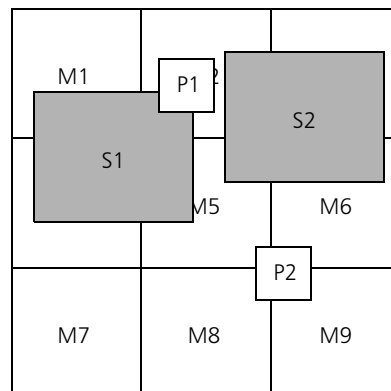


Figure 13-3. Fixed grid approach with overlaid regions

The MÄK RTI fixed grid approach is modeled after application space partitioning (introduced by Coffin, et. al. and Helfinstine, et. al.^{1, 2}). A few generic dimensions are defined, with one “subspace” dimension used to partition the others into application spaces. The remaining dimensions can then be interpreted and partitioned to meet the specific application requirements.

The same fixed grid approach is used for both the HLA 1.3 and 1516 APIs. The application space partitioning is less useful for DDM in HLA 1516, because each attribute is associated with individual dimensions rather than routing spaces. In HLA 1.3, a region contains all the dimensions in a routing space, while in HLA 1516 a region can be comprised of exactly the dimensions needed (that is, a subset of the available dimensions). Using the same implementation makes it possible for federates that use the different APIs to interoperate.

-
1. Experimentation with DDM schemes, D. Coffin, M. Calef, D. Macanucco, and W. Civinskas, Spring 1999 Simulation Interoperability Workshop, 99S-SIW-053
 2. Experiences with Data Distribution Management in Large-Scale Federations, B. Helfinstine, D. Wilbert, M. Torpey, and W. Civinskas, Fall 2001 Simulation Interoperability Workshop, 01F-SIW-032

13.3.1. The Fixed Grid Representation

The fixed grid is defined by a set of n axes that form grid cells when they are partitioned. One specific axis can be defined as a subspace axis. The partitioning of a subspace axis creates application spaces. For each application space, the partitioning of the remaining $n-1$ axes is separately defined. If no subspace axis is specified, there is one specification for the partitioning of the n axes.

Each cell of the fixed grid is assigned a multicast address. The assignment of multicast addresses starts with a base address. The addresses are assigned sequentially through a “depth first” recursion. With no subspace axis, this amounts to assigning the addresses uniformly across the dimensions. The assignment is similar with a subspace axis, but each subspace results in a distribution of addresses for the remaining dimensions based on the subspace partitioning.

Figure 13-4 illustrates a fixed grid partitioning and multicast address assignment. The figure shows a subspace axis partitioned into six application spaces. Each application space has a single axis. In the 0th application space partition the axis has one partition, in the 1st application space it has two partitions, in the 2nd application space it has four partitions, and so on.

GRID CELLS	224.0.0.1	224.0.0.2	224.0.0.4	224.0.0.8	224.0.0.16	224.0.0.32				
			224.0.0.5	.	.	.				
		224.0.0.3	224.0.0.6	.	.	.	.			
			224.0.0.7							
		224.0.0.15								
	224.0.0.31									
					224.0.0.63					
Subspace Partition	0	1	2	3	4	5				

Figure 13-4. Fixed grid partitions and multicast assignment

For details about configuring the fixed grid approach, please see “[Configuring a Fixed Grid DDM](#),” on page 13-3.

13.3.2. Federate Region Intersections with the Fixed Grid

A federate region intersection with the fixed grid maps each dimension range onto the associated fixed grid axes. The result of this mapping is a beginning and an ending index along each of the fixed grid axes. These indices specify the overlapped grid cells that contain the multicast addresses. The set of multicast addresses are then used for sending updates and interactions or subscribing to object class attributes or interaction classes.

Each fixed grid axis has an upper and lower bound. The lower bound is always 0. For HLA 1.3, the upper bound is the maximum value of an unsigned long integer (that is, 4,294,967,295 or ((unsigned long)-0)). For HLA 1516, the associated FOM dimension defines the upper bound. The width of a grid cell along an axis is determined by dividing the axis upper bound by the number of partitions. For an axis with n partitions the width is calculated as:

$$W = \text{upperBound}/n$$

The mapping of a region's upper or lower range bound value, V , to an index along a fixed grid axis is calculated as:

$$I = \text{floor}(V / W)$$

For example, assume the following configuration with an upper bound of 400 for all dimensions:

```
(RS1
  (SubspacePartition subspace 6 (one two) 1
    (
      (0 (level0 (one 1) (two 1)))
      (1 (level1 (one 2) (two 2)))
      (2 (level2 (one 4) (two 4)))
      (3 (level3 (one 8) (two 8)))
      (4 (level4 (one 16) (two 16)))
      (5 (level5 (one 32) (two 32)))
    )
  )
)
```

The fixed grid with the same partitioning as [Figure 13-4](#) would have grid cell boundaries as illustrated in [Figure 13-5](#). A federate region has a lower and an upper bound for each dimension in the region. Using the index formula, the lower bound is mapped to a beginning index and the upper bound is mapped to an ending index. The dimension specified as the subspace dimension is mapped first. The range for the subspace is assumed to be a point range (only the lower bound used), because a single subspace must be selected.

Given that the subspace dimension has 6 partitions, a range bound value from 0 through 66 would result an index of 0, from 67 through 133 would result in an index of 1, and so on. For the last partition, the maximum upper bound would be mapped down into the maximum index, so a value of 400 would result in an index of 5 versus 6. For the other dimensions, the index would depend on which subspace index was specified. For subspace 0, any value would result in an index of 0. For a subspace of 1, values 0 through 199 would result in an index of 0, and values of 200 through 400 would result in an index of 1. The indices for the other subspace partitions are calculated similarly.

G R I D C E L L S	399		300		375	388	
				350	374		
				349			
			299	.	.	.	
				.	.	.	
				.	.	.	
				.	.	.	
		200	199	100			
				99	50		
					49	25	13
				24			
			0				

Subspace Partition	[0	-	66][67	-	133][134	-	199][200	-	266][267	-	333][334	-	399]

Figure 13-5. Grid Cell Partition Boundaries

13.3.3. Normalization of Federate Data for Range Bounds

Similar to the mapping of the region's range bounds onto the fixed grid axes, federate data must be mapped or normalized into the region range bounds. Mappings that work well with the fixed grid approach are “enumerated” (for example, vehicle type) and “linear” (for example, location).

For an enumerated mapping, the magnitude of a dimension's full extent is partitioned by the number of enumerations. The range bound is then set as a point value in the midpoint of the partition. An obvious configuration for a dimension with an enumerated mapping would be to match the fixed grid partitions on that axis with the number of enumerations. Then each value of the enumeration would specify a separate multi-cast address.

The formula¹ for an enumerated mapping is calculated as follows:

1. Determine a scale between the dimension extent and the enumeration:

```
double scale = dimensionUpperBound / double(maxEnum - minEnum + 1)
```

2. Move the rangeBound into the center of the partition by adding $scale/2$:

```
double rangeBound = (scale * (value - minEnum)) + scale / 2
```

3. Truncate the final value to the nearest integer:

```
unsigned long result = rangeBound
```

For a linear mapping, the magnitude of the dimension's full extent is scaled by the linear extent. A typical linear mapping for a coordinate system would first define the bounds of the coordinate values (for example, minimum and maximum longitude and latitude). The partition of the fixed grid axes could then represent the resolution of filtering (that is, more partitions would allow finer-grained location filtering).

The formula for a linear mapping is similar to that for an enumerated mapping, but without the adjustment to the center of the partition. It is calculated as follows:

1. Determine a scale between the dimension's extent and the linear bounds:

```
double scale = dimensionUpperBound / double(max - min)
```

```
double rangeBound = scale * (value - min)
```

2. Truncate the final value to the nearest integer:

```
unsigned long result = rangeBound
```

1. Based on formulas used by MC02.

13.3.4. The Fixed Grid Approach and DDM Services

The fixed grid approach is not fully compliant with any of the HLA specifications. It relaxes some requirements and imposes some restrictions. This section lists each of the DDM services and related object management services with a description of how the MÄK RTI fixed grid DDM implementation behaves and how the services may differ from strict HLA requirements.



In the following sections, all comments regarding HLA 1516 also apply to HLA Evolved.

Create Region

A fixed grid region is initialized with the exact dimensions that are used to configure the fixed grid. As a result, any additional dimensions are ignored (even in HLA 1.3 when they belong to the same routing space). Since the region is initialized with the fixed set of dimensions, any of these dimensions not specified when creating the region are automatically added. The initial values for the range bounds are the default range bounds. In HLA 1.3, the default range bounds are 0 to 4,294,967,295. In HLA 1516, the default range bounds are set by the FOM. If the FOM omits the default range bounds or the default range bounds are excluded, the default range bounds are 0 to 1.

Modify Region / Commit Region Modifications

A region can be committed without setting all of its range bounds. Any range bound that has not been set assumes the dimension's default range. In HLA 1516, you cannot exclude fixed grid dimensions. Excluded dimensions take on the default range bound values of 0 to 1.

Delete Region

This service behaves as expected. A region in use cannot be deleted.

Register Object Instance With Region

For a given object instance, all of its attribute instances must use one, and only one, region. The RTI uses the first region and ignores any others. Registering an object with region for any attribute results in all attributes being associated with that region.

Associate Object Instance With Region

All object attribute instances must use one, and only one, region. The RTI uses the first region and ignores any others. Associating an object with region for any attribute results in all attributes being associated with that region. Subsequent associations must be with that same region and thus are redundant, including a region used to register the object.

Unassociate Object Instance With Region

All attribute instances of the object become unassociated with region when any one or more attributes are unassociated.

Subscribe Object Class Attributes With Region

Multiple regions can be used for object class attribute subscriptions, including successive subscriptions. However, the object class attribute subscriptions are additive across all regions. In essence, an additive declaration management subscription occurs along with the subscription to the regions' multicast addresses. The result is that all subscribed attributes that match an update are reflected regardless of which region's multicast subscription transported them.

Unsubscribe Object Class Attributes With Region

All attributes of a class are unsubscribed from the region. When no regions are subscribed to the object class attributes, the object class is unsubscribed. Any declaration management subscriptions to the class are also unsubscribed. Further, a declaration management unsubscription results in those attributes no longer being included in reflects (even though the messages are received by the LRC).

Subscribe Interaction Class With Regions

An interaction subscription with region behaves as expected.

Unsubscribe Interaction Class With Regions

The region is unsubscribed from the interaction class. When no regions are subscribed with the interaction class, the interaction becomes unsubscribed. Any declaration management subscription is also unsubscribed. Further, a declaration management unsubscription results in the interaction no longer being received (even though the messages are received by the LRC).

Send Interaction With Regions

When sending interactions with regions, only point regions are supported. The RTI only uses the lower bound.

Request Attribute Value Update With Regions

A request with region is sent to each of the region's multicast addresses.

Update Attribute Values

When sending attribute updates that have been associated with regions, only point regions are supported. The RTI only uses the lower bound.

Miscellaneous

In HLA 1.3, the region's first extent is used and any additional extents are ignored. In HLA 1516, a region only has one extent.

The fixed grid approach disables all advisories. No subscription or publication information is exchanged between federate LRCs.

13.4. Implicit DDM

Implicit DDM provides the advantages of DDM filtering without a federate using the DDM services. Implicit DDM uses a MÄK RTI plug-in that substitutes non-DDM method calls from a federate with DDM method calls. The DDM calls create regions and modify region extents using a configurable set of criteria, based on geographic data as specified by the RPR FOM.

The Implicit DDM plug-in provides geographic range-based filtering. It allows you to specify three ranges: an update range, a sensor range, and a weapon range, for any RPR FOM BaseEntity object, but primarily for PhysicalEntity::Platform objects. The update range is the distance an entity can move in one update interval. The sensor range is the range in which an entity can sense other vehicles. The weapon range is the range in which an entity can engage another entity. It allows different BaseEntity subclasses to be configured with different ranges to reflect the different capabilities of the platform. For example, the sensor range of an aircraft can be set to a different value than that of submersible vessels.

The effect of the Implicit DDM plug-in is that as the federate executes, it will only receive updates and interactions from entities that are within range of the federate's entities. Based on implicitly created and modified DDM regions, entities are within range when the update range or weapon range of discovered entities overlaps the sensor range of the federate's registered entities.

The plug-in works with either the MÄK RTI's distributed region approach (described in [“The Distributed Region Approach,”](#) on page 13-7) or its fixed grid approach (described in [“The Fixed Grid Approach,”](#) on page 13-9.) At runtime the plug-in determines what type of approach is in use based on RID file parameters and adjusts its behavior so you can use the same plug-in DLL (or .so) regardless of the underlying DDM approach.

The following sections describe how implicit DDM is implemented and how to configure the RTI to use it.

13.4.1. Implicit DDM Implementation

The Implicit DDM plug-in uses the RTIspy API. (For details about the RTIspy API, please see Chapter 17, *The RTIspy Plug-in API*.) The plug-in changes the way objects are registered, published, subscribed, and updated, and introduces changes when the federation joins (for example, modifying the RTI's internal FED routing space).



Almost all of the the plug-in's work is performed above the actual RTI Ambassador implementation, with perhaps some alterations to other classes employed by the RTI Ambassador such as the FOM Manager (*DtFomClassManager*). It was not necessary to change the Federate Ambassador to implement the implicit DDM approach.

Since the plug-in must intercept and decode attribute value updates and interactions, it needs some knowledge of the underlying federation object model. The Implicit DDM plug-in was designed to be compatible with HLA RPR FOM 1.0 and HLA RPR FOM 2.0 Draft 17 or their derivatives. All references to actual classes, attributes, interactions, and parameters will use RPR FOM 1.0 nomenclature.

13.4.2. Adding Implicit DDM to the FED or FDD

During the join process, the Implicit DDM plug-in creates its own routing space with its own dimensions representing geocentric coordinates. In HLA 1.3, the plug-in adds a routing space named “ImplicitDdmSpace” to the RTI's internal FED. In the 1516 specifications, just the dimensions are added to the RTI's internal FDD. The added dimensions are called HLA_X, HLA_Y, and HLA_Z and correspond to the geocentric coordinates X, Y, and Z. The routing space and dimensions are associated with the attributes of certain classes of objects (*BaseEntity*) and to certain interactions associated with those objects (*WeaponFire* and *MunitionDetonation*).

The implicit DDM routing space is automatically added to the RTI's internal FED and it should not be declared in the actual FED file.

The 1516 implementations of the Implicit DDM plug-in add the implicit DDM dimensions to the MÄK RTI's 1516 global routing space.

Normalization and Initial Bounds

By default the implicit DDM dimensions are centered on the geocentric origin (0,0,0). This can be changed to any point on the Earth's surface by specifying the latitude and longitude of the origin in the implicit DDM configuration file.

The default implicit DDM routing space defines the limits of each dimension to be:

- ♦ min — -12000000.0 m
- ♦ max — 12000000.0 m.

The approximate radius of the Earth is 6000 km. Therefore, the default space subtends a cube roughly twice the diameter of the Earth on all sides. The minimum and maximum extent limits are configurable.

All the dimensions are normalized to take values between 1 and MAX_INT-1 (for 1516 the normalized boundaries are from 2 to MAX_INT-2). This range allows for point regions to be created at {0,0,0} and {MAX_INT,MAX_INT,MAX_INT} which an object can never occupy.

Because the DDM is implicit, no region information is associated with an object instance when it is registered. So until the first location attribute for the new object instance is decoded by the plug-in, it cannot set the region bounds based on the actual coordinates of the object instance. Instead, the plug-in creates initialization regions and sets the initial bounds to one of the two inaccessible points. This ensures that when a new object instance is created, its update regions will not inadvertently overlap with any remote federate's subscription region and trigger erroneous detection by remote federates. Similarly, the federate's subscription region will not inadvertently overlap with remote federate update regions.

13.4.3. Implicitly Associating DDM Regions

The Implicit DDM plug-in keeps track of each instance of certain object classes. It maintains a list of handles for all the object instances of type `BaseEntity` that are registered by the federate. For each such instance it creates two regions: an update region (the update range) similar to that used in normal DDM processing, and an interaction region (the weapon range). The plug-in also creates a single subscription region (the sensor range) used by all objects. For each registered object it adds a subscription extent.



HLA 1516 APIs do not support subscription regions with multiple extents. However it was necessary for the plug-in to be able to create and modify an arbitrary number of extents within each object's subscription region. Consequently, in certain circumstances, the range-based Implicit DDM plug-in bypasses the HLA 1516 API calls to directly access the underlying classes.

The plug-in then intercepts the attribute updates and interactions from each instance of the desired classes and extracts location information. Before it allows the attribute updates or interactions to be processed further, the plug-in uses the location information to modify the extents of the regions it has created and to make the necessary DDM calls to distribute the region modifications. If necessary, a DDM call is substituted for the normal DM call (for example, `sendInteractionWithRegion`). No additional changes are necessary, because the normal underlying DDM processing uses the implicit DDM region information as if the federate had made the DDM calls directly.

13.4.4. Modifying Region Extents

Region extents are modified slightly differently depending on the DDM approach that is being used by the RTI.

Modifying Distributed Region Extents

The three regions used for implicit DDM are updated based on the changes to the locations of the federate's objects. Each object has an update region and a weapon region. A single subscription region is maintained for the federate, but the region has an extent for each registered object. The update and weapon region as well as the subscription extent are checked whenever the corresponding object's location changes. The implicit DDM uses ranges and margins that can reduce the actual changes significantly, which is important with distributed regions. (For details, please see [“Configuring Implicit DDM,”](#) on page 13-5.)

When using the distributed region approach the update region reflects the furthest distance the object instance could move during a single update interval (the update range). The region has a single extent which surrounds the object. The range of the extent can be configured by registered class. It assumes that the object can change direction instantaneously, including 180° reversals. In other words the object's current orientation and velocity (momentum) are ignored in calculating the object's range. This simplifies the determination of region boundaries when processing location updates.

The object's weapon region reflects the range at which an object can interact with another (that is, sending a weapon fire or detonation interaction). Its range is an extension of the update range. The weapon range can also be configured by class.

The subscription extent reflects the furthest range at which an object can detect another object (the sensor range). The sensor range is also an extension of the object's update range. It can also be configured by class.

The Implicit DDM plug-in allows all range values to be configured on a class-by-class basis, so update region, interaction region, and subscription region sizes can be tailored to the type of object (vehicle, aircraft, surface vessel, and so on).

When any BaseEntity-class object performs an attribute value update, the plug-in decodes the WorldLocation attribute if it is included in the update and compares it against the three regions' extents, updating any of them if necessary. The region modifications are initiated before the attribute value updates are processed further. Aside from initiating the region modifications, the plug-in does no further special processing on the attribute value updates. The RTI's DDM processing checks for overlaps with other known objects' regions and establishes the communications channels.

To minimize the number of region modifications that must be performed, the plug-in can expand the boundaries of the regions by a factor that you can configure. This is particularly beneficial when using distributed regions in which each region modification results in one or more messages being sent over the network to be processed by all federates. This is less important when fixed grid DDM is used, because region modifications do not prompt additional network traffic or require processing at the remote federates.

When a federate sends an interaction using `sendInteraction()` the plug-in decodes the `FiringObjectIdentifier` parameter and looks in a map of object instance names to determine if the interaction was sent by a BaseEntity object which has an interaction region associated with it. If an interaction region can be associated with the interaction, the plug-in simply substitutes the `sendInteractionWithRegion()` call to the RTI for further processing.

i

If you use reliable internal messaging, it is strongly recommended that you enable the `RTI_dualTransmitFirstInteractionRegions` RID file parameter to compensate for a potential race condition between the region modification messages and the interaction message.

13.4.5. Modifying Fixed Grid Region Extents

When fixed grid DDM is enabled, the update and interaction regions become point regions rather than volumes. The update range and weapon range do not affect the region scope. Instead, when the federate updates an object's location attribute the update region's extent is set to the single point specified by the location. No change is made to the interaction region. This single point is enough for the underlying fixed grid implementation to establish a communication channel.

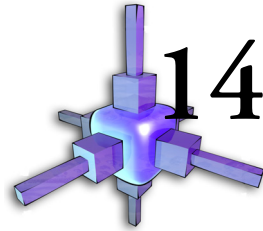
The interaction region is also handled differently in that the region is based on the location of the interaction, rather than the location of the object that initiated the interaction. The plug-in decodes the `FiringLocation` parameter (for `WeaponFire` interactions) or `DetonationLocation` parameter (for `MunitionDetonation` interactions), and the `FiringObjectIdentifier` parameter from the interaction message data. It looks in the map of the registered object instance to retrieve the interaction region for the object that sent the interaction. It then modifies that interaction region to reflect the interaction location. Again, this single point is sufficient for the fixed grid implementation to establish a communication channel.

The subscription region is also modified when the federate updates the location of one of its objects. The object's subscription region extent represents a volume determined by the corresponding object's location extended in all dimensions by the sensor range. The subscription region volume results in the local RTI listening to the communication channels where the sensor range overlaps the update or weapon range.

Fixed Grid DDM does not exchange region modification messages, so additional bandwidth savings can be realized.



The bandwidth savings will be offset somewhat by the fact that the smart forwarding feature is incompatible with fixed grid DDM, so all updates and interactions will transit the network even when there are no subscribers joined to the same multicast group.



Running the MÄK RTI Using Shared Memory

This chapter explains how to configure, run, and tune the use of shared memory.



HLA Evolved does not support use of shared memory.

Introduction	14-2
Configuring Use of Shared Memory	14-2
Enabling Shared Memory Mode.....	14-3
Specifying the Name of the Shared Memory Region.....	14-4
Specifying the Shared Memory Queue Length.....	14-4
Specifying the Shared Memory Manager Maximum Wait Interval.....	14-5
Message Queue Limits.....	14-5
Starting the Shared Memory Manager.....	14-5
Shared Memory Queue Manager Command-Line Options	14-6
Terminating the Shared Memory Manager	14-8
Abnormal Termination of the Shared Memory Manager	14-9
Tuning Shared Memory	14-9
Calculating the Queue Length.....	14-9
The RTI tick() Member Function	14-11
The Shared Memory Manager	14-12

14.1. Introduction

The MÄK RTI allows co-located federates (federates running on the same CPU or multi-processor machine) and an rtiexec process to communicate with each other using a message queue that is implemented using shared memory. This chapter describes how to configure and use this feature.



HLA Evolved does not support use of shared memory.

14.2. Configuring Use of Shared Memory

To configure the use of shared memory, edit the following configuration parameters in *rid.mtl*:

- ♦ **RTI_sharedMemoryMode**. Enables or disables use of shared memory
- ♦ **RTI_sharedMemoryName**. Specifies a name for the shared memory region
- ♦ **RTI_sharedMemoryQueueLength**. Specifies the length of the shared memory queue
- ♦ **RTI_sharedMemoryMgrMaxWait**. Specifies how long the shared memory manager waits for incoming messages before it polls the network.

Each co-located federate or rtiexec in a federation using shared memory must set the shared memory RID parameters to the same values.



For simplicity, it is assumed that all co-located federates belong to the same federation. It is possible to support multiple distinct federations either on a single shared memory message queue or on their own distinct shared memory message queues.

14.2.1. Enabling Shared Memory Mode

To enable or disable shared memory mode, edit the `RTI_sharedMemoryMode` parameter. The possible values are:

- ♦ 0 — disable shared memory mode. The RTI communicates using the default TCP or UDP sockets.
- ♦ 1 — enable shared memory communication only. All federates and the `rtiexec` (if used) must be co-located on the same host.
- ♦ 2 — enable shared memory communication with messages relayed to the network. All co-located federates communicate using shared memory. Messages are relayed between shared memory and the network to other remote federates or RTI components. This mode should only be used if there are RTI components executing on remote hosts. Otherwise, the shared memory manager will waste processing time sending messages on the network for no purpose.

When the shared memory feature is enabled, a Shared Memory Queue Manager process is required. It creates the shared memory region, sets up and manages the message queue within that shared memory region, and relays messages between shared memory and the network. You can start the Shared Memory Queue Manager manually or automatically. For details, please see [“Starting the Shared Memory Manager,”](#) on page 14-5.

14.2.2. Specifying the Name of the Shared Memory Region

You must specify a name for the shared memory region. To specify a name, edit the `RTI_sharedMemoryName` parameter. The name can have a maximum of 32 characters, with no spaces allowed.

The shared memory name uniquely identifies the shared memory region allocated by the operating system. The operating system creates a special file using the shared memory name specified as the filename. On Windows this file is in *.lib*. On UNIX this file is in */tmp*, with *.shared* appended to the filename. Make sure the user process that the federate application runs under has sufficient privileges to create this file in the appropriate location.

On Windows, the Shared Memory Manager process also creates a special lock file. This lock file only exists while a Shared Memory Manager process is running. It prevents a second manager process from running. The lock file is in the same location as the shared memory file. Its filename is the same as the shared memory file with *_mgr* appended to the end. The file's hidden attribute is set.

All federates that attach to the same shared memory region have read access to all messages in the message queue within that region. As a result, they must process each of those messages to some extent. Therefore, it is advisable to create a different shared memory region for each federation when there is more than one federation running on the same machine. However, if a single `rtiexec` is to be used by all the federations and that `rtiexec` is connected to shared memory on the same host, then you must use a single shared memory region.

14.2.3. Specifying the Shared Memory Queue Length

To specify the shared memory queue length, edit the `RTI_sharedMemoryQueueLength` parameter. The shared memory queue length specifies the number of message queue segments (called buckets) that are generated in the circular queue structure used by the MÄK RTI shared memory subsystem. Each bucket consists of a 20 byte header and a 200 byte payload. The shared memory subsystem automatically fragments and re-assembles messages larger than 200 bytes. There is also a queue header structure in shared memory that occupies a little over 1.3 KB.

On Windows, there is no fixed maximum size for the shared memory region, other than that which the operating system can allocate within memory at the time the shared memory region is created. On UNIX, the maximum size of the shared memory region is governed by the value of the kernel parameter `SHMMAX`. Typically this is 32 MB for 32-bit Linux kernels (versions 2.4 and 2.6), but may be tuned within the distribution you are using (check */proc/sys/kernel/shmmax* for the maximum segment size of your kernel). For more information, please see [“Calculating the Queue Length,”](#) on page 14-9.

14.2.4. Specifying the Shared Memory Manager Maximum Wait Interval

The shared memory manager maximum wait interval controls how long the Shared Memory Queue Manager process waits for messages to arrive in the message queue before it polls the network interface for incoming messages. To specify the wait interval, edit the `RTI_sharedMemoryMgrMaxWait` parameter. Since the network interface is only enabled in shared memory mode 2, this value has no effect in mode 1. For details, please see “[Calculating the Queue Length](#),” on page 14-9.

14.2.5. Message Queue Limits

The maximum number of concurrent attachments to a single shared memory message queue is 16,384. The Shared Memory Queue Manager counts as an attachment, as does the `rtiexec` process.

14.3. Starting the Shared Memory Manager

A separate shared memory manager process is needed to create and initialize the shared memory region before federates can attach to the message queue created in shared memory. By default, the `rtiexec` or a federate’s LRC automatically starts the shared memory manager process. The shared memory manager should exist for the lifetime of its shared memory region.



Each distinct shared memory region must have its own shared memory manager.

You can start the shared memory manager manually prior to starting the `rtiexec` or any federates. The shared memory manager application is in `./bin`. The executable is named `rtiShmQMgr` (for HLA 1.3) or `rtiShmQMgr1516`. The shared memory manager process must have access to a RID file with the same parameters used by the `rtiexec` or federates.



The shared memory manager does not support the connection configuration feature used by federates and the `rtiexec`. When you run the shared memory manager, it must be configured to listen to the port and address being used by the federation execution you want to join.

On Windows, the shared memory manager runs in a separate console window.

On UNIX, if the shared memory manager is started automatically, it runs as a foreground task attached to the terminal owned by the LRC that launched it. This possible sharing of the console terminal can cause difficulties for some federates. For example, the rtiexec running in console mode would lose its console input to the shared memory manager process. Therefore, on UNIX we recommend that you start the shared memory manager manually from a separate terminal session before you start your federate (or the rtiexec).



You must restart the shared memory manager process when you change RID parameters.

14.3.1. Shared Memory Queue Manager Command-Line Options

If you start the shared memory manager from the command line, the syntax is as follows:

```
{rtiShmQMgr | rtiShmQMgr1516} options
```

Table 14-1 describes the command-line options.

Table 14-1: Shared memory manager command-line options

Option	Description
(-A --destAddrString) address	Overrides the IP address (configured by RTI_destAddrString) to which best-effort federation execution traffic is sent.
(-D --Daemon)	Run ShmExec in the background. (Linux only)
(-h --help)	Displays a syntax help screen.
(-k --ruthLess)	Specifies ruthless takeover. This option forces the Shared Memory Queue Manager process being launched to take over the shared memory queue named in the RID file and reinitialize the queue regardless of whether or not another active Shared Memory Queue Manager process exists or other processes are attached to and using the shared memory region. This can cause unexpected behavior among other processes subscribed to the shared memory queue at the time. It will not terminate any other processes, although that may be a side-effect. It should only be used in situations where attempting a graceful takeover has failed.
(-K --AutoExit)	Run ShmExec and automatically exit after the last subscriber disconnects.
(-l --setLogFileName) logfile	Enables logging to the specified file.

Table 14-1: Shared memory manager command-line options

Option	Description
(-n --notifyLevel) <i>level</i>	Overrides the notification level configured in the RID file. <i>level</i> can be: 0 — Fatal 1 — Warn (default) 2 — Notify 3 — Verbose 4 — Debug.
(-P --setUdpPort) <i>port</i>	Overrides the value specified by <code>RTI_udpPort</code> . This port is used to exchange messages between federates, the <code>rtiexec</code> , and RTI Forwarders, if applicable.
(-q --setNotifyQuiet)	Sets the notification level to Fatal (0).
(-R --setRidFileName) <i>ridfile</i>	Specifies the name of the RID file to use.
(-t --graceful)	Specifies graceful takeover. This option forces the Shared Memory Queue Manager process being launched to try to take over the shared memory queue named in the RID parameters, even if the queue was created by another Shared Memory Queue Manager process. This option is useful if the Shared Memory Queue Manager process that created the queue terminates abnormally or becomes unresponsive. This can cause unexpected behavior among other processes subscribed to the shared memory queue at the time.
(-v --version)	Displays version information and exits.
(-- --ignore_rest)	Ignore the command-line options that follow this option.

14.4. Terminating the Shared Memory Manager

The Shared Memory Queue Manager process shuts down automatically when all other processes have successfully detached from shared memory. There is a 30 second delay between the last federate detaching and the shared memory manager exiting. If a new federate attaches to shared memory during this interval the shared memory manager continues to operate without interruption.

It is possible that an abnormally terminating federate may not successfully detach from shared memory. When the other federates have detached from shared memory, the Shared Memory Queue Manager may still act as if the crashed federate is connected. In this case, you may need to shut down the shared memory manager manually.

- To shut down the shared memory manager manually, in the console window enter:

`q`

If there are still federates attached to shared memory a warning prompt will be displayed and the shared memory manager will enter a pending state.

- To override the warning and continue with the shut down, enter `q` again.



Ignoring the warning about attached federates will result in unexpected behavior for any federates that are still attached and the loss of communication to federates on remote hosts.

14.4.1. Abnormal Termination of the Shared Memory Manager

If the shared memory manager process terminates abnormally it may affect the operation of all federates attached to shared memory. In this case, it is recommended that you stop all co-located federates and the `rtiexec` (if it also co-located). Depending upon how the shared memory manager process terminated, the shared memory region may not get properly de-allocated. In these situations any attempt to start a new shared memory manager for that named shared memory region may fail and report error messages stating that a shared memory manager is already running. This error may occur despite the fact that a shared memory manager does not show up in the process (or task) list. The cause is most likely that a shared memory manager lock file has been left behind. You can delete the file manually, or you can take over the shared memory region with a new shared memory manager.

You can take over a shared memory region with a new shared memory manager process by launching the new process with the `-t` command-line option, for example:

```
rtiShmMgr -t
```



A shared memory manager initializes the internal shared memory structures. Therefore, attempting to take over an existing shared memory region that has federates attached can result in unexpected behavior in the federates' execution.

14.5. Tuning Shared Memory

This section discusses ways to improve performance when you are using shared memory.

14.5.1. Calculating the Queue Length

The value of the `RTI_sharedMemoryQueueLength` parameter depends on the following factors:

- ♦ The amount of available memory
- ♦ The number of federates
- ♦ The average rate at which federates generate messages
- ♦ The rate at which the slowest federate processes messages.

The RTI 1.3 and RTI 1516 specifications process messages in slightly different ways. In RTI 1.3, a federate processes messages by calling the `tick()` or `tick(min,max)` member functions. In RTI 1516, a federate processes messages by calling `invokeCallback()` or `invokeMultipleCallbacks()`. In this section, both methods are referred to by the term `tick`.

Available Memory and Number of Federates

The more memory available in the machine, the more that can be dedicated to shared memory. The more federates simultaneously attached to the message queue in shared memory, the larger the queue needed to support them.

Message Generation Rate

The rate at which the federates generate and process messages affects performance. The message queue employed within shared memory operates as a reliable broadcast, meaning every message is delivered to every federate attached to shared memory. Consequently, messages are retained in the shared memory queue until they have been read by every federate attached to shared memory. A disparity between a fast producer and a slow consumer can fill up the shared memory queue. To compensate for this case, the shared memory subsystem buffers new messages in main memory. Buffering messages prevents fast producers from blocking when the shared memory queue is full. However, buffering degrades performance and can use up all of the main memory.

MÄK recommends that you use the following formulas to calculate queue length:

```
Capacity = (average_updates_per_second_per_federate) *  
            (number_of_federates) / (slowest_tick_rate)  
Queue Length = Capacity * ceil(average_msg_length /  
                                bucket_payload_size)
```

where:

- `average_updates_per_second_per_federate` is the rate at which most federates generate messages (assumed to be updates).
- The `slowest_tick_rate` is the rate at which the slowest federate ticks its RTI, assuming all messages in shared memory are processed in one tick (that is, `RTI_singleCallbackPerTick = 0`).

For example, consider a scenario in which there are 1000 co-located federates (ignoring the `rtiexec` and Shared Memory Queue Manager for now). Federates #1 – #999 each generate 10 updates per second, or one update every 100 msec. Federate #1000 only ticks its RTI every 50 msec. The average update rate is 10 updates/sec and the slowest tick rate is 20 ticks/sec. This yields a required message capacity of 500 messages. Put another way, on average federates #1 – #999 generate 500 messages between them in the 50 msec between federate #1000's last tick. Since the average update message is 240 bytes long and bucket payloads are fixed at 200 bytes (this is not configurable), this equates to a required queue length of 1,000 buckets. This will occupy about 220 KB of RAM.

If some federates generate a lot of messages, while other federates read messages infrequently (that is, they do not tick their RTI often), the queue can fill up, preventing new messages from being queued. The shared memory subsystem can deal with this to prevent blocking. However, if the disparity between the message generation rate and the slowest federate's tick rate is too great, a serious bottleneck can occur.

A more serious issue arises if a federate has become totally unresponsive (hung or crashed) while subscribed to the shared memory queue. Since these federates are no longer reading messages, the queue will eventually fill up. To avoid permanently blocking new messages, the shared memory subsystem detects unresponsive federates and unsubscribes them from the queue if they have been inactive for one second or more. Once unsubscribed, these unresponsive federates cause an exception to be generated the next time they try to access the message queue in shared memory (if they start processing again). There is no way to allow these federates to resubscribe. Therefore, when you are debugging federates that employ shared memory you must remember that when execution of the federate process is intentionally halted, for example, when a breakpoint is hit, the other federate processes still running may unsubscribe the halted federate.

While it is tempting to simply set the queue size to the maximum allowed by your platform, remember that any memory not allocated to a shared memory region is available for other uses which may improve the overall performance of your federate application.

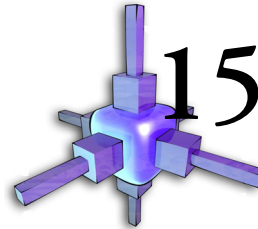
14.5.2. The RTI tick() Member Function

The way in which a federate ticks the RTI can affect performance. For instance, it is strongly recommended that `RTI_singleCallbackPerTick` be set to 0 when using shared memory, because each federate should try to read all unread messages in shared memory to keep up with the rate at which they are generated and avoid the bottleneck mentioned in the previous section.

Also, when multiple federates share CPU time, as they do when you use shared memory, you must be sure each federate process is given its share of CPU time. If the federate application does not yield the CPU occasionally, it is advisable to use the RTI's `tick(min,max)` member function to allow other federates to run once the RTI has processed all available messages.

14.5.3. The Shared Memory Manager

When shared memory is configured in mode 2, the shared memory manager process relays messages between the message queue and the network. However, due to limitations in the way processes can wait for messages in shared memory and the way the `select()` call works for network sockets, the shared memory manager process will only wait (suspend) while waiting for new messages to be enqueued on the message queue in shared memory. To minimize the amount of latency incurred by messages coming from the network, you can configure the shared memory manager to use a timed wait. If the timed wait expires because no new message was queued in shared memory, the manager process activates and checks for messages arriving from the network. If there are no messages arriving from the network, the manager process performs some basic house-keeping functions and again waits (suspend) for messages in shared memory. The federate designer should choose a reasonable wait interval based on the amount of latency that can be tolerated for messages arriving from the network. Setting the `RTI_sharedMemoryMgrMaxWait` parameter to 0 prevents the shared memory manager process from yielding the CPU if there are no messages to process. Instead it continuously polls the queue and the network interface.



Using FOM Modules

Use of FOM modules is functionality from the HLA Evolved specification. It has been implemented in the MÄK RTI so that it also can be used in RTI 1.3 and 1516 federations.

Using FOM Modules.....	15-2
Merging FOM Modules into the Current FOM	15-3
Creating and Joining with FOM Modules	15-4
Validating FOM Modules	15-5
Configuring Use of Modular FOMs.....	15-6
Configuring a MOM Module	15-6
Configuring Create FOM Modules	15-7
Configuring Join FOM Modules.....	15-7
Configuring Use of a Local FOM Module File.....	15-8
Example FOM Module Configuration for 1.3 and 1516.....	15-8

15.1. Using FOM Modules

The MÄK RTI allows a federation to be formed and modified using FOM modules. The use of FOM modules allows federates with different FOMs to interoperate without having to merge the FOM files. The different FOMs must use the same reference FOM, and have non-conflicting extensions. Using Modular FOMs supports more agile federations, because only FOM modules pertinent to a particular federate or subset of federates need to be introduced into the federation. Federates can specify one or more FOM modules when creating a federation, and more significantly, a federate can specify one or more FOM modules when it joins. Examples of FOM modules are:

- ♦ A normal, fully formed FOM. A federate can create a federation using a FOM just as it always has in the past. It can also submit a FOM when joining, to ensure that the classes it uses are included in the federation.
- ♦ A fully formed FOM that is an extension of an existing FOM. It is common to add new classes to a reference FOM to support new simulation concepts or extensions to existing concepts. The use of FOM modules allows these FOM extensions to be introduced by various federates. The extensions do not have to be integrated into a single superset FOM. However, the extensions must not conflict with each other.
- ♦ A partial FOM containing new or derivative concepts. New or derivative concepts are supplied in a self-contained FOM module that is not a complete FOM. The new or derivative concepts might depend on concepts found in an existing FOM (especially in the case of derivative concepts).



Modular FOMs is not part of either the 1.3 or the 1516 API, so the MÄK RTI configures its use through configuration parameters in the RID file. HLA Evolved federates can specify FOM modules during create and join using the HLA Evolved API.

15.2. Merging FOM Modules into the Current FOM

To help understand how FOM modules work together in an HLA 1.3 or 1516 federation we use the concept of the “current FOM”. The current FOM is the result of merging all the FOM modules that have been submitted by federates. The RTI starts the current FOM with the basic MOM classes. Then it merges in the FOM modules specified by the creating federate and joining federates. (In HLA Evolved federations, modular FOMs work as defined in the specification.)

The MÄK RTI supports three types of FOM modules – the MOM module, the create FOM module, and the join FOM module. A FOM module must satisfy specific merging rules when submitted to the RTI. A valid FOM module can extend the current FOM only by adding subclasses. It cannot extend an existing class by adding attributes or parameters, or changing the datatype or any other FOM characteristic of an existing class or its attributes or parameters. All FOM Modules to be merged must have identical time types and switches.

The MOM module is a special case that relaxes the restrictions on adding class attributes and parameters to existing MOM classes. It is explicitly stated in the HLA standard that the MOM classes can be extended, including adding attributes and parameters. Because the initial current FOM consists of the base MOM classes, the rules for adding class attributes and parameters to existing base MOM classes are allowed, but only through the MOM module. In effect, a MOM module replaces the base MOM classes and it becomes the initial current FOM. It is validated to ensure that it contains the necessary classes for the RTI to operate.

The federate that creates the federation can specify one or more create FOM modules in its RID file. It can also specify one MOM module.



In the HLA Evolved specification, the term MOM module, used in early drafts, and therefore in the initial implementation of modular FOMs in the MÄK RTI, was changed to MOM and Initialization Module, or MIM. If FOM module merging is disabled, a MIM cannot be used. If FOM module merging is enabled, a MIM is always used. The federate can specify the MIM it would like to use in the `createFederationExecution` call. If it does not specify the MIM, the standard MIM is used. The standard MIM is *./bin/HLAstandardMIM.xml*.



Merging FOM modules is non-commutative and non-associative. Therefore, the order in which FOM modules are specified may affect whether or not merging a FOM module with the current FOM is a valid operation. The same is true for join modules. This is true both for the order that modules are specified in a particular RID file and the order in which federates join the federation.

15.3. Creating and Joining with FOM Modules

When the FOM module feature is enabled, the MÄK RTI enhances the behavior of the `createFederationExecution` call by merging FOM modules in addition to the FOM specified in the create call. For HLA 1.3 and 1516, the FOMs are merged in the following order:

1. The MOM module (if specified in the RID file).
2. The FED file specified in the create call.
3. The create FOM modules, in the order specified.

For HLA Evolved, FOMs are merged in the following order:

1. The MIM.
2. The FOM modules, in the order specified in the create call.

The join FOM modules are introduced for merging in a similar fashion. When a federate joins, there is already a current FOM consisting of the base MOM (or MOM module) and at least one FOM module (the FOM submitted in the create call). The join FOM modules are merged into the current FOM in the order specified in the RID file.

If you use FOM modules, the RTI must distribute the FED file (specified by the `RTI_distributeFedFile` parameter). This ensures that every federate has the complete set of FOM data being used by the federation. The use of FOM modules also requires the use of the `rtiexec` and that internal messages be sent reliably (`RTI_useRtiExec` and `RTI_internalMsgReliableWhenUsingRtiexec`).

If a federate tries to create a federation that already exists, any FOM modules it specifies have no effect on the current FOM. Additionally, the create and join operations, including the merging of the FOM modules, are an atomic action. A failure to merge a single FOM module results in the failure of the entire call and the current FOM does not change.

15.4. Validating FOM Modules

A FOM module merge requires that the time type and switches tables be identical. A valid merge also requires that a class definition must be complete. A valid module cannot extend a class with new attributes or parameters that have already been defined in another module; it can only extend the FOM by adding new classes.

If a module contains definitions of classes that already exist in the FOM, they must be identical to the existing class definitions or they must be scaffolding definitions. A scaffolding class does not replace existing class definitions nor does it have to match them. However, if a FOM with scaffolding classes is introduced out of order, its empty class definition takes precedence and subsequent complete class definitions will be flagged as an invalid merge.



A scaffolding definition is one that describes the class hierarchy and inheritance information of a class without specifying any attributes or parameters of the class. Since an object class definition must include its entire parentage, but may not be specifying the object classes of its parentage, often a FOM module will include so-called scaffolding class definitions, which do not include any object or interaction class information beyond the hierarchy.

To restate the basic merging rule, once an object or interaction class is added to the current FOM, its attributes and parameters cannot be changed. Merging a non-scaffolding object or interaction class definition requires that the object or interaction class be absent in the current FOM or, if present, match the existing class attributes and parameters exactly.

15.5. Configuring Use of Modular FOMs

The use of FOM modules is configured in the RID file with the `RTI_fomModuleMerging` parameter.

For HLA 1.3 and 1516, the presence of any of the FOM module functions in the RID file enables the FOM module functionality. For HLA Evolved, `RTI_fomModuleMerging` must be enabled to use FOM modules. The use of FOM modules also requires enabling the `rtiexec`, internal message reliable transport, and FED file distribution.

It is not necessary for each federate to have the same entries in its RID file. However, the use of any FOM module must adhere to the modular FOM merging rules described in “[Merging FOM Modules into the Current FOM](#),” on page 15-3.



- ♦ The previous paragraphs in this section apply to all three HLA specifications. However everything else in this section, including the following paragraph applies only to HLA 1.3 and 1516.
 - ♦ Not all modular FOM configuration parameters affect a federation. For example, `RTI-addCreateFomModule` and `RTI_momModuleExtensionFileName` only affect the federation if the federate creates the federation. A MOM module extension or create FOM module of a federate that joins an already created federation have no effect on the federation.
-

15.5.1. Configuring a MOM Module

To specify a MOM module file, use the `RTI_momModuleExtensionFileName` parameter in the RID file. To be executed, the federate must create the federation execution. The syntax is:

```
(setqb RTI_momModuleExtensionFileName "momModuleName")
```

where *momModuleName* is a quoted string that specifies the MOM module file name. The file name can be an absolute path, a path relative to the working directory, or just the filename if it is in the working directory.

The specified MOM module can contain extensions to the existing MOM classes. The module specified in this parameter can be considered the first FOM module specified, because it will be merged into the current FOM before any other modules specified in the RID file. This parameter is optional and the default base MOM classes are used if omitted.

15.5.2. Configuring Create FOM Modules

Create FOM modules are FOM modules specified by the federate that creates the federation. To specify a create FOM module, use the `RTI-addCreateFomModule` function in the RID file. The syntax is:

```
(RTI-addCreateFomModule "fomModuleName")
```

where *fomModuleName* is a quoted string that specifies a FOM module file name. The file name can be an absolute path, a path relative to the working directory, or just the filename if it is in the working directory.

Each function specifies a single create FOM module. You can include multiple entries in the RID file. The order that they appear in the file is the order they are merged into the current FOM. The RTI will try to merge the contents of each FOM module into the current FOM during the create federation execution service call. If successful, the FOM module becomes part of the current FOM supplied to federates when they join. If unsuccessful, the create call fails.

15.5.3. Configuring Join FOM Modules

Join FOM modules are FOM modules specified by a federate that is joining a federation. To specify a join FOM module, use the `RTI-addJoinFomModule` function in the RID file. The syntax is:

```
(RTI-addJoinFomModule "fomModuleName")
```

where *fomModuleName* is a quoted string that specifies a FOM module file name. The file name can be an absolute path, a path relative to the working directory, or just the filename if it is in the working directory.

Each function specifies a single join FOM module. You can include multiple entries in the RID file. The order they appear in the file determines the order that they are merged into the current FOM. The RTI will try to merge the contents of each FOM module into the current FOM during the join federation execution service call. If successful, the FOM module becomes part of the current FOM supplied to other federates when they join. If unsuccessful, the join call fails.

15.5.4. Configuring Use of a Local FOM Module File

When a federate submits a FOM module in a create or join call, the file is transmitted to the rtiexec. The rtiexec sends the FOM module back to the federate when it joins. (The FOM module is also sent to federates that have already joined and to subsequent joiners.) A federate can use its local copy of a FOM module instead of having it sent back from the rtiexec. The `RTI_preferLocalFomModule` RID parameter controls whether a LRC uses its local copy or requests a new copy from the rtiexec for any FOM module files that it has submitted. The syntax is:

```
(setqb RTI_preferLocalFomModule enableOption)
```

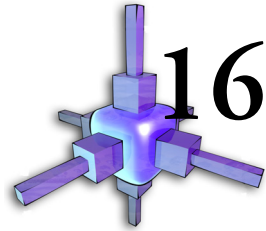
where *enableOption* is 1 to enable (use local copy) and 0 to disable (rtiexec retransmits copy).

15.5.5. Example FOM Module Configuration for 1.3 and 1516

The following example configures the RTI to merge the contents of *momExtFomModule.xml* with the FOM specified in the create federation execution service call. After the MOM module is merged into the FOM, a create module, *createModule1.xml*, is merged into the current FOM. When the federate joins, two join FOM modules, *joinFomModule1.xml* and *joinFomModule2.xml*, are merged into the current FOM. The LRC uses its local copy of the FOM modules to perform merges.

```
(setqb RTI_fomModuleMerging 1)
(setqb RTI_useRtiExec 1)
(setqb RTI_internalMsgReliableWhenUsingRtiexec 1)
(setqb RTI_distributeFedFile 1)
(setqb RTI_preferLocalFomModule 1)

(setqb RTI_momModuleExtensionFileName "momExtFomModule.xml")
(RTI-addCreateFomModule "createModule1.xml")
(RTI-addJoinFomModule "joinFomModule1.xml")
(RTI-addJoinFomModule "joinFomModule2.xml")
```



Update Rate Reduction

Update rate reduction is functionality from the HLA Evolved specification. It has been implemented in the MÄK RTI so that it can also be used with RTI 1.3 or RTI 1516 federates.

Update Rate Reduction.....	16-2
How Update Reduction is Implemented	16-4
Configuring Update Rate Reduction.....	16-5
Specifying the Update Rate in 1.3 and 1516.....	16-5
Specifying the Rate for Object Class Subscriptions in 1.3 and 1516	16-6
Specifying the Receive Filtering Tolerance	16-6
Example Update Rate Reduction Configuration (1.3 and 1516 Only) .	16-7

16.1. Update Rate Reduction

Update rate reduction is a mechanism that allows federates to specify the rate at which best effort attribute updates can be delivered to the federate. This feature augments the declaration management (DM) and data distribution management (DDM) services that provide mechanisms for reducing or filtering the data that a federate receives based on class and data content. Update rate reduction does not affect interactions or reliable attribute updates.

The DM and DDM services provide mechanisms for reducing or filtering the data that a federate receives. The DM services provide filtering on the type of data; the DDM services provide filtering on the content or value of the data (that is, whether the value of the data is within a given range). It may be that even with these powerful filtering mechanisms, a federate might still not be able to keep up with the data that meet its filter criteria. In these cases, the federate would be better off receiving some sampling of the data than being overloaded, falling behind, and overflowing its buffers.

An example of this requirement could be a low fidelity plan view display federate that is viewing an entire simulation. It is not interested in the detailed maneuvers and interactions of the simulation, but in the general macro view of the entity placement and long term movements. While this federate may require a sample of all the data (both type and value), it does not require the data at the resolution that is necessary for entity level simulations. [Figure 16-1](#) illustrates the update rate reduction for this example.

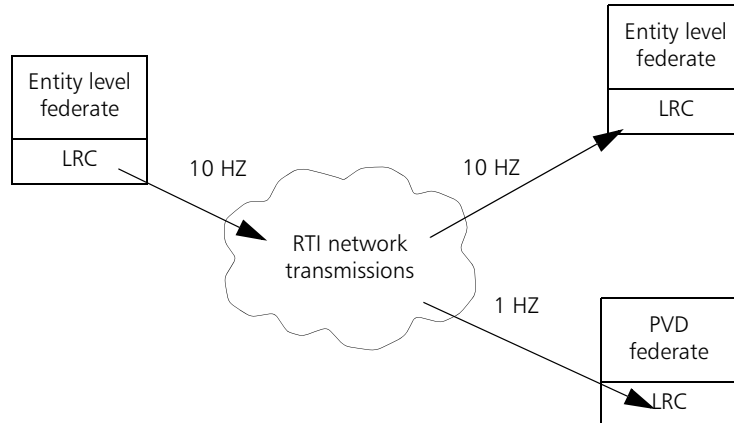


Figure 16-1. Update rate reduction

By using update rate reduction, the federate can specify a hard limit on the rate at which the RTI delivers attribute updates from each object instance. The RTI is responsible for delivering attribute updates for each object to the federate up to the specified rate. There is no guarantee that the RTI will send attribute updates at a given rate, just that the rate will not be exceeded. The criterion used for delivering updates is based on the specified rate and message inter-arrival times. Given a maximum update rate of x Hz and a message that has been delivered at wall-clock time t , no message shall be delivered before $t + (1 / x)$.

The update rate applies to each object attribute instance independently. That means that in practice, the rate that attributes can be reflected (where a reflect contains the attributes from one object) is the number of discovered objects times the update rate.

i

- ♦ The maximum rate would actually be:
`number_of_attributes * number_of_objects * rate.`
However, this maximum is a degenerate case in that it would require the sending federates to update all the object attributes (in practice just a few) each independently of the others (in practice in a single update).
 - ♦ If no update rate reduction is specified for a federate, the federate will receive attribute updates at whatever rate they are sent.
-

16.2. How Update Reduction is Implemented

The MÄK RTI implements update reduction using multicast and receive filtering. The implementation relies on the DM multicast filtering feature that allows object classes to be assigned multicast addresses. (For details, please see “[Filtering by Multicast Groups Using Declaration Management](#),” on page 10-9.) A set of update rates is then defined in the RID file (“[Configuring Update Rate Reduction](#),” on page 16-5) and each rate is assigned an offset. The combination of an object class’s multicast address and update rate offset determine an update rate channel.

When sending an attribute update message, the RTI inspects the set of attributes and determines the interval since each attribute was sent. It selects the greatest interval, and sends the update to the appropriate update rate channel.

On the subscriber side, when a federate subscribes to object class attributes, the RTI inspects the update rate subscription specified for the subscribed class. The RTI then joins the update rate channels for the specified rate and all slower rates. The result is that an update sent at the specified rate or slower will be received by the federate and the sum of the updates on all the update rate channels will equal the subscribed update rate, as illustrated in [Figure 16-2](#). This multicast filtering provides an efficient means for the LRC to receive and process only those update messages that will be delivered to the federate. However, to compensate for timing issues, the LRC also performs receive side filtering to further guarantee the specified rate.

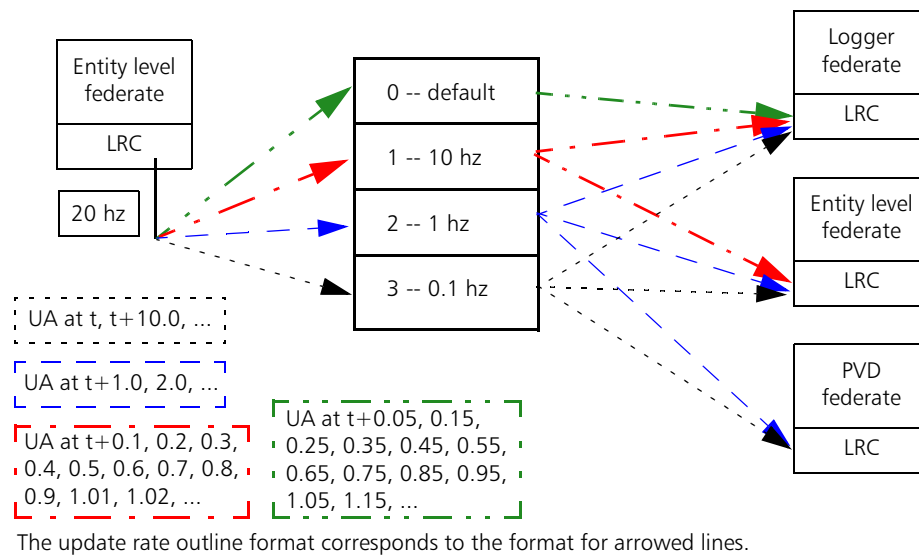


Figure 16-2. Update rate routing

16.3. Configuring Update Rate Reduction

To use update rate reduction, you must specify a set of update rates and associate them with object class attribute subscriptions. You must also configure use of DM multicast filtering. The update rates are specified as named values given in hertz (updates per second). These named values are then associated with object classes with a nesting operator similar to that used in the configuration of DM multicast filtering (that is, inclusive indicates the specified class and all subclasses and exclusive indicates just the specified class). The DM multicast addresses must be assigned to leave room for the update rate offsets, including a base address of no rate.

For example, if three update rates are specified, then each DM multicast assignment must have at least 4 addresses separating it from any other (for example, 229.7.7.10, 229.7.7.14, 229.7.7.18, and so on). You can specify a receive tolerance to allow fine tuning of receive filtering versus multicast filtering.

The update rate reduction is configured in the RID file.

- To enable update rate reduction set `RTI_enableUpdateRateReduction` to 1.

For detailed explanation of DM multicast configuration, please see “[Filtering by Multicast Groups Using Declaration Management](#),” on page 10-9.

16.3.1. Specifying the Update Rate in 1.3 and 1516

- Specify update rates with the function `RTI-addUpdateRate`. Include an entry for each rate as follows:

```
(RTI-addUpdateRate "name" rate)
```

where *name* is a quoted unique string and *rate* is a floating point value specifying the rate in hertz.



In HLA Evolved, the update rates are specified in the FDD file.

16.3.2. Specifying the Rate for Object Class Subscriptions in 1.3 and 1516

You can specify an update rate for object class subscription with the RID function `RTI-addUpdateRateSubscription`. Add an entry for each class that should have update rate reduction applied to its subscription. The nesting qualifier (`inclusive` or `exclusive`) is a shortcut that allows the rate to be applied to all subclasses of the specified class. The entries are applied in the order they appear in the RID file, and the nesting operators override previous operations. For example, a top level class could specify a rate as `inclusive`, meaning it applies to all subclasses. Selected subclasses could override that rate with `exclusive` entries (that apply only to themselves), or `inclusive` entries (that apply to themselves and their subclasses).

The syntax for `RTI-addUpdateRateSubscription` is as follows:

```
(RTI-addUpdateRateSubscription "class" "rateName" "nesting")
```

where:

- `class` is a quoted string specifying an object class name, which must be in the joined FOM
- `rateName` is a quoted string specifying the name of an update rate, which must be specified in an `RTI-addUpdateRate` entry previously in the RID file
- `nesting` is a quoted string specifying either `inclusive` or `exclusive`, where `inclusive` indicates the rate applies to the specified class and all subclasses and `exclusive` indicates the rate applies to just the specified class.



In HLA Evolved, the subscription rate is specified using the API when a federate subscribes to object class attributes.

16.3.3. Specifying the Receive Filtering Tolerance

Specify the receive-filtering tolerance with the `RTI_receiveTolerance` parameter. This parameter is a percentage reduction to the inter-arrival time between attribute updates. It reduces the inter-arrival time for a given rate so that an update that appears to arrive early at the receiver is accepted. Given a rate of 10 hertz, a tolerance of 0.1 would allow the LRC to accept an update arriving 0.09 seconds after a previous update. This tolerance value allows for differences in timing measurements between sending and receiving federates as well as allowing for latency fluctuations (that is, previous update was late, but the next is received faster). The syntax for `RTI_receiveTolerance` is as follows:

```
(setqb RTI_receiveTolerance reductionPercentage)
```

where `reductionPercentage` is a floating point value specifying the percentage to reduce the inter-arrival interval used for receive filtering.

16.3.4. Example Update Rate Reduction Configuration (1.3 and 1516 Only)

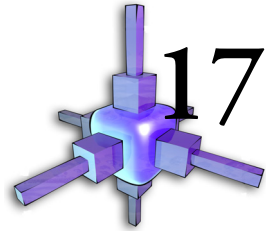
Here is an example configuration of update rate reduction using all the necessary parameters:

```
(setqb RTI_enableUpdateRateReduction 1)
(RTI-addDMObjectMulticastAddr "ObjectRoot.BaseEntity" "229.7.8.1"
 "inclusive" "0.0.0.0")
(RTI-addDMObjectMulticastAddr "ObjectRoot.BaseEntity.GroundVehicle"
 "229.7.8.5" "exclusive" "0.0.0.0")
(RTI-addDMObjectMulticastAddr "ObjectRoot.BaseEntity.Aircraft"
 "229.7.8.9" "exclusive" "0.0.0.0")

(RTI-addUpdateRate "high" 10.0) ;; up to 10 updates per second
(RTI-addUpdateRate "medium" 1.0) ;; up to 1 update per second
(RTI-addUpdateRate "low" 0.1) ;; up to 1 update every 10 seconds

(setqb RTI_receiveTolerance .01)

(RTI-addUpdateRateSubscription "ObjectRoot.BaseEntity" "low" "inclusive")
(RTI-addUpdateRateSubscription "ObjectRoot.BaseEntity.GroundVehicle"
 "medium" "exclusive")
(RTI-addUpdateRateSubscription "ObjectRoot.BaseEntity.Aircraft" "medium"
 "exclusive")
```

The RTIs Spy Plug-in API

The RTIs Spy plug-in API for the MÄK RTI allows you to monitor the actions of the MÄK RTI and to modify its behavior.

Introduction	17-2
The RTI Managers.....	17-3
Initializing a Plug-in	17-4
Monitoring RTI Service Invocations	17-6
Monitoring Federate Ambassador Services	17-7
Creating Custom RTI Managers	17-9
Subclassing DtRtiAmbassadorImplementor	17-10
Communicating with Remote LRCs and the rtiexec	17-11
Time Factory Wrapper.....	17-11
The Connection Manager	17-12
The Asynchronous I/O Connection Manager.....	17-13
Changing the RTI's Communications Infrastructure.....	17-14
Messages.....	17-17
Responding to Dropped Federates	17-19
Responding to Missing Heartbeats	17-19
Finding Out when Data is Available to be Read	17-20
rtiexec Plug-ins	17-21
Accessing RID File Parameters	17-22
Compiling and Linking	17-23
Compiling and Linking on UNIX.....	17-23
Compiling and Linking on Windows	17-24
Loading Plug-Ins.....	17-24

17.1. Introduction

The RTIspy API is an interface to the internals of the MÄK RTI. Using the API, you can develop plug-ins – dynamically linked libraries that can monitor, extend, or alter the functionality of the MÄK RTI. You can develop custom RTIs using the MÄK RTI as a starting point. For example, within your plug-ins you can monitor the RTI service calls, change the “wire format” used by the RTI on the network, implement communication mechanisms other than the default UDP and TCP networking schemes, or make performance optimizations based on the specific needs of your federation. Plug-ins are loaded by the `rtiexec`, by each LRC, or by both, when instructed to do so by the RID file. The LRC web client, printer example, and implicit DDM feature are plug-ins.

The RTIspy API does not replace or change the external RTI API, which is based on the HLA Standard Interface. Federates still interact with the RTI through the `RTI::RTIambassador` and `RTI::FederateAmbassador` classes, with their familiar functions like `sendInteraction()` and `updateAttributeValues()`. The plug-in API just allows you to change the implementation of these standard RTI functions. Federate code does not need to change to use an RTI that has been extended with a plug-in, since federates typically do not need to be modified even when switching to an entirely different RTI implementation.

Generally speaking, the job of an RTI is to implement the `RTI::RTIambassador` portion of the RTI interface specification. The implementation of a particular `RTI::RTIambassador` function might change the state of the LRC, cause data to be sent to remote LRCs, or – as in the case of `tick()` – cause `RTI::FederateAmbassador` functions to be invoked. In the MÄK RTI, implementation of `RTI::RTIambassador` function calls is managed by an object called a *DtRtiAmbassadorImplementor* (*ambImp.h*). The *DtRtiAmbassadorImplementor* has an API that mirrors the external `RTI::RTIambassador` API. When an `RTI::RTIambassador` service is called by the federate, the ambassador calls down to the corresponding function in the *DtRtiAmbassadorImplementor* to execute the function. The *DtRtiAmbassadorImplementor* in turn, delegates most function calls to various manager objects within the RTI.

When a federate instantiates `RTI::RTIambassador`, it creates an instance of *DtRtiAmbassadorImplementor*, which is passed to your plug-in. Through this *DtRtiAmbassadorImplementor* object, your plug-in can register service call observers and get to all of the RTI’s managers.

i

The MÄK RTI classes rely on some classes and utilities from the VR-Link *vlutil* and *mtl* libraries. To build plug-ins, you must install a copy of the VR-Link version that the MÄK RTI was built against. You do not need a VR-Link license.

17.2. The RTI Managers

As mentioned in the previous section, the *DtRtiAmbassadorImplementor* delegates to, or uses, various manager classes and other objects to perform most of the actual implementation of RTI services. The managers are also where most of the state of the LRC resides. The managers are:

- ♦ *DtFederateManager* (defined in *fedMgr.h*) manages most of the federation management services, like join, resign, and synchronization points, as well as tick(). It also reads the FED file (which occurs within a join call).
- ♦ *DtObjectManager* (*objMgr.h*) manages most of the object management and ownership management services. It also handles the object-related declaration management functions, in conjunction with the *DtFomClassManager*. It maintains a list of *DtRtiObjects* (*obj.h*) representing all known objects in the federation. Each *DtRtiObject* stores all of the bookkeeping information associated with that object – known class, actual class, ownership information for each attribute instance, and so on.
- ♦ *DtInteractionManager* (*interMgr.h*) handles the interaction-related declaration management and object management services.
- ♦ *DtTimeManager* (*timeMgr.h*) manages time management services and synchronizes the advancement of time with the federation execution.
- ♦ *DtDataDistMgr* (*ddmMgr.h*) manages the DDM services that directly affect regions (for example, create, delete, and notification). The object and interaction managers use specialized derived classes (which add DDM functionality) to handle the DDM services that are tied to objects and interactions (for example, register object with region and send interaction with region).
- ♦ *DtMomManager* (*momMgr.h*) manages the RTI's MOM objects (federation and federate) and handles the MOM interactions.
- ♦ *DtConnectionMgr* (*connectMgr.h*) manages the RTI's connections to other federates and to the LRC. In the default implementation, connections are network based, but in subclasses, other kinds of connections are possible.
- ♦ *DtFomClassManager* (*fomMgr.h*) manages a hierarchy of class descriptors (*DtObjectClassInfo*, and *DtInterClassInfo*, defined in *objClassInfo.h* and *intClassInfo.h* respectively) mirroring the FOM's class hierarchy. (The *DtFomClassManager* is initialized by reading the FED file.) These objects store elements of RTI state that are associated with FOM classes, attributes and parameters, such as, information about which classes and attributes are currently published and subscribed, default transport type for each attribute, and so on. This manager also implements the name and handle mapping services of the RTI.

The *DtFederateManager* is instantiated by the RTI ambassador *DtRtiAmbassadorImplementor*. The *DtFederateManager* instantiates the *DtConnectionMgr*. The *DtFederateManager* and *DtConnectionMgr* are needed by both joined and unjoined federates. The other managers are not instantiated until the join-FederationExecution() call, since they are only used by a joined federate, and depend on FOM data, which is not available until the federate is joined.

The RTI adds DDM functionality to the *DtObjectManager* and the *DtInteractionManager* by deriving new classes *DtObjectDataDistMgr* (*objDdmMgr.h*) and *DtInterDataDistMgr* (*interDdmMgr.h*). These classes are further derived for the 1.3 and 1516 specifications. When DDM services are enabled, these classes are instantiated instead of the base classes. When DDM services are disabled, the base classes are instantiated.

Pointers to all of the managers are available through *DtRtiAmbassadorImplementor*'s accessors.

For information about the various functions that you can use to inspect or alter the portion of the LRC's state managed by each manager, please see the class definitions for the managers.

i

The 1516 RTIspy APIs have the same internal components and managers as the 1.3 RTIspy API. The classes may have been modified to some degree (for example, RTI type replacement), but they essentially perform the same functions.

17.3. Initializing a Plug-in

All MÄK RTI plug-in libraries must export one or more of the following three “C” style functions. The LRC or the rtiexec looks for these functions when a plug-in is loaded, and executes them at the appropriate times.

- ♦ *SetPluginCreators()* is called immediately when a plug-in is loaded, before the *DtRtiAmbassadorImplementor* or any of the managers are created. This is the place to inform the LRC or the rtiexec that it should instantiate derived versions of these classes instead of the default versions. For details, please see [“Creating Custom RTI Managers,”](#) on page 17-9, and [“Subclassing DtRtiAmbassadorImplementor,”](#) on page 17-10. If you are not subclassing any of these classes, you do not need to include *SetPluginCreators()* in your plug-in.
- ♦ *InitLRCPlugin()* is called by the LRC after the *DtRtiAmbassadorImplementor*, *DtFederateManager*, and *DtConnectionMgr* are created. The *DtRtiAmbassadorImplementor* is passed to *InitLRCPlugin()*. Through the implementor, you can register RTI service call observers or get pointers to all of the other managers.
- ♦ *InitRtiexecPlugin()* is called by the rtiexec after the *DtRtiexec* is instantiated. A pointer to the *DtRtiexec* is passed to *InitRtiexecPlugin()*. Through this object, you can register callbacks on creation and destruction of federation executions and federates, and inspect other aspects of the rtiexec's state. For details, please see [“rtiexec Plug-ins,”](#) on page 17-21.



For MÄK RTI 1516, use `SetPluginCreators1516()`, `InitLRCPlugin1516()`, and `InitRtiexecPlugin1516()`. For HLA Evolved, use `SetPluginCreators1516e()`, `InitLRCPlugin1516e()`, and `InitRtiexecPlugin1516e()`.

You can use a single plug-in library for both the LRCs and the rtiexec. Each ignores the `Init*()` function that does not apply to it. Alternatively, you could create separate plug-ins for the rtiexec and for the LRCs. In this case, there is no reason to include `InitRtiexecPlugin()` in the LRC's plug-in, or `InitLRCPlugin()` in the rtiexec's plug-in.

The following example shows the expected function prototype for each of the three functions, and the mechanics of exporting your functions:

```
#if WIN32
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

extern "C"
{
    void EXPORT SetPluginCreators()
    {
        // Register your derived class creator functions here
    }

    void EXPORT InitLRCPlugin(DtRtiAmbassadorImplementor* imp)
    {
        // Use imp to get pointers to managers, register observers, and so on
    }
    void EXPORT InitRtiexecPlugin(DtRtiExec* exec)
    {
        // Use exec to register callbacks on new federates, and so on.
    }
}
```

17.4. Monitoring RTI Service Invocations

The key to finding out when RTI Ambassador functions are called by a federate, is the *DtRtiAmbassadorObserver* class (*ambObserver.h*, *ambObserver1516.h*, and *ambObserver1516e.h*). *DtRtiAmbassadorObserver* is a base class whose interface closely resembles that of the RTI::RTIAmbassador. You can create a subclass of *DtRtiAmbassadorObserver*, and provide implementations for any functions you want to monitor. For example, if you want to print a message every time an interaction is sent, override *sendInteraction()* in your observer with a version that will do so. Then, register an instance of your observer with the RTI, by calling *addObserver()* on the *DtRtiAmbassadorImplementor*, typically from within *InitPlugin()*. Whenever an RTI Ambassador function is called by a federate, the call is executed, then the RTI calls the comparable function on any *DtRtiAmbassadorObservers* that are currently observing the RTI.

The following example shows how to create a subclass of *DtRtiAmbassadorObserver*, and register it with the RTI:

```
class MyObserver : public DtRtiAmbassadorObserver
{
public:

    // Override any base class functions you choose, to be notified
    // when the corresponding RTI::RTIAmbassador function is called
    virtual void sendInteraction (
        RTI::InteractionClassHandle theInteraction,
        const RTI::ParameterHandleValuePairSet& theParameters,
        const char *theTag, RTI::Exception* except = NULL)
    {
        printf("SendInteraction called for class %d\n", theInteraction);
    }
    ...
};

void InitLRCPlugin(DtRtiAmbassadorImplementor* implementor)
{
    // Create an instance of your observer, and register it with the
    // implementor.
    MyObserver* observer = new MyObserver();
    implementor->addObserver("MyObserver", observer);
};
```

The RTI deletes your observer when *DtRtiAmbassadorImplementor* is destroyed, so you should not try to delete the observer yourself if it is registered with the RTI. If, during execution, you want to unregister your observer, use *DtRtiAmbassadorImplementor::removeObserver()*. If you remove an observer, you are responsible for deleting it. You unregister an observer by name.

You can register more than one observer with the RTI. Appropriate functions will be called on all of them.

DtRtiAmbassadorObserver's function prototypes differ slightly from those in the `RTI::RTIambassador` (the RTI's external API). Most functions include an optional `RTI::Exception*` argument, and some include arguments that match the return types of the corresponding `RTI::RTIambassador` functions. This is to allow your code to learn not only what RTI calls are being made by a federate, but what the results of those calls are. If a non-NULL value is passed as the **except** argument to a *DtRtiAmbassadorObserver* function, that means the federate's call to the RTI service generated an exception. The **except** argument is a copy of the `RTI::Exception` object that was thrown to the federate. For `RTI::RTIambassador` service calls that generate return values, the value returned to the federate is passed to the observer's function as an extra argument. For example, `DtRtiAmbassadorObserver::registerObjectInstance()` takes an argument called **handle**, which represents the handle that was returned to the federate when it called `RTI::RTIambassador::registerObjectInstance()`.

The MÄK RTI's implementation of the management object model (MOM) uses an observer to generate `ReportServiceInvocation` interactions.

17.4.1. Monitoring Federate Ambassador Services

Being notified when the RTI invokes `RTI::FederateAmbassador` services works slightly differently from `RTIambassador`. You must subclass *DtFedAmbWrapper* (*fedAmbWrap.h*, *fedAmbWrap1516.h*, and *fedAmbWrap1516e.h*), and register an instance of your subclass with the RTI using `DtRtiAmbassadorImplementor::addFedAmbWrapper()`.

DtFedAmbWrapper is derived from `RTI::FederateAmbassador`, so it inherits the Federate Ambassador's public API. Normally, when the RTI wants to make calls to a federate, it invokes the appropriate functions on the `RTI::FederateAmbassador` object that has been passed to the RTI by the federate (for example, `reflectAttributeValues()`). However, if your plug-in has a federate ambassador wrapper registered with the RTI, federate ambassador calls are made on your wrapper instead, allowing you to intercept calls to the federate ambassador.

Because you are intercepting calls that were destined for the federate, it is your responsibility to make sure that calls still get made on the federate's original `RTI::FederateAmbassador`. To help you do this, *DtFedAmbWrapper* maintains a pointer to that original ambassador, which we refer to as its child ambassador. Furthermore, the default implementation of all *DtFedAmbWrapper* functions is to call the corresponding function on its child ambassador. So for functions that you do not specifically override in your *DtFedAmbWrapper* subclass, you do not have to do anything special to make sure that the federate gets the data it needs. But for functions that you override, you must call down to the base *DtFedAmbWrapper*'s version of the function at some point within your function.

The following example creates and registers a *DtFedAmbWrapper* that intercepts the `receiveInteraction` call, and prints a message both before and after the federate's version of this function is executed:

```
class MyFedAmbWrapper : public DtFedAmbWrapper
{
public:

    virtual void receiveInteraction (
        RTI::InteractionClassHandle theInteraction,
        const RTI::ParameterHandleValuePairSet& theParameters,
        const char* theTag)
    throw (
        RTI::InteractionClassNotKnown,
        RTI::InteractionParameterNotKnown,
        RTI::FederateInternalError)
    {
        printf("About to call receiveInteraction.\n");

        // Make sure to call down to the base class so that the
        // "real" Federate Ambassador function is called.
        DtFedAmbWrapper::receiveInteraction(
            theInteraction, theParameters, theTag);

        printf("Just called receiveInteraction.\n");
    }
};

void InitLRCPlugin(DtRtiAmbassadorImplementor* implementor)
{
    // Create an instance of your observer, and register it with the
    // implementor
    MyFedAmbWrapper* wrapper = new MyFedAmbWrapper();
    implementor->addFedAmbWrapper(wrapper);
};
```

As long as a wrapper is registered with the RTI, the RTI deletes it when the *DtRtiAmbassadorImplementor* is destroyed. However, you can remove a wrapper at any time using *DtRtiAmbassadorImplementor*'s `removeFedAmbWrapper()`, at which point you regain ownership of the wrapper.

You can chain federate ambassador wrappers together, so that you can have, for example, an outer wrapper wrapping an inner wrapper, which wraps the federate's original `RTI::FederateAmbassador` (arbitrary length chains are supported). If you register multiple wrappers, each subsequent wrapper becomes the new top-level wrapper, and its child ambassador automatically becomes the previous wrapper. The innermost wrapper has the federate's original ambassador as its child. In effect, outer-wrappers intercept calls from lower-level wrappers, but if you always call down to the base *DtFedAmbWrapper*'s version of each function that you override, you will ensure that each wrapper invokes its child ambassador's version of the function, and that the federate's version will eventually be called.

17.5. Creating Custom RTI Managers

One powerful way of customizing the RTI's implementation through the plug-in API is by subclassing one or more of the RTI's managers, overriding appropriate functions, and then telling the RTI to instantiate your subclass instead of the default implementation of the manager. Regardless of which manager you want to override, when subclassing any of the managers, the procedure is similar:

1. Include a static creator function that returns a new'd instance of your subclass:

```
class MyInterMgr : public DtInteractionMgr
{
public:
    // Creator function
    static DtInteractionMgr* create(DtFederateMgr* fedMgr,
        DtFomClassMgr* fom, DtRIDParameters* params)
    {
        return new MyInterMgr(fedMgr);
    }
    // Provide the constructor and override the functions you choose
};
```

2. Within SetPluginCreators(), register your subclass's creator function with the RTI using the manager's static setCreatorFunction():

```
// Register your new manager subclass's creator function with the RTI
DtInteractionManager::setCreatorFunction(MyInterMgr::create);
```

When it is time for the RTI to instantiate an interaction manager, it will use your create function, which will result in the creation of a MyInterMgr instance.

All of the managers have similar setCreatorFunction() member functions.



The setInternalCreatorFunction() is used internally by the RTI. It should not be used by plug-ins, because the assignment might be overridden internally. Use the setCreatorFunction().

When you derive from the MÄK RTI's managers, it is important to understand that the RTI itself sometimes uses subclasses of *DtInteractionMgr*, *DtObjectMgr*, and *DtConnectionMgr* rather than using those base classes directly. Therefore, if you need to extend or override default functionality you may need to derive from the RTI's subclasses rather than directly from these base manager classes.

Specifically, if DDM is disabled, the RTI uses *DtObjectMgr* and *DtInteractionMgr*. If DDM is enabled, it uses *DtObjectDataDistMgr* (*objDdmMgr.h*) and *DtInterDataDistMgr* (*interDdmMgr.h*) plus a derived class specific to the HLA specification (for example, *DtObjectDataDistMgr13* (*objDdmMgr13.h*) and *DtInterDataDistMgr13* (*interDdmMgr13.h*). Similar classes exist for the 1516 specifications. If your federation uses DDM, you should derive from one of the classes for the HLA specification that you are using.

Similarly, the RTI normally uses an instance of *DtAsyncConnectionMgr* (*asyConMgr.h*), rather than the base *DtConnectionMgr*. (*DtAsyncConnectionMgr* adds to *DtConnectionMgr* the ability to handle asynchronous I/O.) If you want to extend or alter the connection manager while preserving the ability to handle asynchronous I/O, derive your class from *DtAsyncConnectionMgr* rather than from the base *DtConnectionMgr*.

17.6. Subclassing DtRtiAmbassadorImplementor

You will probably not want to completely override the way particular RTI services are implemented by *DtRtiAmbassadorImplementor*. It is more likely that you will want to allow the default logic to be executed and just alter the way some of the specific tasks are handled. For example, to change the way the RTI communicates, you can subclass *DtConnectionMgr*. But the basic logic, for example, the fact that a call to *updateAttributeValues()* causes us to check various exception conditions, and then create and send one or more messages, remains the same.

However, if you want to override RTI service implementations wholesale, so that you can have full control over how the services are implemented, you can.

DtRtiAmbassadorImplementor's interface mirrors that of the real RTI::RTIAmbassador class, except that all of its functions are virtual, and therefore can be overridden in a derived class. If you want to re-implement RTI Ambassador functions, the process is essentially the same as for overriding managers, discussed in “[Creating Custom RTI Managers](#),” on page 17-9.

1. Create a subclass of *DtRtiAmbassadorImplementor* and override the functions you choose:

```
class MyImplementor : public DtRtiAmbassadorImplementor
{
public:

    // Overridden versions of any RTI Ambassador services you choose.
    // ...

};
```

2. Use *SetPluginCreators()* to tell the RTI to use an instance of your subclass instead of the default implementor:

```
// Register your new implementor subclass's creator function with the RTI
DtRtiAmbassadorImplementor::setCreatorFunction(MyImplementor::create);
```

Remember that when you override implementations of RTI services, you must keep the RTI's internal state correct and consistent, otherwise other services may fail to work properly.



You do not need to override the implementation of a service to be notified that it has been called. If you just want some code of yours to be executed whenever particular RTI services are invoked, please see “[Monitoring RTI Service Invocations](#),” on page 17-6.

17.7. Communicating with Remote LRCs and the rtiexec

To understand how communication works in the MÄK RTI, you need to understand:

- ♦ Connections. A connection, represented by the *DtRtiConnection* class, is a logical connection between the local LRC and other federates' LRCs and the rtiexec (if you are running it). *DtRtiConnections* are used to send and receive RTI messages.
- ♦ Messages. Messages are represented by instances of subclasses of *DtRtiMsg* (*rtiMsg.h*).
- ♦ The Connection Manager. The *DtConnectionMgr* is responsible for creating and managing the LRC's connections.



If you are familiar with the VR-Link API, you will see that a *DtRtiConnection* is analogous to the *DtExerciseConn* in the DIS version of VR-Link. Similarly, *DtRtiMsg* and its subclasses closely resemble VR-Link's *DtPdu* and its subclasses.

17.7.1. Time Factory Wrapper

The definition of time is different in the 1.3, 1516, and HLA Evolved APIs. Internally, the RTI must use the HLA API for time to handle time management and to process other services that deliver time parameters. The MÄK RTI creates a wrapper, *DtLogicalTimeFactory*, around the different time factories to provide a neutral interface to either implementation.

17.7.2. The Connection Manager

When a piece of code within the RTI needs to send a message to other federates or the rtiexec, it creates an instance of the appropriate kind of *DtRtiMsg*, and asks the *DtConnectionMgr* to send it by calling its `sendStamped()` or `sendStampedAndDelete()` function. Based on the value of the **transportType** argument to these functions (either *DtBest_Effort* or *DtReliable*), the *DtConnectionMgr* sends the message through an appropriate connection. For internal RTI bookkeeping messages (for example, `publish` and `subscribe`), the **transportType** passed to the send functions should be the value returned by `DtConnectionMgr::internalTransportType()`, which is dictated by your RID settings.

The *DtConnectionMgr* creates and manages up to two logical connections – a best effort connection and a reliable connection. (If your RID settings indicate that all data should be sent best effort, the reliable connection is not created.) Messages are sent through these connections without regard to how the connections are actually implemented. In other words, just because data is sent through the reliable logical connection, it is not guaranteed that it goes out over a reliable physical connection. Connections are created in the virtual method `DtConnectionMgr::initConnections()`. Prior to making that call, the Connection Manager invokes the virtual method `DtConnectionMgr::initParams()` to allow for a separate parameter setup phase of the connection initialization. Both of these calls are made in `DtConnectionMgr::init()`, which is also virtual.

The *DtConnectionMgr* functions `reliableConn()` and `udpConn()` return pointers to the two connections (`reliableConn()` could return NULL). The `internalConn()` function returns a pointer to whichever of these connections is currently being used to send RTI internal bookkeeping data. Although pointers to individual connections are available through these functions, RTI code should always send messages by calling the *DtConnectionMgr* sending functions rather than by directly using the individual connections. This gives *DtConnectionMgr* the opportunity to do queueing and buffering if asynchronous I/O is enabled.

There is no separate logical connection to the rtiexec. The rtiexec listens to the same data streams that remote LRCs do, and picks out the messages that it needs to process.

Figure 10-1 shows how the physical connections used by messages sent through the Connection Manager's logical connections vary depending on the setting for the `RTI_fomDataReliableWhenUsingRtiexec` and `RTI_internalMsgReliableWhenUsingRtiexec` parameters in the *rid.mtl* file.

17.7.3. The Asynchronous I/O Connection Manager

DtAsyncConnectionMgr is a subclass that supports asynchronous I/O. While the interface to the connections is through the *DtConnectionMgr*, it is the *DtAsyncConnectionMgr* class that is instantiated by the RTI. The *DtAsyncConnectionMgr* class supports asynchronous or polling I/O based on RID file parameters. You can override the Connection Manager (from either *DtConnectionMgr* or *DtAsyncConnectionMgr*) using the *DtConnectionMgr::setCreatorFunction()* member function.

The *DtAsyncConnectionMgr* class handles all asynchronous I/O decisions by overriding member functions in *DtConnectionManager* and calling up to *DtConnectionManager* if asynchronous I/O is disabled.

For example, the tick member function calls *processPendingMsg()* and then *processConnections()* as follows:

```
RTI::Boolean DtConnectionMgr::tick()
{
    DtBoolean pendingResult = processPendingMsg();
    DtBoolean processResult = processConnections();

    return (processResult || pendingResult) ? RTI::RTI_TRUE :
        RTI::RTI_FALSE;
}
```

The implementation of *DtConnectionMgr::processConnections()* processes the connections directly. *DtAsyncConnectionMgr* overrides the *processConnections()* member function. If asynchronous I/O is disabled, it calls up to *DtConnectionManager::processConnections()* and returns.

On the sending side, *DtConnectionMgr::send(...)* sends messages directly using the connections. The *DtAsyncConnectionMgr* overrides the *send()* method. The implementation of *DtAsyncConnectionMgr::send(...)* checks the asynchronous status. If asynchronous I/O is disabled, it calls up to *DtConnectionMgr::send(...)* and returns. If asynchronous is enabled, *DtAsyncConnectionMgr::send(...)* invokes *DtAsyncConnectionMgr::queueMsg(...)*.

Several member functions in *DtAsyncConnectionMgr* encapsulate the signaling between federate and I/O thread.

```
// Signal the receive event
// Used by I/O thread to signal federate thread
virtual void signalReceiveEvent();

// Clear receive event
// Used by the federate thread to clear signal
virtual void clearReceiveEvent();

// Signal the send event
// Used by the federate thread to signal I/O thread
virtual void signalSendEvent();
// Clear send event
// Used by the I/O thread to clear signal
virtual void clearSendEvent();
```

The `readFileDescriptor(RTI::TransportType transport)` and `readFileEvent(RTI::TransportType transport)` member functions generically support checking the status of pending messages.

17.7.4. Changing the RTI's Communications Infrastructure

One thing you might like to do with the RTIspy API is change how the RTI communicates, for example you might want to have the RTI encrypt and decrypt the messages sent over the network.

To change the RTI's communications infrastructure, you need to change the way the connections are set up and configured. You can approach this with some or all of the following:

- ♦ Create new subclasses of the *DtSocket* class that *DtRtiConnection* uses
- ♦ Create new subclasses of *DtRtiConnection*
- ♦ Instantiate existing *DtRtiConnections* with different parameters.

Whatever approach you take, you will probably need to subclass *DtConnectionMgr* and override some of the functions responsible for creating connections.

Subclassing DtSocket

A *DtRtiConnection* uses a *DtSocket* to send and receive messages. Depending on which constructor you use, it can create its own or you can pass one in. If you subclass *DtSocket*, pass it to the *DtRtiConnection*. The RTIspy API uses the VR-Link *DtSocket*.

To replace the MÄK RTI's default network-based communication infrastructure by changing the behavior at the *DtSocket* level, you would probably want to:

1. Derive a new kind of *DtSocket* that implements sending and receiving by writing and reading using your alternative communication method. (For complete information about creating a *DtSocket*, please see VR-Link documentation.)
2. Derive a new kind of *DtConnectionMgr* that uses a single *DtRtiConnection* that is initialized by passing in an instance of your new type of socket.

The Role of DtRtiConnection

A *DtRtiConnection* is a wrapper around a *DtSocket*. A *DtSocket* just sends and receives messages. A *DtRtiConnection* knows what kind of messages they are. It has functions for creating and removing callbacks that can execute code in response to particular message types. It also has accessor and mutators for federation execution and federate handles and the federation execution name. It is probably not necessary to subclass *DtRtiConnection*, because the best way to change networking details is to subclass *DtSocket*.

Overriding Connection Manager Functions

If you want to completely reimplement the way we create connections, ignoring what the RID file says, and so on, you can ignore most of the functions in *DtConnectionMgr*, and just override *init()*, *internalConn()*, *reliableConn()*, and *udpConn()*. Subclass *DtRtiConnection* to create the modified connection that you want to use. Then have *internalConn()*, *reliableConn()*, and *udpConn()* return pointers to instances of your new connections.

If you want the Connection Manager to generally work like the default version, but need to make some small changes, please see the header files (or class documentation) for information about the various functions.

Overriding *DtConnectionMgr::tick()*

The *tick()* function calls *readAndProcess()*, which checks for new messages. The default is to call *readAndProcess()* on each connection. You probably will not want to change this behavior. However, if you do not want to check a particular connection every tick, you can override *tick()* to change the behavior.

Example

The following example demonstrates methods for changing how the RTI communicates, as discussed in this section.

1. Derive a new *DtSocket* that does sending and receiving using whatever communications mechanism you choose:

```
class MySocket : public DtSocket
{
public:

    // Constructors
    ...
    // Override sendTo
    virtual int sendTo(caddr_t packet, size_t size,
        DtInetAddr addr = DtUSE_DEFAULT_IP_ADDR)
    {
        // Your implementation here. Return number of bytes sent, -1 for
        // failure.
    }

    // Override the two versions of DtRecv
    virtual int DtRecv(caddr_t* buffptr, int* status = NULL)
    {
        // Your implementation here. Set buffptr to point to a buffer that
        // contains the received data. Return number of bytes received, -1
        // for failure.
    }
    virtual int DtRecv(caddr_t buff, size_t size)
    {
        // Your implementation here. Copy received data into buff, up to
        // size bytes. Return number of bytes received, -1 for failure.
    }
};
```

2. Derive a new *DtConnectionMgr* that uses a *DtRtiConnection* configured to use an instance of your new kind of *DtSocket*:

```
class MyConnectMgr : public DtConnectionMgr
{
public:

    // Constructor
    MyConnectMgr(DtRIDParameters* params, DtFederateMgr* fedMgrPtr) :
        DtConnectionMgr(params, fedMgrPtr)
    {
    }

    // Destructor
    ~MyConnectMgr
    {
        // Since we create the socket and connection (in init), we should
        // delete them. Set myUdpConn, myTcpConn, and myInternalConn to NULL
        // so that the base destructor will not also try to delete them. In
        // this example, all three pointers point to the same connection, so
        // only delete it once.
        delete myUdpConn;
        myUdpConn = myReliableConn = myInternalConn = NULL;
    }

    virtual void init(DtRIDParameters* params)
    {
        // Instantiate your derived socket, and a DtRtiConnection that will
        // use it
        DtSocket* sock = new MySocket(...);
        DtRtiConnection* conn = new DtRtiConnection(sock);

        // In this example, we set up myUdpConn, myReliableConn, and
        // myInternalConn to point to this one connection. Alternatively,
        // you might not want to use a single physical connection for all
        // three logical connections.
        myUdpConn = myReliableConn = myInternalConn = conn;
    }

    // Creator function, which we will register with the RTI
    static DtConnectionMgr* create(DtRIDParameters* params,
        DtFederateMgr* fedMgrPtr)
    {
        return new MyConnectMgr(params, fedMgrPtr);
    }
};
```

3. Tell the RTI to use your derived connection manager, within *SetPluginCreators()*:

```
void SetPluginCreators()
{
    DtConnectionMgr::setCreatorFunction(MyConnectMgr::create);
}
```

17.7.5. Messages

Messages in the RTI are represented by instances of subclasses of *DtRtiMessage* (*rtiMsg.h*). Different subclasses represent different kinds of messages that an LRC needs to send to other LRCs or to the rtiexec. Examples include *DtJoinMsg* (*joinMsg.h*), which is sent by a federate as a result of a call to *joinFederateExecution()*, and *DtUpdateRoMsg* (*updateRoMsg.h*), which contains receive order attribute updates. For a full list of message kinds, please see *rtiMsg.h*.

You might want to use the RTIspy API to change the wire format for messages. This typically involves subclassing one or more of the *DtRtiMsgs* and changing the way the message is represented on the network. For example, to change the way attribute updates are encoded, subclass from *DtUpdateRoMsg*, *DtUpdateTsoMsg*, or both.

You can also create brand new types of messages, but these will only be useful if you are creating custom managers and are writing code to send and receive the new messages.

The various *DtRtiMsg* subclasses have accessor functions to set and inspect various fields. All messages share common header information. Accessors for header fields are inherited from the base *DtRtiMsg*. *DtRtiMsg* inherits from *DtBaseMsg*, which is taken from the *vlutil* library in the VR-Link API.

When a *DtRtiMsg* is sent through a *DtRtiConnection*, the connection calls *netRep()* on the message to get a buffer that contains exactly the bytes that need to be communicated to other LRCs or the rtiexec. It then passes that buffer to its *DtSocket* for sending. On the receiving side, when a connection gets a packet from its *DtSocket*, it constructs the appropriate kind of *DtRtiMsg* by passing that network representation buffer to the message's constructor.

When deriving your own kinds of *DtRtiMsgs*, you could just ignore the way we normally implement the accessors, *netRep()*, and the from-network-representation constructors, and provide brand new implementations for these functions. However, we suggest taking advantage of the infrastructure that the *DtRtiMsg* class provides for managing the buffer used to store the network representation, managing the message headers, and so on. The recommended method for subclassing *DtRtiMsg* very closely parallels the method of deriving new kinds of *DtPdus* in VR-Link. For examples, please see the VR-Link documentation.

Once you have created new subclasses of *DtRtiMsg*, you must instruct the RTI to use them instead of the default versions. Do this by properly configuring the RTI's message factory. A message factory, represented by the *DtRtiMsgFactory* class (*rtiMsgFact.h*), is a table that maps message types to creator functions that are used to instantiate the appropriate subclass of *DtRtiMsg*. When the RTI needs to instantiate a particular kind of message, it looks up the message kind in the *DtRtiMsgFactory*, finds the appropriate creator function, and uses it to create the right type of message. The *DtRtiMsgFactory* is obtained from the Connection Manager, using its *msgFactory()* member.

To configure the message factory so that it will contain your creator functions, use its *addSendCreator()*, and *addReceiveCreator()* member, passing the message kind, along with a creator function to be used for outgoing or incoming messages.

The following example assumes you are creating a subclass of *DtUpdateMsg* called *MyUpdateRoMsg*:

```
class MyUpdateRoMsg : public DtUpdateRoMsg
{
public:

    // Creator function for sending:
    static DtMsg* create(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER)
    {
        return new MyUpdateRoMsg(buffer);
    }

    // Creator function for receiving
    static DtMsg* create(const DtNetMessageHeader* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER)
    {
        return new MyUpdateRoMsg(initial, buffer);
    }

    ...
    // Constructors, and overridden member functions
    ...
};

void InitPlugin(DtRtiAmbassadorImplementor* implementor)
{
    DtRtiMsgFactory* fact = implementor->connectMgr()->msgFactory();
    fact->addSendCreator(DtUpdateRoMsgKind, MyUpdateRoMsg::create);
    fact->addReceiveCreator(DtUpdateRoMsgKind, MyUpdateRoMsg::create);
}
```

17.8. Responding to Dropped Federates

The rtiexec detects dropped TCP connections (via the RTI Forwarder *DtTcpForwarderThread*) and automatically resigns the federate. When the *DtTcpForwarderThread* creates a TCP connection to the LRC, it sends a *DtConnectMsg* to it point-to-point without a forward to any other LRC. The *DtConnectMsg* assigns a unique ID to the LRC (across all federations). The member function *DtExecFederateMgr::init()* expects to process the *DtConnectMsg* if a reliable connection was created. The LRC uses this value to replace myID. If it does not get this message, myID reverts to its original value (either a random number or process ID (*RTI_useRandomNumberForFedHandle*)). So, even though the message is not received, the LRC should still be able to function (without fault tolerance based on lost TCP connections). A failure message is displayed if the *DtConnectMsg* is not received. This message is only a warning. To get rid of the warning message, the LRC would have to be directly fed a *DtConnectMsg* when the *DtConnectMgr* was created.

A *DtDisconnectMsg* sent to the rtiexec with the LRC ID causes that federate to be resigned from its federation.

17.8.1. Responding to Missing Heartbeats

The MÄK RTI automatically resigns federates that fail to send heartbeat messages. You can disable this functionality in *rid.mtl*.

17.9. Finding Out when Data is Available to be Read

Some applications want to avoid excessive polling of the RTI – calling `tick()` repeatedly even when no data is available to be read. The MÄK RTI lets you find out when data is available to be read in an event-based fashion.

Federates can call the `DtReadFileDescriptor()` member function, declared in *makRti.h*, to obtain a file descriptor on which data will be present whenever there is incoming data waiting to be processed by the RTI:

```
int DtReadFileDescriptor(RTI::RTIambassador* amb, RTI::TransportType
    transport);
```

A separate file descriptor is used to signal the presence of reliable versus best effort data. The file descriptor can be used in calls like `select()` to avoid unnecessary polling.

On Windows, this function works only if you are not using asynchronous mode. If you are using asynchronous mode, `DtReadFileDescriptor()` returns -1, and you must use an alternate method to avoid polling.

The `DtReadFileEvent()` member function (also declared in *makRti.h*), returns a WIN32 event handle that signals when data is available to be read:

```
WSAEVENT DtReadFileEvent(RTI::RTIambassador* amb, RTI::TransportType
    transport);
```

This function can be used in calls such as `WaitForSingleObject` to avoid polling.



These functions are not part of the standard RTI API. If you rely on these functions, you will not be able to switch to other RTI implementations without recompiling (and avoiding these functions).

17.10. rtiexec Plug-ins

In addition to customizing the behavior of the LRCs, you may need to customize the rtiexec. For example, if you develop a plug-in that changes the wire format of the RTI or alters the communication mechanism used, you will also want to have the plug-in affect the rtiexec, so that it knows how to send, receive, or interpret according to the new scheme. Another reason to build an rtiexec plug-in is to monitor the federation. For example, you might want to have some code execute whenever federates join or leave the federation execution.

As previously discussed in [“Initializing a Plug-in,”](#) on page 17-4, you initialize an rtiexec plug-in with `InitRtiexecPlugin()`. The object passed to `InitRtiexecPlugin()` is a *DtRtiExec* pointer. *DtRtiExec* (*exec.h*) is the top-level object that implements the rtiexec's functionality.

The *DtRtiExec* processes create and destroy messages from LRCs and maintain a list of *DtFedExec* objects, one for each active federation execution. You can get a pointer to this list using `DtRtiExec::executionList()`. *DtFedExec* (*fedex.h*) manages joins, resigns, and synchronization points for a particular federation execution. It manages a list of *DtFederate* objects (*federate.h*), one per federate, that is joined to the federation execution. You can get a pointer to this list using `DtFedExec::federates()`. You can use member functions on *DtRtiExec*, *DtFedExec*, and *DtFederate* to query these objects for information about current state. The classes also support callback mechanisms that allow a plug-in to be notified when federation executions are created or destroyed, when federates join or resign, and so on. For details, please see the appropriate header files.

The rtiexec uses an instance of *DtConnectionMgr* to manage its connections to the various LRCs. This is the same class used by the LRCs themselves. To change the communication mechanism used by the rtiexec, follow the same process you would use for the LRCs. That is, subclass *DtConnectionMgr* with a version that configures your *DtRtiConnections* appropriately, and then make sure that your version of *DtConnectionMgr* gets used by registering a creator function for it within your plug-in's `SetPluginCreators()` implementation.

To customize the behavior of the rtiexec, you might want to subclass *DtRtiExec*, *DtFedExec*, or *DtFederate*, and override some of its virtual functions. Of course you will need to tell the rtiexec to use instances of your subclasses. To install your subclasses, call `DtRtiExec::setCreatorFunction()`, `DtFedExec::setCreatorFunction()`, or `DtFederate::setCreatorFunction()` from within your `SetPluginCreators()` function. Pass to these calls a static function that returns a new'd instance of your subclass. The procedure is similar to creating custom managers, described in [“Creating Custom RTI Managers,”](#) on page 17-9.

17.11. Accessing RID File Parameters

RID file parameters are made available to RTI code and to plug-ins through an instance of the *DtRIDParameters* class (*RIDparams.h*). When a federate instantiates an `RTI::RTIambassador`, one of the first things that `RTI::RTIambassador` does is to create an instance of *DtRIDParameters* and fill in its member data by reading the RID file. The *DtRIDParameters* object is passed to each of the RTI's managers as it is constructed, so that the parameters can be used during construction. If you derive your own subclasses from one or more managers, you can access the RID parameters by calling the appropriate inspector functions on *DtRIDParameters*.

You can also add your own custom RID parameters, so that you can configure some of the custom functionality that you have added to the RTI in your plug-in. The following example assumes that you want to add an integer parameter called **MyParam** to the RID file. RID file writers would add a line like this to the RID file:

```
(setq MyParam 10)
```



The RID parameters implemented by MÄK begin with `setqb`. User-defined parameters should begin with `setq`. For details, please see “[MÄK Technologies Lisp \(MTL\) Files](#),” on page A-2.

From within your plug-in code, you can obtain the value of **MyParam** as follows, assuming that `params` is a *DtRIDParameters* object that has been passed to one of your derived managers:

```
int val;
DtObjPtr value;
value = params->mtlEnvironment().lookup("MyParam");
if(!value.null())
{
    val = value->fixnum();
}
```

If your parameter value is a string, use `string()` instead of `fixnum()`. For floats, use `flonum()`.

This technique relies on the MTL library, a utility library that is actually part of VR-Link. You must include *mtlInc.h* and link with *libmtl* to use *DtObjPtr*.

You do not need to use *rid.mtl* and its accompanying lisp-like interpretation system to configure your plug-ins. Custom configuration mechanisms will not harm the RTI.

17.12. Compiling and Linking

This section explains how to compile and link a plug-in created with the RTIsPy plug-in API.

There is a single RTIsPy API for both the three RTI implementations. The precompile directive controls the compilation of code as either 1.3 (DtIFSPEC13), 1516 (DtIFSPEC1516), or HLA Evolved (DtIFSPEC1516e). You must use the definition appropriate to the specification you are compiling for. When a plug-in is developed and compiled, it should adhere to the types and components of the respective implementation. The HTML class documents are generated from preprocessed files to present the classes as they are compiled for the respective implementations.

17.12.1. Compiling and Linking on UNIX

When compiling source files that make up your plug-in DLL, you must compile against the RTIsPy header files in the *./plugins/include* directory, as follows:

```
-I rtiDir/include
```

where *rtiDir* is the path to the root of your MÄK RTI installation.

When you link your shared library, there is usually no need to link with any MÄK libraries. Your plug-in will find the symbols it needs in the RTI libraries at run-time.

You must link against the RTI libraries, including the specialized RTI versions of the VR-Link utility libraries. Include the following object and library modules in your project settings:

- ♦ *libRTI-NG.lib*
- ♦ *rtimatrixRT.lib*
- ♦ *rtimtlRT.lib*
- ♦ *rtivlutilRT.lib*.

Provide a path to the MÄK RTI *./lib* directory in your additional library path.

Example Makefiles are in the *./plugins/examples/printer* directory.



The RTIsPy 1516 plug-ins must be linked against *librti1516.lib*. The RTIsPy plug-ins for HLA Evolved must be linked against *librti1516e.lib*.

17.12.2. Compiling and Linking on Windows

Under Windows, you must set the paths to the RTIsPy *include* directory (*./plugins/include*) in your project settings.

When you link your shared library, you must link against the RTI libraries, including the specialized RTI versions of the VR-Link utility libraries. Include the following object and library modules in your project settings:

- ♦ *libRTI-NG.lib*
- ♦ *rtimatrixRT.lib*
- ♦ *rtimtlRT.lib*
- ♦ *rtivlutilRT.lib*.

To build debug versions of your DLL, append a *d* to the library name, for example, *rtimatrixRTd.lib*. Provide a path to the MÄK RTI *./lib* directory in your additional library path.



The RTIsPy 1516 plug-ins must be linked against *librti1516.lib*. The RTIsPy plug-ins for HLA Evolved must be linked against *librti1516e.lib*.

Example project files are in the *./plugins/examples/printer* directory.

17.13. Loading Plug-Ins

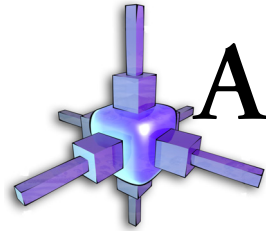
After you build a plug-in, you must tell the RTI to load it. To load a plug-in, add the following line to *rid.mtl*:

```
(RTI-addPluginName "MyPlugin.dll")
```

where *MyPlugin.dll* is the full path to your plug-in library.

You can load multiple plug-ins by adding additional **RTI-addPluginName** lines in *rid.mtl*. It is your responsibility to make sure that the plug-ins are compatible.

Remember that in the MÄK RTI, all LRCs, as well as the *rtiexec*, load their own RID files, so make sure you configure all copies of *rid.mtl* appropriately.



RID File Parameters Reference

This appendix describes the parameters in the *rid.mtl* file.

MÄK Technologies Lisp (MTL) Files.....	A-2
Using Environment Variables in an MTL File	A-2
Using Conditional Statements.....	A-3
RID File Parameters.....	A-4
Federation-Wide Parameters	A-4
LRC-Specific Parameters.....	A-14
Configuring tick(min,max)	A-27

A.1. MÄK Technologies Lisp (MTL) Files

Many of the configuration files for VT MÄK products are ASCII text files that use MÄK Technologies Lisp (MTL) to encode configuration information. MTL files have the extension *.mtl*. You can edit them in any text editor. Be sure that your text editor uses the proper end-of-line characters for your platform.

Most parameter entries take the format:

```
(setq parameter_name value)
```

For example:

```
(setq EnableWireframe 1)
```

or:

```
(setqb parameter_name value)
```

The `setqb` syntax indicates a bound variable.

To comment out a line, precede the text with two semi-colons (`::`). You can also add a comment at the end of a line by preceding it with two semi-colons.

In most cases, the parameter names are known to the MÄK application and the MTL commands simply assign values to them. However, you can use the `setq` command to create arbitrary symbolic names or aliases that are then used in later commands. For example, in the following two commands the symbolic name `plugin-directory-root` is assigned to a directory path. Then the symbolic name is used in the `if` command. The name `plugin-directory-root` has no intrinsic meaning to the application. It is given meaning by the `setq` command, which allows it to be used later in the `if` command.

```
(setq plugin-directory-root "../plugins/B-HAVE/")  
(if (equal Transport "DIS") (setq plugin-directory (string-append plugin-  
    directory-root "binDIS/")))
```

In addition to the `setq` command, a commonly used command is the `load` command. This command instructs the application to load the file specified, for example:

```
(load mtl-path "config.mtl")
```

A.1.1. Using Environment Variables in an MTL File

If you want to use an environment variable in an MTL file, use the following syntax:

```
(setqb parameter_name (getenv (quote env_var)))
```

For example:

```
(setqb disDestAddr (getenv (quote DEST_ADDR)))
```

A.1.2. Using Conditional Statements

MTL commands can be conditional. Conditional statements are used frequently when parameters are different for DIS and HLA. A simple conditional statement has the format:

```
(if (condition) (statement_to_evaluate))
```

If you have more than one statement to evaluate, use a sequence block:

```
;; HLA specific parameters
(if (equal HLA 1)
  (sequence
    (setqb fomMapperLib "")
    (setqb fomMapperInitData "")
    (setqb rprFomVersion 1.0)
    (setqb pathToSublistFile "sublist.mtl")
    (setqb ignoreAdvisories 0)
    (setqb fedLookahead 1.0)
  )
)
```

A.2. RID File Parameters

The MÄK RTI RID file uses two formats for parameters. Most parameters are in the following format:

```
(setqb RTI_parameterName value)
```

This format behaves like an assignment in which the value is assigned to *RTI_parameterName*. Repeated occurrences of an assignment parameter result in its taking the value from the last occurrence. Assignment RID parameters begin with **RTI_**.

Some RID parameters behave more like a function, in which the function is applied to the value. Repeated occurrences of these function parameters usually result in the values from each occurrence being accumulated. The function parameters begin with **RTI-** and they omit the **setqb** keyword as in the following example:

```
(RTI-parameterName value)
```

A default *rid.mtl* file is in the MÄK RTI root directory.

A.3. Federation-Wide Parameters

[Table A-1](#) lists and briefly describes parameters in *rid.mtl* that must be the same for every federate in a federation. The table is organized to match the *rid.mtl* organization. If applicable, the description includes a cross reference to further information in the manual.

Table A-1: Exercise-wide RID file parameters (Sheet 1 of 11)

Parameter	Description
RTI Compliance	
RTI_forceFullCompliance	Specifies all relevant RID file parameters to support full compliance with the HLA Interface Specification. 0 — Do not force full compliance 1 — Force full compliance. Default: 0. For more information, please see “Enabling and Disabling HLA Services,” on page 12-2. This parameter is ignored unless RTI_configureConnectionWithRid is set to 1.

Table A-1: Exercise-wide RID file parameters (Sheet 2 of 11)

Parameter	Description
Communications Configuration	
RTI_useRtiExec	Determines whether the application should try to connect to an rtiexec application when it creates, destroys, or joins a federation execution. 0 — Do not use the rtiexec 1 — Use the rtiexec. Default: 0. This parameter is ignored unless RTI_configureConnectionWithRid is set to 1.
RTI_udpPort	Specifies the port number to be used for UDP and TCP messages. Default: 4000. This parameter is ignored unless RTI_configureConnectionWithRid is set to 1.
RTI_destAddrString	Specifies the IP address to which best effort federation execution traffic is sent. This will typically be a broadcast address or multicast address. Default: "229.7.7.7" (a multicast address). A value of "0.0.0.0" is a special value that indicates that the default broadcast address should be used. If any of your federates are running on platforms on which multicast is not supported, you must use a broadcast address rather than a multicast address. For more information, please see "Configuring Best Effort (UDP) on a LAN," on page 11-3. This parameter is ignored unless RTI_configureConnectionWithRid is set to 1.
RTI_tcpForwarderAddr	Specifies the IP address of the machine that the rtiexec application is running on. This value is required if you are using reliable transport. Default: "127.0.0.1" For more information, please see "Configuring UDP and Centralized TCP Forwarding on a LAN," on page 11-3 and "Configuring UDP and Centralized TCP on a WAN," on page 11-4. This parameter is ignored unless RTI_configureConnectionWithRid is set to 1.

Table A-1: Exercise-wide RID file parameters (Sheet 3 of 11)

Parameter	Description
RTI_internalMsgReliableWhenUsingRtiexec	Specifies whether or not you want to send the RTI's internal bookkeeping messages using reliable transport. This parameter is ignored if RTI_useRtiExec is set to 0. 0 — Use best effort for all internal messages. 1 — Use reliable for all internal messages. Default: 1. For more information, please see "Choosing Best Effort or Reliable Transport," on page 10-2. This parameter is set to 1 unless RTI_configureConnectionWithRid is set to 1.
RTI_fomDataReliableWhenUsingRtiexec	Specifies whether or not you want to send object attributes and interactions using reliable transport, when so stated in your FED file. This parameter is ignored if RTI_useRtiExec is set to 0. 0 — Use best effort for all FOM data. 1 — Use reliable or best effort as indicated in the FED file. Default: 1. For more information, please see "Choosing Best Effort or Reliable Transport," on page 10-2. This parameter is set to 1 unless RTI_configureConnectionWithRid is set to 1.
RTI_forceFomDataReliable	Specifies transmission of FOM data via the reliable transport channel. Default: 0. For more information, please see "Starting the Shared Memory Manager," on page 14-5. This parameter is set to 0 unless RTI_configureConnectionWithRid is set to 1.
Optional Services	
RTI_momServiceAvailable	Enables (1) or disables (0) management object model (MOM) services. Default: 0. For more information, please see "Configuring MOM Services," on page 12-2.
RTI_timeMgmt	Enables (1) or disables (0) time management services. Default: 0. For more information, please see "Configuring Time Management Services," on page 12-2.
RTI_extend13and1516interop	Extends interoperability between 1.3 and the 1516 versions (for example, handling DDM dimensions). For more information, please see "RTI Interoperability," on page 1-8.

Table A-1: Exercise-wide RID file parameters (Sheet 4 of 11)

Parameter	Description
RTI_dataDistMgmt	Enables (1) or disables (0) data distribution management (DDM) services. Default: 0. For more information, please see “Configuring DDM Services,” on page 13-2.
RTI_ddmFixedGrid	Enables (1) or disables (0) fixed grid DDM versus distributed region DDM when DDM is enabled. Default: 0. For more information, please see “Configuring a Fixed Grid DDM,” on page 13-3.
RTI_ddmFixedGridFilename	Specifies the fixed grid configuration file name. For more information, please see “Fixed Grid Search Order,” on page 13-4.
RTIspy Web GUI	For more information about all parameters in this section, please see “Configuring the RTIspy,” on page 6-2.
RTI_webserviceHttpPort	Specifies the HTTP Service port for web client connections to the HTTP Server. Default: 8008.
RTI_webserviceRtiPort	Specifies the port used by RTI components to connect to the HTTP Server back-end. This parameter would be the same for multiple federates, but does not have to be the same for all federates. Default: 6002.
RTI_webserviceAddr	Specifies the IP address of the HTTP server for this RTI component to connect to. This parameter would be the same for multiple federates, but does not have to be the same for all federates.
Service Implementation	
RTI_responseInterval	Specifies the amount of time (in seconds) that the local RTI component waits to receive a response from the rtiexec when processing the createFederation, destroyFederation, or joinFederation service calls. Default: 3 seconds. For more information, please see “Specifying the Response Interval,” on page 7-20.

Table A-1: Exercise-wide RID file parameters (Sheet 5 of 11)

Parameter	Description
RTI_useRandomNumberForFedHandle	Instructs the RTI to create federate IDs based on random numbers rather than the federate's process ID, when not using the rtiexec. 0 — Use process ID. 1 — Use pseudo random number. Default: 0.
RTI_enableHlaObjectNamePrefix	Specifies that the RTI, and only the RTI, will create object names that begin with "hla". 0 — Do not force object names to start with "hla". 1 — Force object names to start with "hla". Default: 0.
RTI_sendDiscoveredClass	Enables (1) or disables (0) transmission of discovered class advisory messages. Default: 0. To meet the interface specification requirements for the turn updates on or off advisories, the discovery class of remote objects must be relayed to federates that own attributes of those objects. The transmission of the discover messages can be disabled, reducing the overhead with only a slight degradation of the advisory functionality (the actual class is used instead of the discovered class).
RTI_enableFomBackwardsCompatibility	Enables (1) or disables (0) use of the handle numbering scheme used in MÄK RTI 2.3.3 and earlier. It supports federates that recorded old handles in a file. Default: 0.
RTI_variableLengthDataUsesNull	Determines how VariableLengthData represents zero length data. By default, when an instance of the 1516 (or HLA Evolved) VariableLengthData class is empty (that is, length is zero), its data method returns a pointer to a single byte initialized to zero. To have the data method of an empty VariableLengthData instance return a NULL pointer, enable RTI_variableLengthDataUsesNull.
RTI_reuseReleasedObjectHandles	Allows the RTI to reuse object instance handles if the original instance has been deleted. Use of this feature breaks the guarantee that handles are unique over the lifetime of a federation. Handles will still be unique at any given instant.
RTI_distributeFedFile	Enables (1) or disables (0) distribution of the FED or FDD file. Default: 0. The federation must also enable RTI_useRtiExec and RTI_internalMsgReliableWhenUsingRtiexec. For more information, please see "Distributing the FED File," on page 7-4.

Table A-1: Exercise-wide RID file parameters (Sheet 6 of 11)

Parameter	Description
RTI_enableNameReservation	Enables (1) or disables (0) the name reservation service in HLA 1516 or HLA Evolved. Default: 1.
RTI_strictNameReservation	Enables (1) or disables (0) use of strict 1516 name reservation. If enabled, it ensures that no more than one object can reserve a name for the lifetime of the federation. Default: 0.
Fault Tolerance	
RTI_deleteOrphans	Specifies whether or not the objects for which a terminated federate owns the privilegeToDelete will be automatically deleted. 0 — disabled (all owned attributes are orphaned). 1 — enabled. Default: 1. For more information, please see “Handling Orphaned Objects,” on page 7-19.
RTI_enableFederateHeartbeat	Enables (1) or disables (0) the requirement that federates send a heartbeat.
RTI_federateHeartbeatInterval	Specifies the frequency, in seconds, with which each federate LRC sends a heartbeat message. Default: 10. Valid only if RTI_enableFederateHeartbeat is set to 1. For more information, please see “Responding to Abnormal Termination,” on page 7-18.
RTI_federateTimeoutInterval	Specifies the time period in which the federation must receive a heartbeat or it will forcibly remove the federate. Default: 25. Valid only if RTI_enableFederateHeartbeat is set to 1. For more information, please see “Responding to Abnormal Termination,” on page 7-18.
RTI_reconnectEnabled	Specifies that the LRC try to reconnect to the RTI Forwarder RTI_federateReconnectPause seconds after a TCP connection is lost. Default: 0. For more information, please see “Reconnecting to the RTI Forwarder,” on page 7-20.
RTI_federateReconnectPause	If RTI_reconnectEnabled is enabled, specifies the amount of time, in seconds, to wait before trying to reconnect to the RTI Forwarder. Default: 5. “Reconnecting to the RTI Forwarder,” on page 7-20.

Table A-1: Exercise-wide RID file parameters (Sheet 7 of 11)

Parameter	Description
RTI_rtiExecReconnectPause	If RTI_reconnectEnabled is enabled, specifies the amount of time, in seconds, to wait before trying to reconnect to the RTI Forwarder. Default: 4. For more information, please see “Reconnecting to the RTI Forwarder,” on page 7-20.
Advanced Network Configuration	
RTI_use32BitsForValueSize	Enables (1) or disables (0) support for the size of attributes and parameters to exceed 64 KB. The default message format uses 16-bit integers (max 64 KB) to represent the size of elements in handle value pair sets. Default: 0. For more information, please see “Enabling Support for Large Attributes and Parameters,” on page 10-7.
RTI_enableTcpCompression	Enables compression of RTI message payloads transmitted over TCP connections. Consistency checking is enforced on this parameter. For more information, please see “Compressing RTI Messages,” on page 10-6.
RTI_tcpCompressionLevel	Specifies the amount of compression applied to RTI message payloads over TCP connections as follows: 1 — minimal compression. 2 through 8 — increasing levels of compression. 9 — maximal compression. 10 — maximal compression of packet bundles (super bundling). This is only valid when packet bundling is enabled. For more information, please see “Compressing RTI Messages,” on page 10-6.
RTI_enableUdpCompression	Enables compression of RTI message payloads transmitted in UDP datagrams. Consistency checking is enforced on this parameter. For more information, please see “Compressing RTI Messages,” on page 10-6.
RTI_udpCompressionLevel	Specifies the amount of compression applied to RTI message payloads in UDP datagrams, using the same options as RTI_tcpCompressionLevel. For more information, please see “Compressing RTI Messages,” on page 10-6.

Table A-1: Exercise-wide RID file parameters (Sheet 8 of 11)

Parameter	Description
RTI_enablePacketBundling	Bundles messages together into a single TCP packet or UDP datagram, up to the size limit specified. The size value should be $\leq$ MTU of the network. If <i>size</i> is 0 (zero), then packet bundling is disabled. Default: 0. For more information, please see “Packet Bundling for Reliable and Best Effort Messages,” on page 10-3.
DM Class Multicast Filtering	
RTI-addDMObjectMulticastAddr	Allows you to assign object classes to separate multicast groups. Default: disabled. For more information, please see “Filtering by Multicast Groups Using Declaration Management,” on page 10-9.
RTI-addDMInteractionMulticastAddr	Allows you to assign interaction classes to separate multicast groups. Default: disabled. For more information, please see “Filtering by Multicast Groups Using Declaration Management,” on page 10-9.
RTI Forwarder Parameters	
RTI-addForwarderGroupAddrString	Specifies the IP address of a member of the federate’s local Forwarder Group. A separate entry for each RTI Forwarder in the group is necessary. Required only when forwarder groups are used. For more information, please see “Configuring RTI Forwarders for Distributed Forwarding,” on page 11-16
RTI_distributedUdpForwarderMode	Determines whether or not distributed forwarders handle UDP forwarding, and if so, the mode to use. Consistency checking is enforced on this parameter. The mode can be: 0 — Distributed forwarders do not handle UDP forwarding. 1 — Distributed forwarders imitate UDP Forwarders (Legacy Mode). 2 — Distributed forwarders handle combined UDP/TCP forwarding duties. For more information, please see “Configuring Distributed UDP Forwarding,” on page 11-24.

Table A-1: Exercise-wide RID file parameters (Sheet 9 of 11)

Parameter	Description
RTI_forwarderRoutesFile	Specifies the RTI Forwarder routes file to use. Default: "routes.mtl". ! You cannot use this parameter with a centralized RTI Forwarder. For more information, please see "The RTI Forwarder Application," on page 5-10.
RTI_distributedForwarderPort	Specifies the port for the distributed forwarder. Default: 5000.
RTI_smartForwardingLevel	Specifies the sender-side filtering mode, where: 0 — no subscription-based routing or filtering. 1 — enable subscription based routing. 2 — enable subscription based routing and disable send for updates and interactions that none of the federates have subscribed to. Default: 0. For more information, please see "Configuring Sender-Side Filtering (Smart TCP Forwarding)," on page 10-4.
RTI_maxNumFederates	Specifies the maximum number of federates allowed in a federation at any one time. This parameter only applies when sender-side filtering is enabled. Each update and interaction message has a bitmask of at least this number of bits appended, so increasing the number of federates increases the overhead. Default: 100. For more information, please see "Configuring Sender-Side Filtering (Smart TCP Forwarding)," on page 10-4.
RTI_forwardingDelay	When publication changes affect forwarding, routing filters are ignored for the specified interval to allow updates to remote subscription information. Default: 3.0.
RTI_routeToAllByDefault	When publication changes affect forwarding, route to all federates until specifically informed by remote federates that they have not discovered an object or are not subscribed to an interaction class. 0 — disable. 1 — enable. Default: 0.

Table A-1: Exercise-wide RID file parameters (Sheet 10 of 11)

Parameter	Description
RTI_discoveryMsgBundlingDelay	Delays the sending of discovery messages for the specified number of seconds. This allows discovery messages to be bundled together. Bundling can reduce traffic and processing time if a federate is creating a large number of objects at once. Default: 0.0 seconds.
RTI_enableFedexMsgRouting	This parameter makes sure that internal messages destined for the rtiexec are sent only to the rtiexec and that any messages (both internal and FOM) that the rtiexec does not process are not sent to the rtiexec. Default: 1. 0 — disable filtering. 1 — enable filtering. For more information, please see “Configuring Sender-Side Filtering for rtiexec Messages,” on page 10-6.
RTI_connectionTestInterval	Specifies the time to wait between connection tests for connections that may be idle for long periods.
Save/Restore Configuration	
RTI_enableSaveRestoreWhenUsingRtiexec	Enables (1) or disables (0) save/restore when you are using the rtiexec.
RTI_saveRestoreTimeout	Specifies the number of seconds the federation execution waits for a save/restore operation to complete before indicating a failure. Default: 120 seconds. For more information, please see “Configuring Save/Restore,” on page 7-22.
RTI_saveTransientMessages	Enables (1) or disables (0) saving of transient messages during a save, to be delivered upon restore. Default: 0. For more information, please see “Configuring Save/Restore,” on page 7-22.
Update Rate Reduction	
RTI_enableUpdateRateReduction	Enables (1) or disables (0) update rate reduction.
RTI-addUpdateRate	Registers an update rate with the LRC. You must specify a unique name and the update rate in hertz. For example, RTI-addUpdateRate("low", 0.1) registers an update rate called "low" with the value of 0.1 hertz.

Table A-1: Exercise-wide RID file parameters (Sheet 11 of 11)

Parameter	Description
RTI-addUpdateRateSubscription	Applies update rate reduction to a specified object class. Can be inclusive or exclusive. (This parameter is federate-specific.)
RTI_receiveTolerance	Specifies a percentage reduction to the inter-arrival time between attribute updates. It reduces the inter-arrival time for a given rate so that an update that appears to arrive early at the receiver is accepted.

A.4. LRC-Specific Parameters

Table A-2 lists and briefly describes parameters in *rid.mtl* that can be different for the members of a federation. The table is organized to match *rid.mtl* organization. If applicable, the description includes a cross reference to further information in the manual.

Table A-2: RID file parameters for LRCs (Sheet 1 of 14)

Parameter	Description
RTI_configureConnectionWithRid	Specifies that the rtiexec or federate use the connection configuration parameters in the RID file instead of those in the RTI Assistant connection configurations.
License Checking	
RTI_forceUnlicensedForTwo	Forces the MÄK RTI to use the unlicensed version without checking for a license. This setting saves time at startup. If you have a licensed version of the MÄK RTI, do not skip license checking. 0 — Check for a license. 1 — Do not check for a license. Default: 0.
RTI_disableUnlicensedForTwo	Enables (0) or disables (1) unlicensed mode in Windows (even if the forceUnlicensedForTwo parameter is set). When this parameter is enabled, federates must check out a license to run. Default: 0.
RTI_rtiExecPerformsLicensing	Enables (1) or disables (0) license checkout from the rtiexec. The rtiexec will check out a license for each joining federate. Default 0.

Table A-2: RID file parameters for LRCs (Sheet 2 of 14)

Parameter	Description
Multicast Discovery Configuration	
RTI_mcastDiscoveryEnabled	Enables (1) or disables (0) multicast discovery. Default: 0. For more information, please see “Automatically Locating the RTI Forwarder and rtiexec,” on page 10-3. This parameter is set to 0 unless RTI_configureConnectionWithRid is set to 1.
RTI_networkInterfaceAddr	Sets the address of the network device to be used for multicast traffic.
RTI_mcastDiscoveryPort	Specifies the port for multicast discovery. Default: 6001.
RTI_mcastDiscoveryAddrString	Specifies a multicast discovery address. Default: “229.7.7.8”. For more information, please see “Automatically Locating the RTI Forwarder and rtiexec,” on page 10-3.
RTI_lrcMcastDiscoveryDelay	Specifies the interval that the LRC waits for a multicast discovery response. Default: 3.0.
RTI_execMcastDiscoveryDelay	Specifies the interval that the rtiexec waits for a multicast discovery response. Default: 1.0.
RTI_mcastDiscoveryTries	Specifies the number of times the federate should try to find the multicast discovery address before it defaults to best effort transport. Default: 5. For more information, please see “Automatically Locating the RTI Forwarder and rtiexec,” on page 10-3.
Optional Services	
RTI_throwExceptionForDisabledService	Specifies whether or not function calls throw an exception when they call a service that is not activated in the MÄK RTI. 0 — Do not throw exceptions. 1 — Throw exceptions. Default: 1. For more information, please see “Choosing Silent Failure Instead of Exceptions,” on page 7-21.
RTI_enableWarningsForDisabled Services	Specifies whether or not the RTI Assistant displays a message when an exception is thrown for a disabled service. Default: 1 (on).
Switch Table Management	

Table A-2: RID file parameters for LRCs (Sheet 3 of 14)

Parameter	Description
RTI_preferRidToFedSwitchTable	Specifies which set of switch tables to use. 1 — Use the RID parameters. 0 — Use the switch tables defined in the FED file. Force full compliance forces use of the FED file switch table.
RTI_autoProvideSwitch	Specifies whether the Auto Provide switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
RTI_conveyRegionDesignatorSwitch	Specifies whether the Convey Region Designator switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
RTI_conveyProducingFederateSwitch	Specifies whether the Convey Producing Federate switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 1.
RTI_serviceReportingSwitch	Specifies whether the Service Reporting switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
RTI_exceptionReportingSwitch	Specifies whether the Exception Reporting switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
RTI_delaySubscriptionEvaluationSwitch	Specifies whether the Delay Subscription Evaluation switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
Advisory Configuration	For all parameters in this section, please see “Configuring Use of Advisory Messages,” on page 4-3 for more details.
RTI_interactionRelevanceAdvisorySwitch	Specifies whether the Interaction Relevance Advisory switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 1.

Table A-2: RID file parameters for LRCs (Sheet 4 of 14)

Parameter	Description
RTI_objectClassRelevanceAdvisorySwitch	Specifies whether the Object Class Relevance Advisory switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 1.
RTI_attributeRelevanceAdvisorySwitch	Specifies whether the Attribute Relevance Advisory switch is is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
RTI_attributeScopeAdvisorySwitch	Specifies whether the Attribute Scope Advisory switch is enabled (1), or disabled (0). RTI_preferRidToFedSwitchTable must be enabled to use this parameter. Otherwise it is overridden by the FED file switch table definitions. Default: 0.
Diagnostic Configuration	
RTI_notifyLevel	Specifies the category of notification messages to print. 0 — Fatal. 1 — Warn. 2 — Notify. 3 — Verbose. 4 — Debug. Default: 1.
RTI_logfileName	Enables logging to a file with the specified name. If just a filename is specified, a file is written for each federate on the computer in the directory from which a federate is executed. If a pathname is specified, data for all federates on the machine is written to the same file unless RTI_reuseLogFile is 0. Default: "makRti.log". For more information, please see "Enabling LRC Diagnostic Data Logging in the RID File," on page 8-11.
RTI_rtiExecLogFileName	Enables (1) or disables (0) the rtiexec log file and specifies the file's name. Default: "rtiExec.log". For more information, please see "Logging rtiexec Output," on page 5-3.
RTI_rtiForwarderLogFileName	Enables (1) or disables (0) the RTI Forwarder log file and specifies the file's name. Default: <i>rtiForwarder.log</i> .

Table A-2: RID file parameters for LRCs (Sheet 5 of 14)

Parameter	Description
RTI_reuseLogFile	Enables (1) or disables (0) writing logs to the file specified by RTI_logfileName (default: <i>makRti.log</i>). When enabled, it is possible that multiple federates might write to the same log file. If this option is disabled, the logs are written to <i>makRti###.log</i> where ### is replaced by the system time in seconds. Each federate will have a different log file. Default: 1. For more information, please see “Logging rtiexec Output,” on page 5-3.
RTI_dumpFed	Enables (1) or disables (0) dumping of the contents of the FED file when the notification level is set to verbose (> 3). Default: 0.
RTI_dumpRid	Enables (1) or disables (0) dumping of the contents of the RID file when the notification level is set to verbose (> 3). Default: 0.
RTI_detachNotifyLevelFromStdOut	Detaches all notification messages at the given level and above from the standard output. This setting has no affect on logging notifications to a file. Default: 5 (no notification levels detached).
RTI_ridConsistencyChecking	Enables (1) or disables (0) checking of the RID file for consistency with the rtiexec’s RID file. For more information, please see “The RID File Consistency Checker,” on page 4-9.
RTI-addRidParametersToOverride	Specifies a list of parameters to add to the default list of parameters that are checked for consistency. For more information, please see “Selecting the Parameters to be Checked,” on page 4-11.
RTI-removeRidParametersToOverride	Specifies a list of parameters to remove from the default list of parameters to check for consistency. For more information, please see “Selecting the Parameters to be Checked,” on page 4-11.
RTIsPy Configuration	
RTI_enableNetworkMonitoring	Enables (1) or disables (0) the RTIsPy Network Monitoring Tool. Default: 1. The RTIsPy web services plug-in must be loaded and enabled to enable network monitoring. For more information, please see “Enabling Network Monitoring,” on page 9-10.
RTI_enableNetworkTesting	Enables (1) or disables (0) network testing.

Table A-2: RID file parameters for LRCs (Sheet 6 of 14)

Parameter	Description
RTI_logNetworkMonitorStatistics	Enables (1) or disables (0) network statistic file logging for the RTIspy Network Monitoring Tool. Default: 0. The RTIspy Network Monitoring Tool must be enabled for this parameter to be relevant. For more information, please see “Logging Network Monitoring Statistics,” on page 9-10.
RTI_pluginDirectory	Specifies a directory relative to the RTI library path in which plug-in dynamic libraries are located. Default: <code>../plugins</code> .
RTI-addPluginName	Specifies the name of a plug-in that the RTI should load.
RTIspy GUI	For all parameters in this section, please see “Configuring the RTIspy,” on page 6-2.
RTI_enableRtiexecWebservice	Enables (1) or disables (0) the RTIspy at the rtiexec.
RTI_enableLrcWebservice	Enables (1) or disables (0) the RTIspy at this federate.
RTI_enableLrcWebserviceEventLog	Enables (1) or disables (0) the LRC web service event log for this federate.
RTI_webserviceRtiPort	Specifies the port used by RTI components to connect to the HTTP Server back-end. Default: 6002.
RTI_webserviceAddr	Specifies the IP address of the HTTP server for this RTI component to connect to.
RTI_webserviceEnableServerAutoStart	Attempts to start an HTTP server locally if one does not already exist (and RTI_webserviceAddr does not equal localhost : 127.0.0.1).
RTI_webserviceEnableForwarding	Allows the rtiexec to forward web client requests to federates running the web service plug-in. This allows the rtiexec to act as a proxy for the entire RTI network.
RTI_webserviceNotifyLevel	Specifies the notification level for messages to the web service event log.
Service Implementation	
RTI_singleCallbackPerTick	Specifies how many packets should be read for each tick. Also applies to tick(min,max). 0 — A call to RTI::tick with no arguments reads all pending packets. 1 — A single call to RTI::tick() should result in one packet being processed from the input stream. Default: 0. For more information, please see “Configuring tick(min,max),” on page A-27.

Table A-2: RID file parameters for LRCs (Sheet 7 of 14)

Parameter	Description
RTI_checkFlag	Enables (1) or disables (0) checking the validity of attributes and parameters and the ownership of attributes passed to updateAttributeValues() and sendInteraction(). Default: 1. Setting this flag to 0 results in a small performance gain, but it means that the AttributeNotDefined, AttributeNotOwned, and InteractionParameterNotDefined exceptions will never be thrown by these services.
RTI_processUnknownUpdatesForDiscovery	Enables (1) or disables (0) processing unknown objects for discovery. Default: 1. For more information, please see “Processing Discovery of Unknown Objects,” on page 7-21.
RTI_defaultFedFile	Specifies a default FED file in the RID file to support joining without calling create when operating in lightweight mode. In lightweight mode and using the default FED file, the join federation service always associates the federation name with the default FED file. If the value is an empty string, then the FED file will either be established by a previous create federation service call or federation name “.fed”. Default: “”. For more information, please see “Specifying the FED File,” on page 7-3.
RTI_crcCheckFedFile	If RTI_distributeFedFile is disabled (0), specifies whether or not a federate whose FED file has a different CRC than the rtirexec’s FED file can join the federation. 0 — Notify federate if the CRC of its FED file is different from the rtirexec’s FED file. Allow the federate to join. 1 — If the CRC of the federate’s FED file is different from the rtirexec’s FED file, do not allow the federate to join. Default: 0.
RTI_catchFedAmbExceptions	Configures the RTI so that exceptions thrown by the federate in a federate ambassador callback are caught and handled internally by the RTI. If not enabled, the RTI rethrows the exception to be handled by the federate by catching exceptions from tick(). The internal RTI state may be corrupted if this option is disabled. 0 — Do not catch exceptions. 1 — Catch exceptions. Default: 1.

Table A-2: RID file parameters for LRCs (Sheet 8 of 14)

Parameter	Description
RTI_strictFomChecking	Configures the RTI to strictly check the format of the FDD file (for example, to ensure presence of transport and order or disallow unknown tags). If enabled, an error in the FDD file causes the create federation service to fail. This parameter affects the HLA 1516 or HLA Evolved FDD files. 0 — Do not perform strict checking. 1 — Perform strict checking. Default: 0.
RTI_defaultTimeImplementation	Configures the RTI with the name of a default time implementation if an empty string is provided in the createFederationExecution service call. This default is used to override the default in the logical time library that is returned when the logical time name is an empty string. Default: "".
RTI_flushTimeoutInterval	Specifies the amount of time, in seconds, that an LRC will try to flush messages to the network when it is resigning from a federation. Default: 3.0.
Advanced Network Configuration	
RTI_mcastTtl	Specifies an integer value that gets used as the Time-ToLive parameter in the header of multicast packets. Default: 2.
RTI_tcpNoDelay	Allows you to configure the RTI to emphasize minimal latency or maximum throughput. 0 — Trade latency for throughput. 1 — Set the TCP_NODELAY socket option for TCP sockets, yielding minimum latency. Default: 1.
RTI_tcpBufferSize	Specifies the size of the buffer into which TCP packets get read. Default: 20000 bytes. For more information, please see "Choosing Best Effort or Reliable Transport," on page 10-2.
RTI_socketReceiveBufferSize	Specifies the size of the TCP and UDP network receive buffers. Increasing the size can increase the RTI's tolerance of peak network loading. Default: 50000. For more information, please see "Configuring the Network Buffer Sizes," on page 10-7.
RTI_socketSendBufferSize	Specifies the socket send buffer size. Default: 50000.
RTI_maxUdpPacketSize	Specifies the maximum size for a UDP packet. The default is 15000 for legacy purposes. However, packets this large are not recommended.

Table A-2: RID file parameters for LRCs (Sheet 9 of 14)

Parameter	Description
RTI_enableBestEffortSendRetry	Enables (1) or disables (0) automatic resending of failed attempts to send best effort messages. Configured with RTI_bestEffortSendRetryWaitUsec and RTI_bestEffortSendRetries. Default 0.
RTI_bestEffortSendRetries	If RTI_enableBestEffortSendRetry is enabled (1), RTI_bestEffortSendRetries configures the number of times a failed best effort send is retried. Default: 3. For more information, please see “Controlling Congestion for Best Effort Sockets,” on page 10-8.
RTI_bestEffortSendRetryWaitUsec	If RTI_enableBestEffortSendRetry is enabled, RTI_bestEffortSendRetryWaitUsec configures the time to wait between failed best effort sends, in microseconds. Default: 100.
RTI_useBusyWaitForTickMinMax	Specifies that the RTI will use busy wait in tick(min,max) when no messages are pending. 0 — Disabled (use select). 1 — Enabled. Default: 0. For more information, please see “Configuring tick(min,max),” on page A-27.
RTI_enableMessageThrottling	Enables (1), or disables (0) throttling of transmission and receipt of internal messages and the buffering of provideAttributeValueUpdate callbacks in HLA 1516 or HLA Evolved. Default 0 (disabled).
RTI_transmitDelay	If RTI_enableMessageThrottling is enabled this parameter configures the time to delay before processing requests. Default 0. For more information, please see “Configuring the Delay Time for Processing Internal Messages,” on page 12-3.
RTI_transmitRate	If RTI_enableMessageThrottling is set to 1, RTI_transmitRate specifies the number of messages per second that you want to send. Default 0.0 (disabled). For more information, please see “Transmitting Internal Messages Across Ticks,” on page 12-4.
RTI_autoProvideDelay	If RTI_enableMessageThrottling is set to 1, enables buffering of automatic provideAttributeValueUpdate callbacks in HLA 1516 or HLA Evolved to limit the number of provides requested if multiple federates require updates. Default: -1 (disabled). For more information, please see “Buffering Automatic Update Requests (1516 and Evolved),” on page 10-9.

Table A-2: RID file parameters for LRCs (Sheet 10 of 14)

Parameter	Description
RTI_dualTransmitFirstInteractionRegions	Enables (1) or disables (0) sending the DDM interaction region information both best effort and reliably the first time. A race condition between a best effort DDM interaction and its region information (sent reliably) is alleviated by transmitting the region information via best effort the first time. Default: 0.
RTI-addDestAddrString	Adds one or more additional destination addresses for best effort packets, which can be broadcast, unicast, or multicast addresses. All best effort data is sent to all registered addresses. Default: "0.0.0.0". For more information, please see "Configuring UDP Over a WAN without Distributed Forwarding," on page 11-6.
RTI_maxForwarderQueue	Specifies the size of the queue for messages that exceed the size of the receive buffer. When a federate does not tick fast enough and its receive buffer fills, it starts placing the messages into the queue. Once the queue has reached the maximum size, the federate is dropped. If the size is set to -1 (default), it queues messages indefinitely until the federate processes them all, or until all system memory is completely consumed.
RTI-addUpdateRateSubscription	Applies update rate reduction to a specified object class. Can be inclusive or exclusive.

Table A-2: RID file parameters for LRCs (Sheet 11 of 14)

Parameter	Description
Asynchronous Processing	
RTI_processingModel	Specifies the synchronous/asynchronous I/O operating mode, as follows: 0 — Synchronous mode. 1 — Asynchronous I/O. Perform network reads and writes in an asynchronous thread (reduces dependency on tick rate to avoid msg drops). 2 — Asynchronous Message Processing. In addition to performing network reads, the asynchronous thread processes messages. However, it will not perform any asynchronous federate ambassador callbacks. The callbacks will be cached and delivered when the federate calls tick or evokeCallbacks. 3 — Asynchronous Callbacks. Process messages and invoke callbacks in an asynchronous thread.
	i For Asynchronous Callbacks (not Asynchronous I/O) the Federate must be able to receive Federate Ambassador callbacks asynchronously (in a thread safe way).
	Default: 1.
RTI_IOPeriod	Specifies the amount of time polling operations take, in seconds. Not used in Windows, which is event driven. Must be greater than or equal to zero. Default: 0.01.
RTI_tickFavorsNetwork	Enables (1) or disables (0) a thread yield in tick(). Default: 1.
RTI_maxIOQueue	Specifies how many messages can be queued to send before blocking. Default 20000.
RTI_maxIOCount	Specifies the number of times the thread can cycle before it must yield. Must be greater than zero. Default: 1000.
RTI_IOLockQueue	Specifies the number of messages processed per locking of the queues. Adjustment of this value allows for fine or coarse locking. Must be greater than zero. Default: 500.

Table A-2: RID file parameters for LRCs (Sheet 12 of 14)

Parameter	Description
DDM Configuration	
RTI_minChannelAddr	Specifies the starting network multicast address for a pool of multicast groups used by DDM. The pool defined by this parameter and the RTI_maxChannelAddr parameter is used by any federate that loads this configuration file. Default: "224.0.0.1".
RTI_maxChannelAddr	Specifies the final network multicast address for a pool of multicast groups used by DDM. The pool defined by this parameter and the RTI_minChannelAddr parameter is used by any federate that loads this configuration file. Default: "239.255.255.255". For more information, please see "Configuring a Fixed Grid DDM," on page 13-3.
RTI_addressDelay	Specifies the delay by which data associated with changed regions and associations is broadcast (default multicast) while the DDM routing information is propagated to remote federates. Default: 1.0.
RTI_conveyOnlyAvailableDimensions	If RTI_conveyOnlyAvailableDimensions is enabled (1), the conveyed regions only include those dimensions that are available to the received interaction class. If the regions used to send the interaction include dimensions that are not available at the received interaction class, they are removed from the conveyed regions. If RTI_conveyOnlyAvailableDimensions is disabled (0), the conveyed regions include exactly those dimensions that were in the regions used to send the interaction. As a result, there may be dimensions that are not available at the received interaction class. Default: 0.
RTI_implicitDdmParamsFile	Specifies the name of the file that has implicit DDM configuration parameters. Default: <i>implicitDdmParams.mtl</i> .
MOM Configuration	
For details about all parameters in this section, please see "Configuring MOM Services," on page 12-2.	
RTI_momExceptionReports	Enables (1) or disables (0) MOM exception reports. Default: 0.
RTI_momExceptionLogging	Enables (1) or disables (0) MOM exception logging. Default: 0.
RTI_momFederateUpdateInterval	Specifies the default interval in seconds at which the MOM Federate object will be updated. Default: 0 (no updates sent).

Table A-2: RID file parameters for LRCs (Sheet 13 of 14)

Parameter	Description
RTI_momServiceReporting	Enables (1) or disables (0) service reporting, even if it is disabled in the FOM. Default: 0.
Save/Restore Configuration	
RTI_saveRestoreDirectory	Specifies the directory used for writing and reading save files. If the saved federation contains multiple federates of the same type, each federate of a given type must have access to all of the LRC save files of that type. The RTI determines the save files used for each federate. Default: ".".
Shared Memory	
RTI_sharedMemoryMode	Enables the RTI to use shared memory for communication between co-located federates and other RTI components (for example, the rtiexec). Valid settings are: 0 — disables shared memory (default). 1 — enables shared memory only (no network I/O). 2 — enable shared memory manager with network I/O.
RTI_sharedMemoryName	Specifies the name of a shared memory file. All co-located federates must use the same shared memory file name in order to communicate through the same shared memory region. Default: "RtiSharedMemory".
RTI_sharedMemoryQueueLength	Specifies the number of message queue segments (that is, 200 byte buckets) that can be queued in shared memory. Default: 5000.
RTI_sharedMemoryMgrMaxWait	Specifies how long (in seconds) the Shared Memory Queue Manager process waits for messages to arrive in the message queue before it polls the network interface for incoming messages. Default: 0.25.
RTI_sharedMemExecLogFileName	Enables logging shared memory executive output to a file with the specified name. Default: "shmExec.log"
Modular FOMs	
RTI_fomModuleMerging	Enables (1) or disables (0) merging of FOM modules. Default: 0.
RTI-addCreateFomModule	Specifies a FOM module to add when creating a federation.

Table A-2: RID file parameters for LRCs (Sheet 14 of 14)

Parameter	Description
RTI-addJoinFomModule	Specifies a FOM module file to add when joining a federation.
RTI_momModuleExtensionFileName	Specifies that the LRC will use the local FOM module file rather than having the rtiexec send it back.
RTI_preferLocalFomModule	Specifies a MOM module file to add when creating a federation.

A.5. Configuring *tick(min,max)*

By default, the implementation of *tick(min,max)* uses select (or windows events) to wait for additional input when there are no pending messages. The granularity of the select mechanism can make the *tick(min,max)* exceed the maximum time by too much (for example, select takes approximately 10 msec under Linux). To have finer control of when *tick(min,max)* returns, use the `RTI_useBusyWaitForTickMinMax` parameter to replace the select mechanism with a busy wait that polls the input source.

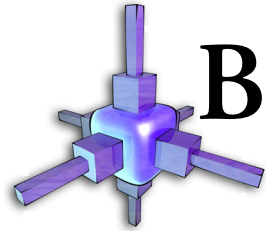


We do not recommend enabling the busy wait mechanism if you use asynchronous I/O. The busy wait will not yield to the I/O thread and the I/O thread must execute to generate input.

The `RTI_singleCallbackPerTick` parameter also applies to *tick(min,max)*. When `RTI_singleCallbackPerTick` is enabled, *tick(min,max)* returns after processing a single message or when the minimum time has been exceeded (effectively max is ignored).



A message may only cause internal processing and not necessarily generate a callback. The return value of *tick()* or *tick(min,max)* will be true if a message has been processed; otherwise, it will be false.



Example Federates

This appendix describes the example federates provided with the MÄK RTI.

Example Federates.....	B-2
The rtisimple Example.....	B-2
The simpleDDM Example.....	B-4
The simpletime Example	B-5
HLA Bounce.....	B-7

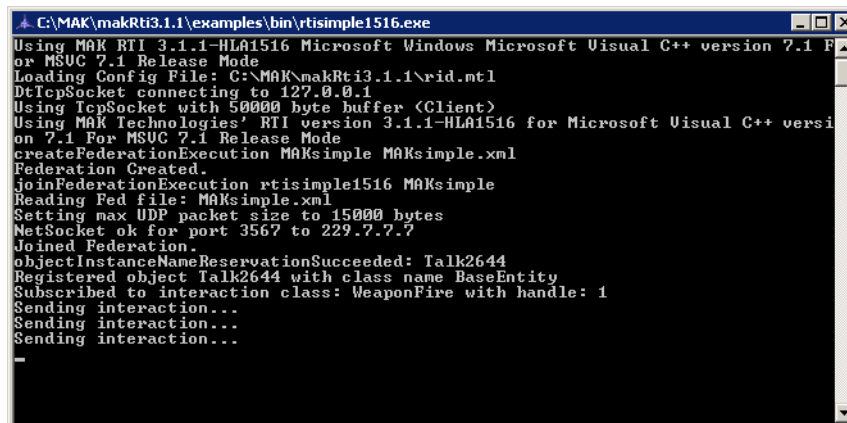
B.1. Example Federates

Explaining how to develop federates for use with HLA is beyond the scope of this manual. In general, when customers ask us how to develop federates, we recommend using the protocol-independent interface provided by VR-Link. For customers who just need a federate to do some basic exploration of HLA and the RTI, we recommend using the example applications provided with VR-Link. However, for those customers who want to develop directly to the HLA API, we include the simple examples described in this appendix. The examples are in the `./examples` directory. Compiled versions of each example are in `./examples/bin`. Each example has an HLA 1.3 version, an HLA 1516 version, and an HLA Evolved version.

This appendix provides brief introductions to the examples. For complete descriptions, please see the Examples link in the class documentation.

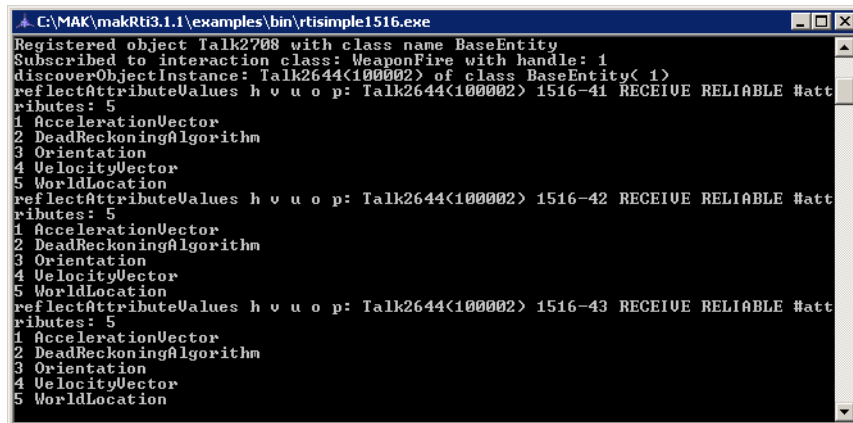
B.2. The *rtisimple* Example

The *rtisimple* example is a simple C++ example of how to use the RTI API. The *rtisimple* federate tries to create a federation and join it. It publishes and subscribes to the `BaseEntity` object class and registers an object of that class. Then it begins to update the object attributes and periodically sends an interaction. The attribute and parameter values contain the string names of the attributes and parameters. If two or more *rtisimple* federates are running, each federate discovers the other's objects, reflects the other's updated attributes, and receives the other's interactions as shown in [Figure B-1](#) and [Figure B-2](#).



```
C:\MAK\makRti3.1.1\examples\bin\rtisimple1516.exe
Using MAK RTI 3.1.1-HLA1516 Microsoft Windows Microsoft Visual C++ version 7.1 For MSUC 7.1 Release Mode
Loading Config File: C:\MAK\makRti3.1.1\rid.mtl
DtTcpSocket connecting to 127.0.0.1
Using TcpSocket with 50000 byte buffer <Client>
Using MAK Technologies' RTI version 3.1.1-HLA1516 for Microsoft Visual C++ version 7.1 For MSUC 7.1 Release Mode
createFederationExecution MAKsimple MAKsimple.xml
Federation Created.
joinFederationExecution rtisimple1516 MAKsimple
Reading Fed file: MAKsimple.xml
Setting max UDP packet size to 15000 bytes
NetSocket ok for port 3567 to 229.7.7.7
Joined Federation.
objectInstanceNameReservationSucceeded: Talk2644
Registered object Talk2644 with class name BaseEntity
Subscribed to interaction class: WeaponFire with handle: 1
Sending interaction...
Sending interaction...
Sending interaction...
=
```

Figure B-1. *rtisimple* console sending interaction



```

C:\MAK\makRti3.1.1\examples\bin\rtisimple1516.exe
Registered object Talk2708 with class name BaseEntity
Subscribed to interaction class: WeaponFire with handle: 1
discoverObjectInstance: Talk2644<100002> of class BaseEntity< 1>
reflectAttributeValues h v u o p: Talk2644<100002> 1516-41 RECEIVE RELIABLE #att
ributes: 5
1 AccelerationVector
2 DeadReckoningAlgorithm
3 Orientation
4 VelocityVector
5 WorldLocation
reflectAttributeValues h v u o p: Talk2644<100002> 1516-42 RECEIVE RELIABLE #att
ributes: 5
1 AccelerationVector
2 DeadReckoningAlgorithm
3 Orientation
4 VelocityVector
5 WorldLocation
reflectAttributeValues h v u o p: Talk2644<100002> 1516-43 RECEIVE RELIABLE #att
ributes: 5
1 AccelerationVector
2 DeadReckoningAlgorithm
3 Orientation
4 VelocityVector
5 WorldLocation

```

Figure B-2. rtisimple console receiving interaction

The rtisimple federates for HLA 1.3, 1516, and HLA Evolved can interoperate using the MÄK RTI. They all use a narrow string format for the attribute values and tags. The default federation file format matches the respective federate (that is, rtisimple13 uses *MAKsimple.fed*; rtisimple1516 and rtisimple1516e use *MAKsimple.xml*). The MÄK RTI supports both formats for the different APIs. However, the MOM specification is different in the between 1.3 and 1516. If the MOM is enabled in the RID file, then the appropriate federation file must be used.

B.3. The simpleDDM Example

The simpleDDM example is an adaptation of the rtisimple example program that incorporates DDM. (For a definition of DDM, please see Section 3.2.6, “[Data Distribution Management](#)”, in *MÄK RTI Users Guide*. For details about how the MÄK RTI implements DDM, please see Chapter 13, *Using Data Distribution Management*.) It performs the same typical federation calls as rtisimple (join, create, and so on) with the exception that when an object is registered, it is done so “with Region.”

The example has an object attribute and an interaction that is published and subscribed with region. To see the effects and features of the example, you must create two federates. One federate must be a publisher (`-isPublisher 1`) and the other a subscriber (`-isPublisher 0`). The regions in the example are conceptualized in one dimension as seconds along a 12 hour axis (ranging from 0 seconds to 43200) and in another dimension as a global time zone.

The published object represents the current time of the publisher and the published interaction represents an hourly chime. One chime (quarterlyChime) occurs at 12, 3, 6 and 9 o'clock; another chime occurs at the other hours. The timeZone dimension ranges from 0 to 23. It represents the different time zones around the world. It can only be changed in the publisher on the command line (`-timeZone time_zone`). The subscriber GUI lets you change it dynamically.

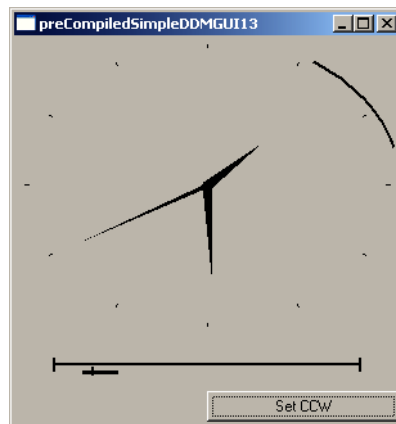


Figure B-3. simpleDDM publishing federate

The GUI displays clock hands, which for the publishing federate show the current time (Figure B-3), and for the subscribing federate show the last reflected position of the publisher (Figure B-4). (The subscriber's hands are not displayed prior to object discovery.) The GUI displays (as an arced segment around the dial) the region in the seconds dimension. It also displays a horizontal timeline at the bottom of the dial representing the 12 hour extent, a horizontal line representing the subscribed region, and the current position or last reflected position.

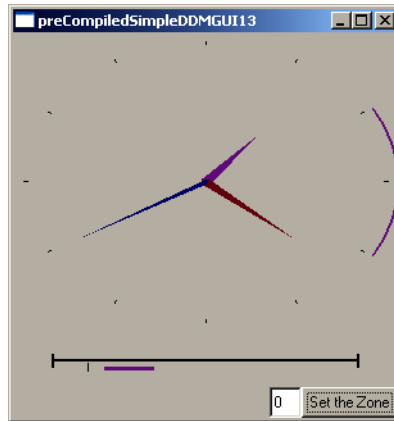


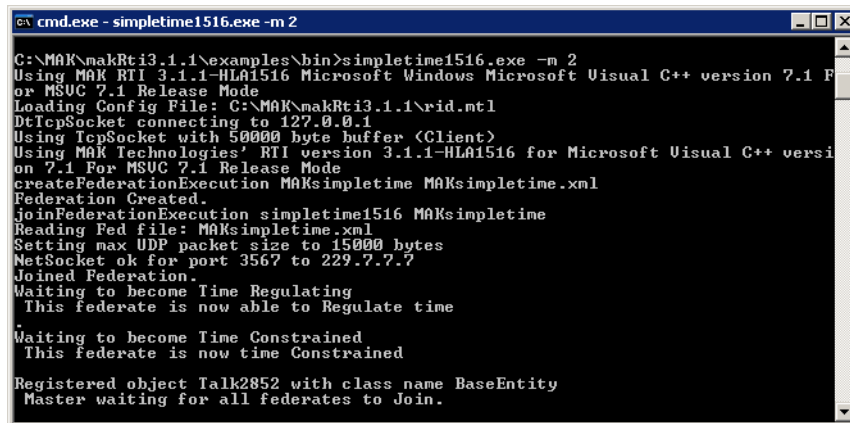
Figure B-4. simpleDDM subscribing federate

B.4. The simptime Example

The simptime example is a simple C++ example of how to use the time management features of the RTI API. The simptime federate tries to create a federation and join it. It enables time management in the RTI by becoming both time managed and time constrained. It publishes and subscribes to the BaseEntity object class and registers an object of that class. Using a federation synchronization point, it synchronizes the federation. Then it publishes and subscribes to the WeaponFire and MunitionDetonation interaction classes.

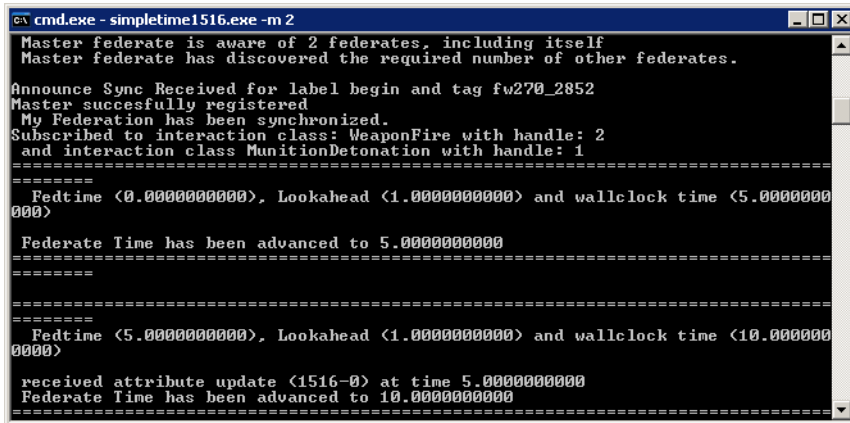
After this initialization process, the federate begins to increase a stored time and position value, updates its attribute values accordingly, and if appropriate, sends a fire or detonate interaction. The federate sends the fire interaction when its stored position exceeds a preset value (the phase-line), or you issue a keyboard command instructing the federate to fire. Then a detonation is sent. If no detonations are sent, the federate continues to update its attribute values and increase its time, while regularly printing out its time state information and its stored position.

If you run two or more simptime federates, each federate discovers the other's object, reflects the other's updated attributes, and receives the other's interactions, as shown in [Figure B-5](#) and [Figure B-6](#).



```
cmd.exe - simletime1516.exe -m 2
C:\MAK\makRti3.1.1\examples\bin>simletime1516.exe -m 2
Using MAK RTI 3.1.1-HLA1516 Microsoft Windows Microsoft Visual C++ version 7.1 For MSVC 7.1 Release Mode
Loading Config File: C:\MAK\makRti3.1.1\rid.mtl
DtTcpSocket connecting to 127.0.0.1
Using TcpSocket with 50000 byte buffer <Client>
Using MAK Technologies' RTI version 3.1.1-HLA1516 for Microsoft Visual C++ version 7.1 For MSVC 7.1 Release Mode
createFederationExecution MAKsimletime MAKsimletime.xml
Federation Created.
JoinFederationExecution simletime1516 MAKsimletime
Reading Fed file: MAKsimletime.xml
Setting max UDP packet size to 15000 bytes
NetSocket ok for port 3567 to 229.7.7.7
Joined Federation.
Waiting to become Time Regulating
This federate is now able to Regulate time
Waiting to become Time Constrained
This federate is now time Constrained
Registered object Talk2852 with class name BaseEntity
Master waiting for all federates to Join.
```

Figure B-5. simletime federate started with -m 2



```
cmd.exe - simletime1516.exe -m 2
Master federate is aware of 2 federates, including itself
Master federate has discovered the required number of other federates.
Announce Sync Received for label begin and tag fw270_2852
Master successfully registered
My Federation has been synchronized.
Subscribed to interaction class: WeaponFire with handle: 2
and interaction class MunitionDetonation with handle: 1
=====
Fedtime <0.000000000>, Lookahead <1.000000000> and wallclock time <5.000000000>
Federate Time has been advanced to 5.000000000
=====
Fedtime <5.000000000>, Lookahead <1.000000000> and wallclock time <10.000000000>
received attribute update <1516-0> at time 5.000000000
Federate Time has been advanced to 10.000000000
=====
```

Figure B-6. simletime federate after second federate starts

B.5. HLA Bounce

The HLA Bounce example is an HLA federate that demonstrates subscription/unsubscription, ownership management, and DDM in the context of a whimsical graphical user interface. Each instance of HLA Bounce displays one or more bouncing balls. You can change their color, view balls controlled by other instances of HLA Bounce, change the ownership of the bouncing balls, and view only those balls that are in a specified region.

B.5.1. Running HLA Bounce

To run HLA Bounce

1. On the Start menu, choose **MAK Technologies** → **MAK RTI x.x** → **Examples** → **HLA Bounce version**, where *version* is 1.3, 1516, or 1516e. The Choose RTI Connection dialog box opens.
2. Select the connection you want to use. To use ownership management or DDM, you should run the rtiexec.
3. Click Connect. HLA Bounce opens (Figure B-7).

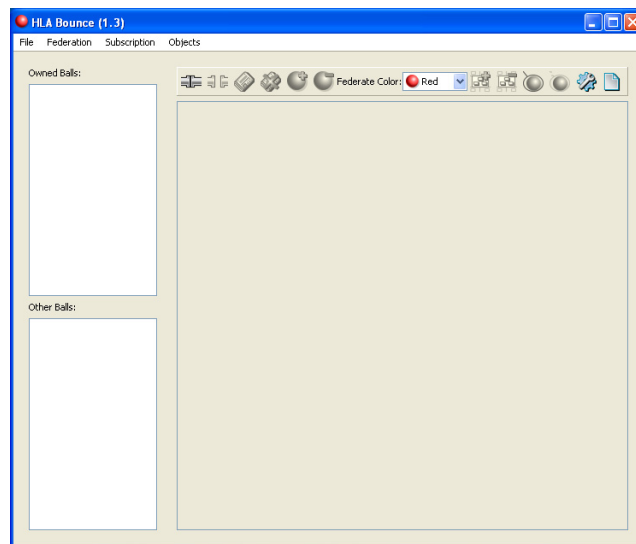


Figure B-7. HLA Bounce

4. Start a second instance of the HLA Bounce example.
5. Select one of the two HLA Bounce windows.
6. Choose **Federation Create/Join**.
7. Choose **Objects** → **Add Ball**. A red ball (the default color) is added to the window and starts bouncing around (Figure B-8).

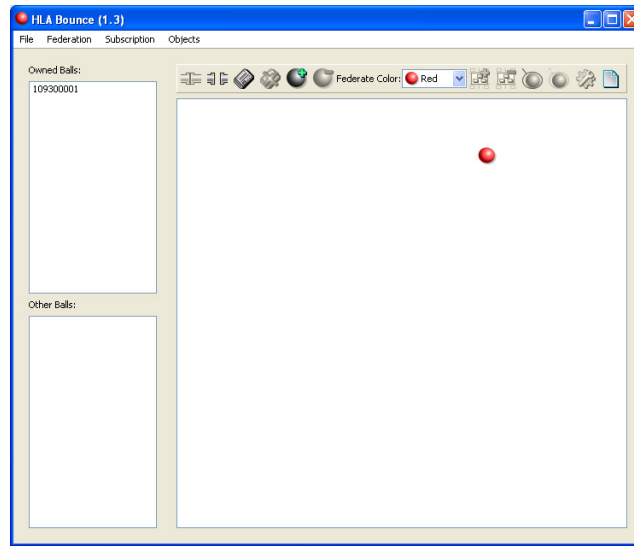


Figure B-8. Bouncing ball

8. Select the other HLA Bounce window.
9. Choose **Federation Create/Join**.
10. In the Federate Color drop-down list, choose a color other than red.
11. Add a ball.

Both windows now have balls bouncing in them, but they only show their own balls.

➤ In one of the two HLA Bounce windows, choose **Subscription** → **Subscribe**.

The window now shows the local ball and the ball being simulated by the other federate (Figure B-9). The locally simulated ball is listed in Owned Balls. The remotely simulated ball is listed in Other Balls.

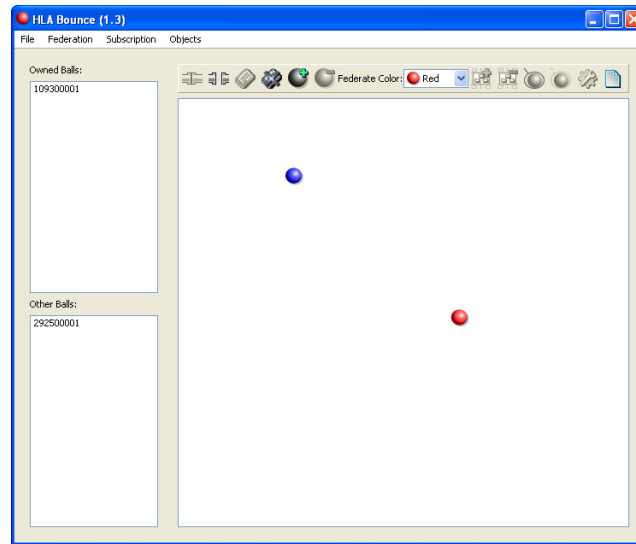


Figure B-9. HLA Bounce subscription enabled

B.5.2. Demonstrating Ownership Management

1. Start two HLA Bounce windows. Use an rtiexec connection and force full compliance.
2. Start one or more balls in each window.
3. In one of the windows, select an entry in Other Balls.
4. Choose **Objects** → **Acquire Ball**. The entry for the remote ball is now in the Owned Balls list and the ball changes color to match the color in this instance of HLA Bounce.

B.5.3. Demonstrating DDM

1. Start two HLA Bounce windows. Use an rtiexec connection and force full compliance.
2. Start one or more balls in each window.
3. In each window choose **Subscription** → **Subscribe**.
4. In one of the windows, Choose **Subscription** → **Add Region**. The cursor changes to a cross-hair.
5. In the window, draw a rectangular area.

6. Watch the two windows and see where the balls are bouncing. In the window in which you drew a region, you see the remote balls only when they are within the region. (You see them all the time in their home window.) In [Figure B-10](#), although the remote federate is simulating two blue balls (as shown in the Other Balls list), only the one within the subscription region is visible. (The federate is not designed to undiscover balls when they leave the region, so they remain listed in the Other Balls list.)

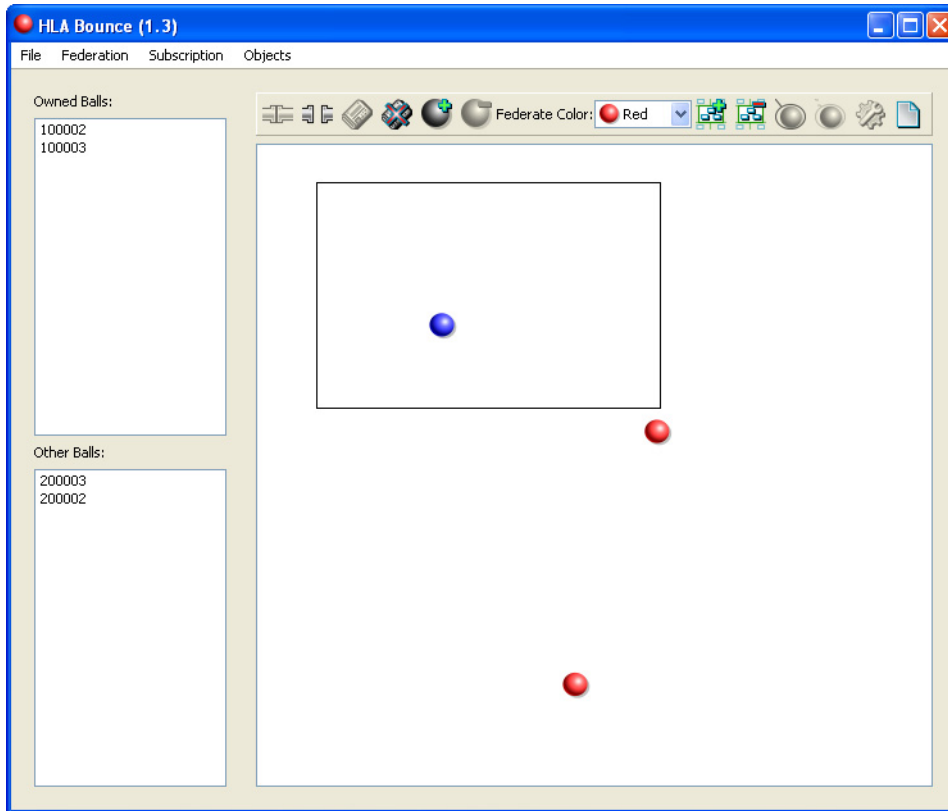
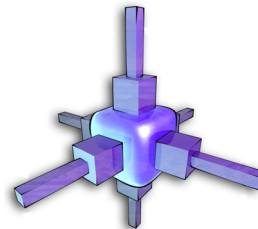


Figure B-10. Subscription with region



Index

- command-line option 6-7, 14-7
 - AutoExit command-line option 6-7, 14-6
 - Daemon command-line option 14-6
 - destAddrString command-line option 5-10, 14-6
 - distributedForwarderPort command-line option 5-10
 - doNotUseRoutesFile command-line option 5-10
 - enableLogging command-line option 6-7
 - forwarderToConnectTo command-line option 5-11
 - graceful command-line option 14-7
 - help command-line option 5-11, 6-7, 14-6
 - ignore_rest command-line option 5-10, 6-7, 14-7
 - manual command-line option 5-11
 - networkInterfaceAddr command-line option 5-11
 - notifyLevel command-line option 5-11, 6-7, 14-7
 - routesFileName command-line option 5-11
 - runDaemonized command-line option 6-7
 - ruthLess command-line option 14-6
 - setLogFileName command-line option 5-11, 14-6
 - setNotifyQuiet command-line option 14-7
 - setRidFileName command-line option 5-11, 14-7
 - setUdpPort command-line option 5-11, 14-7
 - testPage command-line option 6-7
 - version command-line option 5-11, 6-7, 14-7
 - command-line option 5-10
-
- A**
- A command-line option 5-10, 14-6
 - accessing RID parameters 17-22
 - addFedAmbWrapper() function 17-7
 - addObserver() function 17-6
 - addReceiveCreator() function 17-17
 - address
 - IP 11-3
 - RTI Assistant 5-9
 - sending to multiple 11-6
 - addSendCreator() function 17-17
 - advisory message
 - sending 4-3
 - ambassador
 - child 17-7
 - ambImp.h* header file 17-2
 - ambObserver.h* header file 17-6
 - ancillary forwarder 11-7
 - API
 - RTIspy 1-6
 - API Call Statistics
 - displaying 9-13
 - application
 - multi-threaded 2-2
 - argument
 - except 17-7
 - asynchronous
 - callbacks 2-2, 2-4, 2-6
 - message processing 2-6
 - processing 2-4, 10-8
 - asynchronous I/O 2-5, 17-12
 - concepts 2-3
 - attribute
 - displaying for object 8-6
 - passelization 13-9
 - sending 10-2
 - large 10-7

B

- bandwidth
 - minimizing use of 10-4
 - reducing 10-3
- best effort 2-7, 11-3
 - See also* UDP
 - choosing 10-2
 - configuring for LAN 11-3
 - over WAN 11-6
 - topologies 2-7
 - unicast distributed forwarding 11-24
 - WAN 2-7
- binding
 - Java 3-7
- bookmark
 - RTIspy 6-9
- broadcast 2-7
 - address 11-3
 - RTI Assistant 5-9
- broken connection 7-20
- buffer
 - receive 10-7
 - send 10-7
 - TCP 10-7
- buffering
 - update requests 10-9
- building
 - with 1.3 3-2
 - with 1516 3-3
- bundling
 - message A-13
 - packets 10-3

C

- callback
 - asynchronous 2-2, 2-4, 2-6
- centralized server
 - running 6-7
 - web services 6-2, 6-3, 6-6
- centralized TCP forwarding 2-8
 - configuring 11-4
 - on WAN 11-5
- certification
 - HLA 1-2
- chaining wrappers 17-8
- changing
 - notification level
 - LRC log 8-13

- checking
 - RID file consistency 4-9
- child ambassador 17-7
- choosing
 - transport type 10-2
- class
 - descriptors 17-3
 - DtAsyncConnectionMgr* 17-13
 - DtBaseMsg* 17-17
 - DtConnectionMgr* 17-3, 17-10, 17-13, 17-14, 17-16, 17-21
 - DtConnectMsg* 17-19
 - DtDisconnectMsg* 17-19
 - DtFedAmbWrapper* 17-7
 - DtFederate* 17-21
 - DtFederateManager* 17-3
 - DtFedExec* 17-21
 - DtFomClassManager* 17-3
 - DtInteractionManager* 17-3
 - DtInterClassInfo* 17-3
 - DtJoinMsg* 17-17
 - DtlObjPtr* 17-22
 - DtObjectClassInfo* 17-3
 - DtObjectManager* 17-3
 - DtPdu* 17-17
 - DtRIDParameters* 17-22
 - DtRtiAmbassadorImplementor* 17-2
 - subclassing 17-10
 - DtRtiAmbassadorObserver* 17-6
 - function 17-7
 - subclassing 17-6
 - DtRtiConnection* 17-14, 17-16, 17-17, 17-21
 - DtRtiExec* 17-21
 - DtRtiMessage* 17-17
 - DtRtiMsgFactory* 17-17
 - DtRtiObject* 17-3
 - DtSocket* 17-14, 17-15, 17-17
 - DtTcpForwarderThread* 17-19
 - DtUpdateMsg* 17-18
 - DtUpdateRoMsg* 17-17
 - hierarchy in FOM 17-3
 - HLAfloat64Interval* 3-7
 - HLAfloat64Time* 3-7
 - HLAfloat64TimeFactory* 3-7
 - HLAinteger64Interval* 3-7
 - HLAinteger64Time* 3-7
 - HLAinteger64TimeFactory* 3-7
 - interaction
 - displaying 8-8
 - LogicalTimeFactoryFactory* 3-7

- LogicalTimeFactoryImpl* 3-6
- LogicalTimeFactoryImplFloat* 3-6
- LogicalTimeFactoryImplInteger* 3-6
- LogicalTimeImpl* 3-6
- LogicalTimeImplFloat* 3-6
- LogicalTimeImplInteger* 3-6
- LogicalTimeInterval* 3-5
- LogicalTimeIntervallImpl* 3-6
- LogicalTimeIntervallImplFloat* 3-6
- LogicalTimeIntervallImplInteger* 3-6
- object
 - displaying 8-8
- RTIfedTime* 3-5
- clearing
 - console or log window 5-9
- closing
 - RTI Assistant 5-5
- command
 - delete 5-14
- command-line option
 - 5-10, 6-7, 14-7
 - A 5-10, 14-6
 - AutoExit 6-7, 14-6
 - D 5-10, 6-7, 14-6
 - d 5-10
 - Daemon 14-6
 - destAddrString 5-10, 14-6
 - distributedForwarderPort 5-10
 - doNotUseRoutesFile 5-10
 - enableLogging 6-7
 - F 5-11
 - forwarderToConnectTo 5-11
 - graceful 14-7
 - h 5-11, 6-7, 14-6
 - help 5-11, 6-7, 14-6
 - ignore_rest 5-10, 6-7, 14-7
 - K 6-7, 14-6
 - k 14-6
 - l 5-11, 6-7, 14-6
 - M 5-11
 - manual 5-11
 - N 5-11
 - n 5-11, 6-7, 14-7
 - networkInterfaceAddr 5-11
 - notifyLevel 5-11, 6-7, 14-7
 - P 5-11, 14-7
 - q 14-7
 - R 5-11, 14-7
 - r 5-11
 - routesFileName 5-11
 - RTI Forwarder 5-10
 - runDaemonized 6-7
 - ruthLess 14-6
 - setLogFileName 5-11, 14-6
 - setNotifyQuiet 14-7
 - setRidFileName 5-11, 14-7
 - setUdpPort 5-11, 14-7
 - t 6-7, 14-7
 - testPage 6-7
 - v 5-11, 6-7, 14-7
 - version 5-11, 6-7, 14-7
- comment
 - MTL parameter A-2
- communicating
 - over WAN 11-6
- communication 17-11
 - changing structure for 17-14
- compatibility
 - link 1-7
 - network 1-5
 - with other RTIs 1-3
- compiling
 - against MÄK RTI 3-2
 - for 1.3 3-2
 - for 1516 3-3
 - on Windows 3-3, 3-4
 - plug-ins 17-23, 17-24
- compliance
 - HLA 1-2, 12-2
- compressing RTI messages 10-6
- configuration
 - displaying 4-5
 - RID settings 4-7
- configuration file
 - RID 1-4
- configuring
 - asynchronous I/O 2-5
 - best effort on LAN 11-3
 - centralized TCP forwarding 11-4
 - on WAN 11-5
 - create FOM module 15-7
 - DDM 13-2
 - fixed grid 13-3
 - declaration management 10-9
 - delay time for internal messages 12-3
 - diagnostics 5-3
 - discovery processing 7-21
 - distributed forwarding 11-16
 - hierarchical network 11-27
 - UDP 11-24

- distributed region DDM 13-2
- distributed UDP forwarding 11-20
- federate for distributed forwarding 11-15
- FOM module 15-6
- full HLA compliance 12-2
- heartbeat interval 7-18
- HLA 1516 update requests 10-9
- implicit DDM 13-5
- join FOM module 15-7
- latency tests 9-9
- load-balanced forwarder group 11-23
- LRC response interval 7-20
- message compression 10-6
- MOM module 15-6
- MOM services 12-2
- multicast discovery 10-3
- multicast group 10-10
- network
 - buffer size 10-7
 - congestion 10-8
 - device 10-8
 - receive buffer 10-7
- packet bundling 10-3
- port for RTI Forwarder 11-20
- RID files 4-9
- routers and firewalls for distributed forwarding 11-5
- RTI Assistant 5-4
 - messages 5-7
- RTI for FOM modules 15-4
- RTI Forwarders 11-16
- rtiexec 5-3
- RTIspy 6-2
- save and restore 7-22
- sender-side filtering 10-4
- shared memory 14-2
 - performance 14-9
- silent failure 7-21
- single-portal forwarder group 11-21
- smart TCP forwarding 10-4
- time management 12-2
- timeout
 - interval 7-18
 - RTI Forwarder 11-20
- update rate reduction 16-5
- view log prompt 5-6
- connecting
 - to RTI Forwarder 7-20
 - to rtiexec 7-20
- connection
 - RTI 17-3
 - Connection Manager 17-12, 17-14
 - overriding functions 17-15
 - connectMgr.h* header file 17-3
 - consistency
 - RID file
 - checking 4-9, 4-11
 - console
 - clearing messages from 5-9
 - pausing message updates 5-8
 - saving to file 5-8
 - viewing rtiexec 7-22
 - constrained federate 7-13
 - controlling
 - network congestion 10-8
 - conventions
 - manual xvii
 - CPU
 - use 9-2
 - create federation
 - FOM module 15-4
 - create FOM module
 - configuring 15-7
 - merging 15-4
 - create module 15-3
 - createFederationExecution 15-4
 - service 7-6
 - createRTIambassador function 4-4
 - creating
 - custom managers 17-9
 - current
 - FOM 15-3
 - time 7-13
 - custom
 - manager 17-9

D

- D command-line option 5-10, 6-7, 14-6
- d command-line option 5-10
- data
 - availability of 17-20
 - filtering update rate 16-2
- data distribution management 9-2
- DDM 5-2, 7-6, 9-2, 9-16, 10-4, 12-2, 13-9, 16-2,
 - B-7
 - configuring 13-2
 - fixed grid 13-3
 - distributed region
 - configuring 13-2

- distributed region approach 13-7
- example of B-4
- fixed grid DDM 13-9
- implicit 13-17
- interoperability 1-8
- region
 - implicit DDM 13-20
- declaration management 9-2, 10-9, 17-3
 - multicast group 10-10
- declaration management. *See also* DM
- Defense Modeling and Simulation Office 1-2
- delay time
 - internal message 12-3
- delete command 5-14
- deleting
 - federates and federation executions 5-14
 - observers 17-6
 - wrappers 17-8
- demilitarized zone 11-5
- descriptor
 - class 17-3
- destroyFederationExecution service 7-6
- diagnostic
 - configuring 5-3
 - data
 - logging 8-11
 - level 5-11
 - message
 - limiting number of 5-3
 - web services
 - installing 6-2
- dialog box
 - RTI Preferences 5-5, 9-9
- dimensions
 - implicit DDM 13-19
- direct connection
 - RTIs Spy 6-8
- disabling
 - latency testing 9-4
 - RID file consistency checking 4-10
 - RTI Assistant 5-9
 - save and restore A-13
 - shared memory 14-3
 - time management
 - display 7-13
- discovery processing
 - unknown objects 7-21
- displaying
 - API Call Statistics 9-13
 - federate
 - information 8-2, 8-3
 - interactions 8-7
 - federate information 8-4
 - federation-wide messages 7-17
 - FOM
 - interactions 8-8
 - objects 8-8
 - LRC
 - event log 8-10
 - information 8-4
 - messages 8-13
 - publication interest 8-9
 - RID file settings 4-7
 - rtiexec Console entries 7-22
 - subscription interest 8-9
 - time management information 7-13
- distributed forwarding 2-8, 5-10
 - configuring
 - federate 11-15
 - routers and firewalls 11-26
 - forwarder group 11-7
 - hierarchical network 11-26
 - load-balanced 11-14
 - optimizing on WAN 11-12
 - routes file 11-16
 - single-portal interconnection 11-12
 - singleton network 11-9
 - topologies
 - comparison 11-10
 - LAN 11-7
 - UDP 11-24
 - configuring 11-20
 - WAN topology 11-11
- distributed region DDM 13-7
 - configuring 13-2
 - distributed region matching 13-8
 - modifying for implicit DDM 13-20
- distributed singleton forwarder network 11-9
- distributed UDP forwarding 2-7
 - using unicast 11-24
- distributing
 - FED file 7-4
 - HLA 1516 DTD 7-4
- DM 9-2, 10-9, 16-2
 - multicast group 10-10
- DM. *See also* declaration management
- DMZ 11-5
- DtAsyncConnectionMgr* class 17-13
- DtBaseMsg* class 17-17

DtConnectionMgr class 17-3, 17-10, 17-13, 17-14, 17-16, 17-21
DtConnectMsg class 17-19
DTD
 HLA 1516 7-4
DtDisconnectMsg class 17-19
DtFedAmbWrapper class 17-7
DtFederate class 17-21
DtFederateManager class 17-3
DtFedExec class 17-21
DtFomClassManager class 17-3
DtInteractionManager class 17-3
DtInterClassInfo class 17-3
DtJoinMsg class 17-17
DtObjPtr class 17-22
DtObjectClassInfo class 17-3
DtObjectManager class 17-3
DtPdu class 17-17
DtRIDParameters class 17-22
DtRtiAmbassadorImplementor class 17-2
 subclassing 17-10
DtRtiAmbassadorObserver class 17-6
 functions 17-7
 subclassing 17-6
DtRtiConnection class 17-14, 17-16, 17-17, 17-21
DtRtiExec class 17-21
DtRtiMessage class 17-17
DtRtiMsgFactory class 17-17
DtRtiObject class 17-3
DtSocket class 17-14, 17-15, 17-17
DtTcpForwarderThread class 17-19
DtUpdateMsg class 17-18
DtUpdateRoMsg class 17-17
dynamic library 1-3
dynamic link compatibility 1-7
Dynamic Link Compatible 1516 API 1-7
dynamic linking 1-3

E

editing
 RID parameter 4-8
enabling
 latency testing 9-4
 RID file consistency checking 4-10
 save and restore A-13
 shared memory 14-3
 time management
 display 7-13
environment variable

RTI_ASSISTANT_ADDRESS 5-9
RTI_ASSISTANT_DISABLE 5-9
RTI_ASSISTANT_PORT 5-9
RTI_CONFIG 4-2
RTI_RID_FILE 4-2
ethernet device 10-8
event log
 displaying 8-10
evokeCallback 2-6
example
 federates B-2
 HLA Bounce B-7
 rtisimple B-2
 simpleDDM B-4
 simpletime B-5
except argument 17-7
exception
 broken connection 7-20
 throwing 7-21
Exception object 17-7
exec.h header file 17-21
executionList() function 17-21
executive
 RTI 5-2
exiting
 RTI Assistant 5-5
extending FOM
 FOM module, using 15-5
extension
 FOM 15-2

F

-F command-line option 5-11
failure
 FOM module merge 15-4
fault tolerance 7-6, 7-18, 17-19
 orphaned objects 7-19
FDD file 1-4, 7-3
 DTD 7-4
 implicit DDM 13-18
FDD file. *See also* FED file
FED file 1-4, 7-3, 7-6, 10-2, 17-3
 compatibility 1-4
 distributing 7-4
 FOM module in 15-4
 implicit DDM 13-18
 search order 7-4
FED file. *See also* FDD file
fedAmbWrap.h header file 17-7

- federate 7-9
 - abnormally terminated 7-18
 - configuring
 - distributed forwarding 11-15
 - shared memory use 14-2
 - deleting 5-14
 - displaying
 - information about 8-2, 8-3, 8-4
 - interactions 8-7
 - objects 8-6
 - examples B-2
 - filtering data 16-2
 - HLA certification 1-2
 - ID 7-6
 - latency test 9-3
 - listing 5-13
 - log 8-11
 - message
 - displaying 8-13
 - network map 7-7
 - objects and interactions 9-11
 - regulating and constrained 7-13
 - RID file consistency 4-3, 4-9
- Federate Ambassador 9-13
 - monitoring 17-7
 - wrapper 17-7
- Federate FOM View 8-15
- Federate Information window 8-4
- Federate RID File Settings window 8-15
- federate.h* header file 17-21
- federates() function 17-21
- federation
 - data file 1-4
 - deleting 5-14
 - listing 5-13
 - messages 7-17
 - name 7-6
 - save and restore 5-2
 - time 3-5
 - viewing details 7-11
- federation execution. *See* federation
- federation management
 - services 17-3
- Federation Status list 7-12
- Federations window 7-12
- fedex.h* header file 17-21
- fedMgr.h* header file 17-3
- file
 - FED 10-2, 17-3
 - MTL 1-4, A-2
 - RID 4-2, 10-2
 - routes 5-11
- filtering
 - implicit DDM, using 13-17
 - messages 10-4
 - updates 16-2
- firewall
 - configuring for distributed forwarding 11-26
- fixed grid DDM 13-9
 - affect on RTI services 13-15
 - calculating axes 13-12
 - configuring 13-3
 - modifying for implicit DDM 13-21, 13-22
 - range bounds 13-14
- fixnum() function 17-22
- flonum() function 17-22
- FOM
 - attributes and parameters 17-3
 - class hierarchy 17-3
 - current 15-3
 - data 11-4
 - displaying
 - interactions 8-8
 - objects 8-8
 - document data file 1-4
 - extending using FOM module 15-5
 - extension 15-2
 - table 15-5
- FOM module
 - class extension limits 15-5
 - configuring 15-6
 - example configuration 15-8
 - failure mode 15-4
 - limitations on merging 15-5
 - local, configuring 15-8
 - merge order 15-4
 - merging 15-2
 - module FOM 15-2
 - MOM, create, and join 15-3
 - RTI configuration for 15-4
 - specifying
 - create FOM module 15-7
 - join FOM module 15-7
 - table requirements 15-5
 - validating 15-5
- fomMgr.h* header file 17-3
- forceUnlicensedForTwo** parameter A-14
- forwarder group 11-7
 - compared to singleton forwarder 11-10
 - configuring single-portal 11-21

forwarder. *See* RTI Forwarder

full compliance 1-2, 12-2

function

- addFedAmbWrapper() 17-7
- addObserver() 17-6
- addReceiveCreator() 17-17
- addSendCreator() 17-17
- createRTIAmbassador 4-4
- declaration management 17-3
- DtRtiAmbassadorObserver 17-7
- executionList() 17-21
- federates() 17-21
- fixnum() 17-22
- flonum() 17-22
- InitLRCPlugin() 17-4
- InitPlugin() 17-6
- InitRtiexecPlugin() 17-4, 17-21
- netRep() 17-17
- observer 17-6
- overriding 17-7
- readAndProcess() 17-15
- reflectAttributeValues() 17-7
- registerObjectInstance() 17-7
- removeFedAmbWrapper() 17-8
- removeObserver() 17-6
- RTIAmbassador 17-6
- sendInteraction() 17-2, 17-6
- setCreatorFunction() 17-9, 17-13
- SetPluginCreators() 17-4, 17-9, 17-10, 17-16, 17-21
- string() 17-22
- tick() 2-4, 2-5, 17-2, 17-3, 17-15
- updateAttributeValues() 17-2

H

-h command-line option 5-11, 6-7, 14-6

handle

- federate 5-14

header file

- ambImp.h* 17-2
- ambObserver.h* 17-6
- connectMgr.h* 17-3
- exec.h* 17-21
- fedAmbWrap.h* 17-7
- federate.h* 17-21
- fedex.h* 17-21
- fedMgr.h* 17-3
- fomMgr.h* 17-3
- intClassInfo.h* 17-3

- interMgr.h* 17-3

- joinMsg.h* 17-17

- mtlInc.h* 17-22

- objClassInfo.h* 17-3

- obj.h* 17-3

- objMgr.h* 17-3

- RIDparams.h* 17-22

- rtiMsg.h* 17-17

- updateRoMsg.h* 17-17

heartbeat 7-18, 17-19

interval

- configuring 7-18

hierarchical network 11-26

- configuring 11-27

- distributed forwarder 11-26

hierarchy

- FOM classes 17-3

HLA

API

- optimizing performance 9-2

certification 1-2

compliance 1-2

- configuring full 12-2

example applications B-2

specification 1-2

HLA 1.3

- interoperability with 1516 1-8

HLA 1516

- configuring 10-9

- DTD 7-4

- interoperability with 1.3 1-8

- logical time 3-6

- specifying RID file 4-4

HLA Bounce

- example B-7

HLA Evolved

- time implementation 3-7

HLAfloat64Interval class 3-7

HLAfloat64Time class 3-7

HLAfloat64TimeFactory class 3-7

HLAinteger64Interval class 3-7

HLAinteger64Time class 3-7

HLAinteger64TimeFactory class 3-7

HLAlogicalTimeFactoryFactory 3-7

I

IEEE 1516 1-2, 1-7

- API 1-3

implementing

- logical time 3-5
- implicit DDM 13-17
 - adding to FED or FDD file 13-18
 - associating DDM regions 13-20
 - configuring 13-5
 - dimension 13-19
 - initial bounds 13-19
 - modifying distributed region extents 13-20
 - modifying fixed grid DDM 13-21, 13-22
 - normalization 13-19
- information
 - displaying federate 8-4
- initializing plug-in 17-4
- InitLRCPugin() function 17-4
- InitPlugin() function 17-6
- InitRtiexecPlugin() function 17-4, 17-21
- installing
 - RTIsPy 6-2
- intClassInfo.h* header file 17-3
- interaction
 - class
 - displaying in RTIsPy 8-8
 - displaying federate 8-7
 - displaying FOM 8-8
 - federate 9-11
 - ReportServiceInvocation 17-7
 - responding to 17-6
 - sending 10-2
- intercepting calls 17-7
- interMgr.h* header file 17-3
- internal message 11-4
 - configuring delay time 12-3
- reliable
 - enabling for FOM modules 15-6
 - smart forwarding 10-6
 - transmission rate 12-4
- internal state 17-10
- interoperability
 - 1.3 and 1516 1-8
- interval
 - response 7-20
- IP address 5-10, 11-3

J

- Java
 - binding 3-7
- join FOM module
 - configuring 15-7
 - merging 15-4

- join module 15-3
- joinFederationExecution service 7-6
- joinMsg.h* header file 17-17

K

- K command-line option 6-7, 14-6
- k command-line option 14-6

L

- l command-line option 5-11, 6-7, 14-6
- LAN
 - configuring
 - best effort 11-3
 - TCP forwarding 11-4
 - distributed forwarding
 - topologies 11-7
 - topology comparisons 11-10
 - sending messages on 2-7
- latency 2-5
- latency test 9-3
 - configuring 9-9
 - continuous 9-8
 - details 9-7
 - enabling or disabling 9-4
 - lightweight mode 9-4
 - manual 9-8
 - results 9-5
 - running 9-4
- Latency Testing page 9-9
- LBTS 5-2, 7-13
- length
 - shared memory queue 14-4
- libfedtime* library 3-2, 3-5
- libfedtime1516* library 3-2, 3-6
- libfedtime1516e
 - library 3-7
- libmtl* 17-22
- library
 - for compiling 3-2
 - libfedtime* 3-5
 - libfedtime1516* 3-6
 - libfedtime1516e 3-7
- librti1516* 3-2
- libRTI-NG* 3-2
- lightweight mode 7-5
 - latency test 9-4
 - limitations 7-6
 - MOM DDM and time management 5-2

- RTI Assistant 7-5
- link-compatibility 1-3
- linking
 - dynamic 1-3
 - on Windows 3-3, 3-4
 - plug-ins 17-23, 17-24
 - to 1.3 3-2
 - to 1516 3-3
- listing
 - federations and federates 5-13
- load-balanced forwarder group
 - configuring 11-23
- load-balanced forwarder network 11-14
- loading
 - plug-in 17-24
- Local Component Notification History window 7-17
- local FOM module
 - configuring 15-8
- Local RTI Component 1-6, 4-2
- Local RTI Components button 7-10
- local server
 - starting automatically 6-6
 - web services 6-2, 6-3, 6-6
- localDeleteObjectInstance service 8-7
- log
 - clearing 5-9
 - federate
 - displaying 8-13
 - file 8-11
 - LRC 8-11
 - changing notification level 8-13
 - network statistics 9-10
 - pausing 5-8
 - rtiexec
 - displaying recent entries 7-22
 - rtiexec output 5-3
 - saving to file 5-8
 - window 5-8
- log update interval
 - RTI Assistant 5-5
- logging
 - network statistics 9-10
- logical time
 - 1516 3-6
 - implementing 3-5
- LogicalTime 3-6, 3-7
- LogicalTimeFactoryFactory
 - RTI 1516 3-6
- LogicalTimeFactoryFactory* class 3-7

- LogicalTimeFactoryImpl* class 3-6
- LogicalTimeFactoryImplFloat* class 3-6
- LogicalTimeFactoryImplInteger* class 3-6
- LogicalTimeImpl* class 3-6
- LogicalTimeImplFloat* class 3-6
- LogicalTimeImplInteger* class 3-6
- LogicalTimeInterval 3-6, 3-7
- LogicalTimeInterval* class 3-5
- LogicalTimeIntervalImpl* class 3-6
- LogicalTimeIntervalImplFloat* class 3-6
- LogicalTimeIntervalImplInteger* class 3-6
- lookahead 7-13
- lower bound time stamp 5-2, 7-13
- LRC 1-6
 - communicating with remote 17-11
 - delay in response to 7-20
 - event log
 - displaying 8-10
 - information 8-4
 - I/O thread 2-5
 - log 8-11
 - notification level 8-13
 - logging diagnostics 8-11
 - message
 - displaying 8-13
 - reconnecting to RTI Forwarder 7-20

M

- M command-line option 5-11
- MÄK RTI Components list 7-7, 7-10
- MÄK Technologies Lisp 1-4, A-2
- management object model. See MOM
- manager
 - creating custom 17-9
 - RTI 17-3
- manual
 - conventions xvii
 - latency test 9-8
- merge order
 - FOM module 15-4
- merging
 - create FOM module 15-4
 - FOM modules 15-2, 15-4
 - join FOM module 15-4
 - MOM module 15-4
- message
 - asynchronous processing 2-6
 - best effort 11-3
 - bundling 10-3, A-13

- compressing 10-6
 - delay time for internal 12-3
 - diagnostic 5-3
 - displaying RTI 9-12
 - federation-wide 7-17
 - forwarding internal 10-6
 - latency 2-5
 - log
 - pausing updates 5-8
 - RTI Assistant 8-11
 - logging LRC 8-11
 - LRC
 - displaying 8-13
 - queue A-23
 - RTI Assistant
 - configuring 5-7
 - sending 17-12
 - large 10-7
 - transmit rate for internal 12-4
 - viewing in RTIspy 8-10
 - message queue
 - size of 14-5
 - Messages page 9-12, 9-16
 - Modeling and Simulation Coordination Office 1-2
 - modular FOM. See FOM module
 - modules
 - merging 15-4
 - MOM 5-2, 7-6, 9-16, 12-2, 17-7
 - in FOM modules 15-3
 - interoperability 1-8
 - module 15-3
 - service reports 12-2
 - MOM module
 - configuring 15-6
 - merging 15-4
 - specifying 15-6
 - MOM services
 - configuring 12-2
 - monitoring
 - Federate Ambassador
 - services 17-7
 - RTI service invocations 17-6
 - MTL 1-4
 - file A-2
 - parameters A-4
 - millInc.h* header file 17-22
 - multicast 2-7, 10-8, 11-6
 - address 11-3
 - RTI Assistant 5-9
 - discovery
 - configuring 10-3
 - group 10-9
 - address 5-10
 - configuring 10-10
 - multi-homed host 10-10
 - multiple
 - address 11-6
 - multithreaded application 2-2
 - multithreading 2-5
- ## N
- N command-line option 5-11
 - n command-line option 5-11, 6-7, 14-7
 - name
 - shared memory region 14-4
 - name reservation
 - in lightweight mode 7-6
 - NAT 6-8, 11-26
 - See also network address translation
 - nesting wrappers 17-8
 - netRep() function 17-17
 - network
 - compatibility 1-5
 - congestion 10-8
 - device 10-8
 - overhead 9-2
 - statistics
 - log files 9-10
 - logging 9-10
 - topologies
 - LAN 2-7
 - network address translation 6-8, 11-26
 - See also NAT
 - network map 7-7
 - node 7-8
 - opening rtiexec page 7-12
 - network monitoring tool. See RTIspy Network Monitoring window
 - network receive buffer
 - configuring 10-7
 - node
 - information 7-8
 - network map 7-8
 - normalization
 - implicit DDM 13-19
 - notification
 - configuring 5-7
 - level 5-11

notification level
 changing for LRC log 8-13

O

objClassInfo.h header file 17-3
 object
 attributes sent 10-2
 class
 displaying in RTIspy 8-8
 displaying attributes 8-6
 federate 9-11
 management 17-3
 orphan 7-18, 7-19
 undiscovering 8-7
 unknown 7-21
 Object Details button 8-6
 Object Details window 8-6
 Object Instances page 8-14
 returning to 8-5
 objects
 displaying FOM 8-8
 Objects and Interaction Statistics page 9-11
 Objects View 8-6
obj.h header file 17-3
objMgr.h header file 17-3
 observer
 deleting 17-6
 function 17-6
 registering 17-6
 opening
 rtiexec Console 7-22
 optimizing
 distributed forwarding on WAN 11-12
 performance 9-2
 orphaned object 7-18, 7-19
 overhead
 RTIspy 6-9
 overriding
 functions 17-7
 manager 17-9
 RTI services 17-10
 ownership management 7-6, B-7
 service 17-3

P

-P command-line option 5-11, 14-7
 packet
 bundling 10-3

forwarding
 TCP 2-7
 parameter
 accessing RID 17-22
 commenting out A-2
 consistency of 4-9
 forceUnlicensedForTwo A-14
 RID A-4, A-14
 RTI_addForwarder 11-18, 11-19, 11-20, 11-21, 11-27
 RTI_addForwarderConnection 11-27
 RTI_addForwarderRoute 11-27
 RTI_addInterfaceAddr 11-19
 RTI_addressDelay 13-2, A-25
 RTI_allowMulticastForwarding 11-19, 11-20, 11-24
 RTI_autoProvideDelay 10-9, A-22
 RTI_autoProvideSwitch A-16
 RTI_bestEffortSendRetries 10-8, A-22
 RTI_bestEffortSendRetryWaitUsec 10-8, A-22
 RTI_catchFedAmbExceptions A-20
 RTI_checkFlag A-20
 RTI_configureConnectionWithRid 5-11
 RTI_connectionTestInterval A-13
 RTI_conveyOnlyAvailableDimensions A-25
 RTI_conveyProducingFederateSwitch A-16
 RTI_conveyRegionDesignatorSwitch A-16
 RTI_crcCheckFedFile A-20
 RTI_dataDistMgmt 13-2, A-7
 RTI_ddmFixedGrid 13-3, A-7
 RTI_ddmFixedGridFilename 13-3, 13-4, A-7
 RTI_defaultFedFile 7-3, A-20
 RTI_defaultTimelImplementation A-21
 RTI_delaySubscriptionEvaluationSwitch A-16
 RTI_deleteOrphans 7-19, A-9
 RTI_destAddrString 5-10, 11-3, 11-6, 11-24, A-5
 RTI_detachNotifyLevelFromStdOut A-18
 RTI_discoveryMsgBundlingDelay A-13
 RTI_distributedForwarderPort 11-18, 11-20, 11-26, A-12
 RTI_distributedUdpForwarderMode 11-6, 11-7, 11-15, 11-24, A-11
 RTI_distributeFedFile 7-4, 15-4, A-8, A-20
 RTI_dualTransmitFirstInteractionRegions A-23
 RTI_dumpFed A-18
 RTI_dumpRid A-18
 RTI_enableAttributeAdvisory 4-3, A-17
 RTI_enableAttributeScopeAdvisory 4-3, A-17
 RTI_enableBestEffortSendRetry 10-8, A-22
 RTI_enableClassAdvisory 4-3, A-17

- RTI_enableFederateHeartbeat 7-18, A-9
- RTI_enableFedexMsgRouting 10-6, A-13
- RTI_enableFomBackwardsCompatibility A-8
- RTI_enableHlaObjectNamePrefix A-8
- RTI_enableInteractionAdvisory 4-3, A-16
- RTI_enableLrcWebservice 6-2, A-19
- RTI_enableLrcWebserviceEventLog 6-3, A-19
- RTI_enableMessageThrottling 12-3, 12-4, A-22
- RTI_enableNameReservation A-9
- RTI_enableNetworkMonitoring 9-10, A-18
- RTI_enableNetworkTesting 9-4, A-18
- RTI_enablePacketBundling 10-3, A-11
- RTI_enableRtiexecWebservice 6-2, A-19
- RTI_enableSaveRestoreWhenUsingRtiexec A-13
- RTI_enableTcpCompression 10-6, A-10
- RTI_enableUdpCompression 10-6, A-10
- RTI_enableUpdateRateReduction 16-5, A-13
- RTI_enableWarningsForDisabledServices A-15
- RTI_exceptionReportingSwitch A-16
- RTI_execMcastDiscoveryDelay A-15
- RTI_extend13and1516interop 1-8, A-6
- RTI_federateHeartbeatInterval 7-18, A-9
- RTI_federateReconnectPause 7-20, A-9
- RTI_federateTimeoutInterval 7-18, A-9
- RTI_flushTimeoutInterval A-21
- RTI_fomDataReliableWhenUsingRtiexec 10-2, 11-3, 11-4, 17-12, A-6
- RTI_fomModuleMerging 15-6
- RTI_fomModuleMerging A-26
- RTI_forceFomDataReliable A-6
- RTI_forceFullCompliance 1-2, 12-2, A-4
- RTI_forceUnlicensedForTwo A-14
- RTI_forwarderGroupInterconnectMethod 11-19, 11-21, 11-27
- RTI_forwarderInterconnectSetupTimeout 11-19, 11-20
- RTI_forwarderRoutesFile 5-11, 11-20, A-12
- RTI_forwardingDelay A-12
- RTI_implicitDdmDefaultSensorRange 13-5
- RTI_implicitDdmDefaultUpdateRange 13-5
- RTI_implicitDdmMaxExtentLimit 13-6
- RTI_implicitDdmMinExtentLimit 13-6
- RTI_implicitDdmOriginLat 13-6
- RTI_implicitDdmOriginLong 13-6
- RTI_implicitDdmParamsFile A-25
- RTI_implicitDdmRegionMargin 13-5
- RTI_internalMsgReliable 15-4
- RTI_internalMsgReliableWhenUsingRtiexec 4-9, 6-2, 7-4, 10-2, 11-3, 11-4, 17-12, A-6, A-8
- RTI_IOLockQueue A-24
- RTI_IOPeriod A-24
- RTI_logfileName 8-11, A-17, A-18
- RTI_logNetworkMonitorStatistics 9-10, A-19
- RTI_lrcMcastDiscoveryDelay A-15
- RTI_maxChannelAddr 13-2, A-25
- RTI_maxForwarderQueue A-23
- RTI_maxIOCount A-24
- RTI_maxIOQueue A-24
- RTI_maxNumFederates 10-4, A-12
- RTI_maxUdpPacketSize A-21
- RTI_mcastDevAddrString 10-8
- RTI_mcastDiscoveryAddrString 10-3, A-15
- RTI_mcastDiscoveryEnabled 10-3, A-15
- RTI_mcastDiscoveryPort A-15
- RTI_mcastDiscoveryTries 10-3, A-15
- RTI_mcastTtl A-21
- RTI_messageThrottling 10-9
- RTI_minChannelAddr 13-2, 13-3, A-25
- RTI_momExceptionLogging 12-2, A-25
- RTI_momExceptionReports 12-2, A-25
- RTI_momFederateUpdateInterval 12-2, A-25
- RTI_momModuleExtensionFileName 15-6
- RTI_momModuleExtensionFileName 15-6, A-27
- RTI_momServiceAvailable 12-2, A-6
- RTI_momServiceReporting 12-2, A-26
- RTI_networkInterfaceAddr A-15
- RTI_notifyLevel A-17
- RTI_pluginDirectory 6-2, A-19
- RTI_preferLocalFomModule 15-8, A-27
- RTI_preferRidToFedSwitchTable 4-3, A-16
- RTI_processingModel 10-8, A-24
- RTI_processUnknownUpdatesForDiscovery 7-21, A-20
- RTI_receiveTolerance 16-6
- RTI_receiveTolerance A-14
- RTI_reconnectEnabled 7-20, A-9, A-10
- RTI_responseInterval 7-20, A-7
- RTI_reuseLogFile 5-3, 8-11, A-17, A-18
- RTI_reuseReleasedObjectHandles A-8
- RTI_ridConsistencyChecking 4-10, A-18
- RTI_routeToAllByDefault A-12
- RTI_RTI_disableUnlicensedForTwo A-14
- RTI_rtiExecLogFileName 5-3, A-17
- RTI_rtiExecPerformsLicensing A-14
- RTI_rtiExecReconnectPause 7-20, A-10
- RTI_rtiForwarderLogFileName A-17
- RTI_saveRestoreDirectory A-26
- RTI_saveRestoreTimeout 7-22, A-13
- RTI_saveTransientMessages A-13
- RTI_sendDiscoveredClass A-8

- RTI_serviceReportingSwitch A-16
- RTI_sharedMemExecLogFileNames A-26
- RTI_sharedMemoryMgrMaxWait 14-2, 14-5, 14-12, A-26
- RTI_sharedMemoryMode 14-2, 14-3, A-26
- RTI_sharedMemoryName 14-2, 14-4, A-26
- RTI_sharedMemoryQueueLength 14-2, 14-4, 14-9, A-26
- RTI_singleCallbackPerTick 14-10, 14-11, A-19
- RTI_smartForwardingLevel 10-4, A-12
- RTI_socketReceiveBufferSize 10-7, A-21
- RTI_socketSendBufferSize 10-7, A-21
- RTI_strictFomChecking A-21
- RTI_strictNameReservation A-9
- RTI_tcpBufferSize 10-7, A-21
- RTI_tcpCompressionLevel 10-6, A-10
- RTI_tcpForwarderAddr 7-18, 10-3, 11-4, 11-5, 11-15, A-5
- RTI_tcpNoDelay A-21
- RTI_throwExceptionForDisabledService A-15
- RTI_tickFavorsNetwork A-24
- RTI_timeMgmt 12-2, A-6
- RTI_transmitDelay 12-3, A-22
- RTI_transmitRate 12-4, A-22
- RTI_udpCompressionLevel 10-6, A-10
- RTI_udpPort 4-3, 5-11, 6-8, 11-3, A-5
- RTI_use32BitsForValueSize 10-7, A-10
- RTI_useBusyWaitForTickMinMax A-22, A-27
- RTI_useExceptForUnimplServ 7-21
- RTI_useRandomNumberForFedHandle 7-6, A-8
- RTI_useRtiExec 4-3, 7-4, 11-4, 15-4, A-5, A-6, A-8
- RTI_variableLengthDataUsesNull A-8
- RTI_webserviceAddr 6-3, 6-6, A-7, A-19
- RTI_webserviceEnableForwarding 6-4, A-19
- RTI_webserviceEnableServerAutoStart 6-3, 6-6, A-19
- RTI_webserviceHttpPort 6-3, A-7
- RTI_webserviceNotifyLevel 6-4, A-19
- RTI_webserviceRtiPort 6-3, A-7, A-19
- RTI-addCreateFomModule 15-6
- RTI-addCreateFomModule 15-7, A-26
- RTI-addDestAddrString 11-6, 11-24, A-23
- RTI-addDMInteractionMulticastAddr 10-9, 10-10, A-11
- RTI-addDMObjectMulticastAddr 10-9, 10-10, A-11
- RTI-addForwarderGroup 11-19, 11-21
- RTI-addForwarderGroupAddrString 11-15, A-11
- RTI-addForwarderToGroup 11-19, 11-21, 11-23
- RTI-addJoinFomModule 15-7, A-27
- RTI-addPluginName 17-24, A-19
- RTI-addRidParametersToOverride 4-11, A-18
- RTI-addUpdateRate 16-5, 16-6, A-13
- RTI-addUpdateRateSubscription 16-6, A-14, A-23
- RTI-removeRidParametersToOverride 4-11, A-18
- RTI-setObjectSensorRange 13-5, 13-6
- RTI-setObjectUpdateRange 13-5, 13-6
- RTI-setObjectWeaponRange 13-6
- sending large 10-7
- TimeToLive A-21
- passelization 13-9
- pausing
 - message updates 5-8
- performance 2-2
 - affect of rtiexec use 5-2
 - asynchronous I/O 2-3
 - buffering update requests 10-9
 - effect of tick() function 2-4
 - improving with asynchronous options 10-8
 - latency test 9-3
 - message compression 10-6
 - network congestion 10-8
 - optimizing 9-2
 - RTIspy 6-9, 6-10
 - shared memory 14-9
 - update rate reduction 16-2
- plug-in
 - API 1-6
 - compiling and linking 17-23, 17-24
 - initializing 17-4
 - loading 17-24
 - rtiexec 17-21
- polling RTI 17-20
- pop-up blocker
 - RTIspy 6-9
- pop-up message
 - configuring 5-7
- port
 - RTI Assistant 5-9
 - RTI Forwarder 11-20
 - specifying 5-11
 - UDP 11-3
- portal forwarder 11-12, 11-14
- primary forwarder 11-7
- process model
 - RTI 2-3
- processing
 - callbacks 9-14
 - createFederation, destroyFederation, joinFederation 7-20

- product
 - technical support xvi
 - upgrades xvi
- proxy connection
 - RTIsPy 6-8
- publication
 - displaying interest in 8-9

Q

- q command-line option 5-3, 14-7
- queue
 - message A-23
 - shared memory 14-4

R

- R command-line option 5-11, 14-7
- r command-line option 5-11
- range
 - update, sensor, and weapon for implicit DDM 13-17
- range bounds 13-14
- readAndProcess() function 17-15
- reading
 - data 17-20
- receive buffer 10-7
- receiveInteraction 17-8
- reconnecting
 - to RTI Forwarder 7-20
- reflectAttributeValues() function 17-7
- region
 - associating with implicit DDM 13-20
 - shared memory 14-4
- region matching 13-8
- registering
 - observers 17-6
- registerObjectInstance() function 17-7
- regulating federate 7-13
- reliable transport 2-7, 5-2, 5-3, 7-6
 - See also* TCP
 - choosing 10-2
 - topologies 2-7
 - WAN 2-7
- remote
 - LRC 17-11
- removeFedAmbWrapper() function 17-8
- removeObjectInstance service 8-7
- removeObserver() function 17-6
- report

- MOM service 12-2
- ReportServiceInvocation interaction 17-7
- response interval 7-20
- restore
 - configuring 7-22
 - enabling and disabling A-13
- RID file 1-4, 4-2, 10-2, A-4
 - accessing parameters 17-22
 - compatibility 1-4
 - consistency 4-9
 - among federates 4-3
 - displaying settings 4-5, 4-7
 - editing parameters during runtime 4-8
 - FOM module parameters 15-6
 - HLA 1516
 - specifying programmatically 4-4
 - parameters A-4, A-14
 - search order 4-2
- rid.mtl* 1-4, 4-2, A-4
 - checking consistency 4-11
- RIDparams.h* header file 17-22
- router
 - configuring for distributed forwarding 11-26
- routes file 5-11, 11-16
 - See also routes.mtl*
 - search order 11-20
- routes.mtl* 11-16
 - See also* routes file
 - for firewalls and routers 11-26
 - search order 11-20
- RTI
 - compatibility 1-3
 - configuring for FOM modules 15-4
 - connection 17-3
 - definition 1-2, 1-5
 - executive 5-2
 - Initialization Data file 4-2
 - manager 17-3
 - message types 9-12
 - process model 2-3
 - service
 - monitoring 17-6
 - specification 1-2
- RTI 1.3
 - time 17-11
- RTI 1516
 - LogicalTimeFactoryFactory 3-6
 - name reservation 7-6
 - supported services 1-7
 - time 17-11

- RTI Ambassador 9-13, 17-10
- RTI API Call Statistics page 9-13
- RTI Assistant
 - address 5-9
 - configuring 5-4
 - disabling 5-9
 - displaying messages 7-17
 - latency testing 9-4
 - lightweight mode 7-5
 - log update interval 5-5
 - logging messages 8-11
 - pop-ups and notifications 5-7
 - port 5-9
 - shutting down 5-5
 - view log prompt 5-6
- RTI Chooser 4-2
- RTI Forwarder 1-6, 2-7, 5-10, 7-9, 7-18, 10-3, 11-4
 - command-line options 5-10
 - configuring 11-16
 - hierarchical network 11-27
 - load-balanced group 11-23
 - configuring for routers and firewalls 11-5
 - configuring single-portal 11-21
 - group 11-7
 - hierarchical network
 - topology 11-26
 - load-balanced forwarder network 11-14
 - multiple on LAN 11-7
 - network map 7-8
 - port 11-20
 - portal 11-12, 11-14
 - primary and ancillary 11-7
 - reconnecting to 7-20
 - routes file 5-11
 - singleton 11-9
 - timeout 11-20
 - UDP 11-24
 - using unicast for UDP 11-24
- RTI initialization data file 1-4
- RTI messages
 - displaying number sent 9-12
- RTI Network Component View 7-9
- RTI Preferences dialog box 5-5, 9-9
- RTI services
 - affect of fixed grid DDM 13-15
- RTI_addForwarder parameter 11-18, 11-19, 11-20, 11-21, 11-27
- RTI_addForwarderConnection parameter 11-27
- RTI_addForwarderRoute parameter 11-27
- RTI_addInterfaceAddr parameter 11-19
- RTI_addressDelay parameter 13-2, A-25
- RTI_allowMulticastForwarding parameter 11-19, 11-20, 11-24
- RTI_ASSISTANT_ADDRESS environment variable 5-9
- RTI_ASSISTANT_DISABLE environment variable 5-9
- RTI_ASSISTANT_PORT environment variable 5-9
- RTI_autoProvideDelay parameter 10-9, A-22
- RTI_autoProvideSwitch parameter A-16
- RTI_bestEffortSendRetries parameter 10-8, A-22
- RTI_bestEffortSendRetryWaitUsec parameter 10-8, A-22
- RTI_catchFedAmbExceptions parameter A-20
- RTI_checkFlag parameter A-20
- RTI_CONFIG environment variable 4-2, 7-4
- RTI_configureConnectionWithRid parameter 5-11
- RTI_connectionTestInterval parameter A-13
- RTI_conveyOnlyAvailableDimensions parameter A-25
- RTI_conveyProducingFederateSwitch parameter A-16
- RTI_conveyRegionDesignatorSwitch parameter A-16
- RTI_crcCheckFedFile parameter A-20
- RTI_dataDistMgmt parameter 13-2, A-7
- RTI_ddmFixedGrid parameter 13-3, A-7
- RTI_ddmFixedGridFilename parameter 13-3, 13-4, A-7
- RTI_defaultFedFile parameter 7-3, A-20
- RTI_defaultTimelImplementation parameter A-21
- RTI_delaySubscriptionEvaluationSwitch parameter A-16
- RTI_deleteOrphans parameter 7-19, A-9
- RTI_destAddrString parameter 5-10, 11-3, 11-6, 11-24, A-5
- RTI_detachNotifyLevelFromStdOut parameter A-18
- RTI_discoveryMsgBundlingDelay parameter A-13
- RTI_distributedForwarderPort parameter 11-18, 11-20, 11-26, A-12
- RTI_distributedUdpForwarderMode parameter 11-6, 11-7, 11-15, 11-24, A-11
- RTI_distributeFedFile parameter 7-4, 15-4, A-8, A-20
- RTI_dualTransmitFirstInteractionRegions parameter A-23
- RTI_dumpFed parameter A-18
- RTI_dumpRid parameter A-18
- RTI_enableAttributeAdvisory parameter 4-3, A-17
- RTI_enableAttributeScopeAdvisory parameter 4-3, A-17
- RTI_enableBestEffortSendRetry parameter 10-8, A-22

-
- RTI_enableClassAdvisory parameter 4-3, A-17
 - RTI_enableFederateHeartbeat parameter 7-18, A-9
 - RTI_enableFederateHeartbeatparameter 7-18
 - RTI_enableFedexMsgRouting parameter 10-6, A-13
 - RTI_enableFomBackwardsCompatibility parameter A-8
 - RTI_enableHlaObjectNamePrefix parameter A-8
 - RTI_enableInteractionAdvisory parameter 4-3, A-16
 - RTI_enableLrcWebService parameter 6-2, A-19
 - RTI_enableLrcWebServiceEventLog parameter 6-3, A-19
 - RTI_enableMessageThrottling parameter 12-3, 12-4, A-22
 - RTI_enableNameReservation parameter A-9
 - RTI_enableNetworkMonitoring parameter 9-10, A-18
 - RTI_enableNetworkTesting parameter 9-4, A-18
 - RTI_enablePacketBundling parameter 10-3, A-11
 - RTI_enableRtiexecWebService parameter 6-2, A-19
 - RTI_enableSaveRestoreWhenUsingRtiexec parameter A-13
 - RTI_enableTcpCompression parameter 10-6, A-10
 - RTI_enableUdpCompression parameter 10-6, A-10
 - RTI_enableUpdateRateReduction parameter 16-5, A-13
 - RTI_enableWarningsForDisabledServices parameter A-15
 - RTI_exceptionReportingSwitch parameter A-16
 - RTI_execMcastDiscoveryDelay parameter A-15
 - RTI_extend13and1516interop parameter 1-8, A-6
 - RTI_federateHeartbeatInterval parameter 7-18, A-9
 - RTI_federateReconnectPause parameter 7-20, A-9
 - RTI_federateTimeoutInterval parameter 7-18, A-9
 - RTI_flushTimeoutInterval parameter A-21
 - RTI_fomDataReliableWhenUsingRtiexec parameter 10-2, 11-3, 11-4, 17-12, A-6
 - RTI_fomModuleMerging parameter A-26
 - RTI_fomModuleMerging parameter 15-6
 - RTI_forceFomDataReliable parameter A-6
 - RTI_forceFullCompliance parameter 12-2, A-4
 - RTI_forceFullComplianceparameter 1-2
 - RTI_forceUnlicensedForTwo parameter A-14
 - RTI_forwarderGroupInterconnectMethod parameter 11-19, 11-21, 11-27
 - RTI_forwarderInterconnectSetupTimeout parameter 11-19, 11-20
 - RTI_forwarderRoutesFile parameter 5-11, 11-20, A-12
 - RTI_forwardingDelay parameter A-12
 - RTI_implicitDdmDefaultSensorRange parameter 13-5
 - RTI_implicitDdmDefaultUpdateRange parameter 13-5
 - RTI_implicitDdmMaxExtentLimit parameter 13-6
 - RTI_implicitDdmMinExtentLimit parameter 13-6
 - RTI_implicitDdmOriginLat parameter 13-6
 - RTI_implicitDdmOriginLong parameter 13-6
 - RTI_implicitDdmParamsFile parameter A-25
 - RTI_implicitDdmRegionMargin parameter 13-5
 - RTI_internalMsgReliable parameter 15-4
 - RTI_internalMsgReliableWhenUsingRtiexec parameter 4-9, 6-2, 7-4, 10-2, 11-3, 11-4, 17-12, A-6, A-8
 - RTI_IOLockQueue parameter A-24
 - RTI_IOPeriod parameter A-24
 - RTI_logfileName parameter 8-11, A-17, A-18
 - RTI_logNetworkMonitorStatistics parameter 9-10, A-19
 - RTI_lrcMcastDiscoveryDelay parameter A-15
 - RTI_maxChannelAddr parameter 13-2, A-25
 - RTI_maxForwarderQueue parameter A-23
 - RTI_maxIOCount parameter A-24
 - RTI_maxIOQueue parameter A-24
 - RTI_maxNumFederates parameter 10-4, A-12
 - RTI_maxUdpPacketSize parameter A-21
 - RTI_mcastDevAddrString parameter 10-8
 - RTI_mcastDiscoveryAddrString parameter 10-3, A-15
 - RTI_mcastDiscoveryEnabled parameter 10-3, A-15
 - RTI_mcastDiscoveryPort parameter A-15
 - RTI_mcastDiscoveryTries parameter 10-3, A-15
 - RTI_mcastTtl parameter A-21
 - RTI_messageThrottling parameter 10-9
 - RTI_minChannelAddr parameter 13-2, 13-3, A-25
 - RTI_momExceptionLogging parameter 12-2, A-25
 - RTI_momExceptionReports parameter 12-2, A-25
 - RTI_momFederateUpdateInterval parameter 12-2, A-25
 - RTI_momModuleExtensionFileName parameter 15-6, A-27
 - RTI_momModuleExtensionFileName parameter 15-6
 - RTI_momServiceAvailable parameter 12-2, A-6
 - RTI_momServiceReporting parameter 12-2, A-26
 - RTI_networkInterfaceAddr parameter A-15
 - RTI_notifyLevel parameter A-17
 - RTI_pluginDirectory parameter 6-2, A-19
 - RTI_preferLocalFomModule parameter 15-8, A-27
 - RTI_preferRidToFedSwitchTable parameter 4-3, A-16
 - RTI_processingModel parameter 10-8, A-24
 - RTI_processUnknownUpdatesForDiscovery parameter 7-21, A-20
 - RTI_receiveTolerance parameter 16-6
 - RTI_receiveTolerance parameter A-14
 - RTI_reconnectEnabled parameter 7-20, A-9, A-10
 - RTI_responseInterval parameter 7-20, A-7

- RTI_reuseLogFile parameter 5-3, 8-11, A-17, A-18
- RTI_reuseReleasedObjectHandles parameter A-8
- RTI_RID_FILE environment variable 4-2
 - setting programmatically 4-4
- RTI_ridConsistencyChecking parameter 4-10, A-18
- RTI_routeToAllByDefault parameter A-12
- RTI_RTI_disableUnlicensedForTwo parameter A-14
- RTI_rtiExecLogFileName parameter 5-3, A-17
- RTI_rtiExecPerformsLicensing parameter A-14
- RTI_rtiExecReconnectPause parameter 7-20, A-10
- RTI_rtiForwarderLogFileName parameter A-17
- RTI_saveRestoreDirectory parameter A-26
- RTI_saveRestoreTimeout parameter 7-22, A-13
- RTI_saveTransientMessages parameter A-13
- RTI_sendDiscoveredClass parameter A-8
- RTI_serviceReportingSwitch parameter A-16
- RTI_sharedMemExecLogFileName parameter A-26
- RTI_sharedMemoryMgrMaxWait parameter 14-2, 14-5, 14-12, A-26
- RTI_sharedMemoryMode parameter 14-2, 14-3, A-26
- RTI_sharedMemoryName parameter 14-2, 14-4, A-26
- RTI_sharedMemoryQueueLength parameter 14-2, 14-4, 14-9, A-26
- RTI_singleCallbackPerTick parameter 14-10, 14-11, A-19
- RTI_smartForwardingLevel parameter 10-4, A-12
- RTI_socketReceiveBufferSize parameter 10-7, A-21
- RTI_socketSendBufferSize parameter 10-7, A-21
- RTI_strictFomChecking parameter A-21
- RTI_strictNameReservation parameter A-9
- RTI_tcpBufferSize parameter 10-7, A-21
- RTI_tcpCompressionLevel parameter 10-6, A-10
- RTI_tcpForwarderAddr parameter 7-18, 10-3, 11-4, 11-5, 11-15, A-5
- RTI_tcpNoDelay parameter A-21
- RTI_throwExceptionForDisabledService parameter A-15
- RTI_tickFavorsNetwork parameter A-24
- RTI_timeMgmt parameter 12-2, A-6
- RTI_transmitDelay parameter 12-3, A-22
- RTI_transmitRate parameter 12-4, A-22
- RTI_udpCompressionLevel parameter 10-6, A-10
- RTI_udpPort parameter 4-3, 5-11, 6-8, 11-3, A-5
- RTI_use32BitsForValueSize parameter 10-7, A-10
- RTI_useBusyWaitForTickMinMax parameter A-22, A-27
- RTI_useExceptForUnimplServ parameter 7-21
- RTI_useRandomNumberForFedHandle parameter 7-6, A-8
- RTI_useRtiExec parameter 4-3, 7-4, 11-4, 15-4, A-5, A-6, A-8
- RTI_variableLengthDataUsesNull parameter A-8
- RTI_webserviceAddr parameter 6-3, 6-6, A-7, A-19
- RTI_webserviceEnableForwarding parameter 6-4, A-19
- RTI_webserviceEnableServerAutoStart parameter 6-3, 6-6, A-19
- RTI_webserviceHttpPort parameter 6-3, A-7
- RTI_webserviceNotifyLevel parameter 6-4, A-19
- RTI_webserviceRtiPort parameter 6-3, A-7, A-19
- RTI-addCreateFomModule parameter 15-7, A-26
- RTI-addCreateFomModule parameter 15-6
- RTI-addDestAddrString parameter 11-6, 11-24, A-23
- RTI-addDMInteractionMulticastAddr parameter 10-9, 10-10, A-11
- RTI-addDMObjectMulticastAddr parameter 10-9, 10-10, A-11
- RTI-addForwarderGroup parameter 11-19, 11-21
- RTI-addForwarderGroupAddrString parameter 11-15, A-11
- RTI-addForwarderToGroup parameter 11-19, 11-21, 11-23
- RTI-addJoinFomModule parameter 15-7, A-27
- RTI-addPluginName parameter 17-24, A-19
- RTI-addRidParametersToOverride parameter 4-11, A-18
- RTI-addUpdateRate parameter 16-5, 16-6, A-13
- RTI-addUpdateRateSubscription parameter 16-6, A-14, A-23
- RTIAmbassador 17-2
 - functions 17-6
- rtiexec 1-6, 5-2, 7-9
 - configuring 5-3
 - shared memory use 14-2
 - consistency of RID parameters 4-9
 - enabling
 - for FOM modules 15-6
 - latency testing 9-4
 - logging
 - diagnostics 8-11
 - output 5-3
 - network map 7-8
 - opening RTIspy 6-4
 - plug-ins for 17-21
 - required use 5-2
 - response time 7-20
 - running RTI without 7-5
 - save and restore A-13
 - specifying FED file 7-3

- synchronization points 7-16
 - time management display 7-13
 - viewing federation details 7-11
 - rtiexec Console
 - displaying recent log entries 7-22
 - rtiexec page 7-12
 - RTIfedTime* class 3-5
 - RTIinternalError 7-20
 - rtiMsg.h* header file 17-17
 - RTI-removeRidParametersToOverride parameter 4-11, A-18
 - RTI-setObjectSensorRange parameter 13-5, 13-6
 - RTI-setObjectUpdateRange parameter 13-5, 13-6
 - RTI-setObjectWeaponRange parameter 13-6
 - rtiShmQMgr 14-5
 - rtiShmQMgr1516 14-5
 - rtisimple example B-2
 - RTIs Spy 6-1
 - API 1-6, 13-18, 17-2
 - API Call Statistics 9-13
 - bookmark 6-9
 - components list 7-7, 7-10
 - configuring 6-2
 - connection types 6-8
 - displaying
 - federate
 - information 8-4
 - federate interactions 8-7
 - federate objects 8-6
 - FOM interactions 8-8
 - FOM objects 8-8
 - network messages 9-12
 - object and interaction classes 8-8
 - RID settings 4-7
 - editing parameters at runtime 4-8
 - example of using web services 8-14
 - Local RTI Components button 7-10
 - network map 7-7
 - opening web page 6-4
 - overhead 6-9
 - performance issues 6-10
 - pop-up blockers 6-9
 - rtiexec page 7-12
 - running centralized server 6-7
 - starting automatically 6-6
 - synchronization points 7-16
 - time management status 7-15
 - web services 1-6
 - RTIs Spy Network Monitoring window
 - Messages page 9-12
 - Object and Interaction Statistics page 9-11
 - RTI API Call Statistics page 9-13
 - RTIs Spy web server 6-5
 - running
 - centralized web server 6-7
 - latency test 9-4
 - run-time
 - editing RID during 4-8
 - Run-Time Infrastructure
 - definition of 1-5
- ## S
- save
 - configuring 7-22
 - enabling and disabling A-13
 - Save and Restore 7-6
 - saving
 - messages to file 5-8
 - scaffolding definition 15-5
 - search order
 - FED file 7-4
 - fixed grid configuration file 13-4
 - RID file 4-2
 - routes file 11-20
 - send buffer 10-7
 - sender-side filtering 10-4
 - internal messages 10-6
 - sending
 - advisory messages 4-3
 - large attributes and parameters 10-7
 - messages
 - LAN 2-7
 - WAN 2-7
 - object attributes and interactions 10-2
 - sendInteraction() function 17-2, 17-6
 - sensor range 13-17
 - server
 - local or centralized for web services 6-2, 6-3, 6-6
 - RTI central 5-2
 - service
 - createFederation 7-20
 - createFederationExecution 7-6
 - destroyFederation 7-20
 - federation management 17-3
 - joinFederation 7-20
 - joinFederationExecution 7-6
 - object management 17-3
 - ownership management 17-3
 - supported for 1516 1-7

- service reports
 - enabling MOM 12-2
- setCreatorFunction() function 17-9, 17-13
- SetPluginCreators() function 17-4, 17-9, 17-10, 17-16, 17-21
- settings
 - switch value 4-3
- shared memory
 - configuring 14-2
 - memory manager 14-12
 - enabling and disabling 14-3
 - manager
 - configuring performance 14-12
 - message queue size 14-5
 - queue length 14-4, 14-9
 - region name 14-4
 - stopping manager 14-8
 - tick() function 14-11
 - tuning performance 14-9
 - wait interval 14-5
- shared memory manager
 - stopping 14-8
- Shared Memory Queue Manager 14-3, 14-5
 - stopping 14-8
- showing
 - log file 8-13
- shutting down
 - RTI Assistant 5-5
- silent failure 7-21
- simpleDDM example B-4
- simpletime example B-5
- simulation time 7-13
- single-portal forwarder group
 - configuring 11-21
- single-portal interconnection 11-12
- singleton forwarder 11-9
 - compared to forwarder group 11-10
- SISO DLC HLA API 1516 1-3
- SISO Dynamic Link Compatibility HLA API
 - Product Development Group 1-7
- SISO-STD-004.1-2004 1-3, 1-7
- smart TCP forwarding 10-4
 - See also* sender-side filtering
- specification
 - obtaining HLA and RTI 1-2
- specifying
 - port 5-11
 - shared memory queue length 14-4
 - shared memory region name 14-4
 - Warning Tick Interval 5-7

- starting
 - web server automatically 6-6
- static routing 11-5
- stopping
 - shared memory manager 14-8
- string() function 17-22
- subclassing
 - DtRtiAmbassadorImplementor* 17-10
 - manager 17-9
- subscription B-7
 - displaying interest in 8-9
- support
 - technical xvi
- switch value
 - settings 4-3
- synchronization point 5-2, 7-6
 - viewing 7-16
 - in RTIspy 7-16
- synchronous processing 2-4

T

- t command-line option 6-7, 14-7
- table
 - FOM 15-5
 - FOM in FOM modules 15-5
- TCP 5-2, 10-2, 10-3
 - See also* reliable transport
 - buffer 10-7
 - compressing messages 10-6
 - packet 10-3
 - topologies 2-7
- TCP forwarding 2-7, 2-8
 - configuring centralized 11-4
 - configuring smart 10-4
 - internal messages 10-6
 - on WAN 11-5
 - smart 10-4
- technical support xvi
- terminated federate 7-18
- test
 - latency 9-3
- thread
 - asynchronous I/O 2-5
 - federate and asynchronous 2-5
- thread-safety 2-2, 10-8
- tick() function 2-3, 2-4, 2-5, 17-2, 17-3, 17-15
 - not calling 2-6
 - shared memory 14-11
- tick(min,max) A-27

- time
 - advance grant 7-13
 - defining 17-11
 - federation 3-5
 - HLA Evolved 3-7
 - simulation 7-13
- time implementation
 - libfedtime* 3-5
- time management 3-7, 5-2, 7-6, 12-2, 17-11
 - configuring 12-2
 - display 7-13
 - example B-5
 - time implementation 3-5
 - viewing in RTIsPy 7-15
- timeout
 - interval
 - configuring 7-18
 - RTI Forwarder 11-20
- TimeToLive** parameter A-21
- topology
 - distributed forwarding
 - LAN 11-7
 - WAN 11-11
 - network 2-7
- transmit rate 12-4
- transportType 17-12

U

- UDP 10-2, 10-3
 - See also* best effort
 - compressing messages 10-6
 - datagram 10-3
 - distributed forwarding 11-24
 - forwarding 2-7
 - using unicast 11-24
 - over WAN 11-6
 - port
 - setting for best effort 11-3
 - topologies 2-7
- undiscovering
 - object 8-7
- unicast
 - distributed UDP forwarding 11-24
- unimplemented service
 - responding to 7-21
- UNIX
 - compiling and linking 17-23
 - linking on 3-2, 3-4
- unknown objects

- discovery processing for 7-21
- unsubscription B-7
- update
 - filtering 16-2
 - range 13-17
 - reducing rate of 16-2
 - request
 - buffering 10-9
- update rate reduction 16-2, A-13
 - configuring 16-5
- updateAttributeValues() function 17-2
- updateRoMsg.h* header file 17-17
- upgrades xvi
- URL
 - RTIsPy 6-5

V

- v command-line option 5-11, 6-7, 14-7
- validating
 - FOM module 15-5
- view log prompt
 - configuring 5-6
- viewing
 - federation details 7-11
 - log file 8-13
 - RTI messages 8-10
 - rtiexec Console 7-22
 - synchronization points 7-16
 - time management information 7-13
- virtual LAN 11-5
- virtual private network 11-5
- vlutil 17-17
- VPN 11-5

W

- wait interval
 - shared memory 14-5
- WAN
 - centralized TCP forwarding 11-5
 - communicating over 11-6
 - configuring transport type 11-4
 - distributed forwarding
 - optimizing 11-12
 - single-portal interconnection 11-12
 - topology 11-11
 - load-balanced forwarder network 11-14
 - sending messages on 2-7
- Warning Tick Interval

Index

- specifying 5-7
- weapon range 13-17
- Web GUI
 - See* RTIspy
- web services
 - local servers and centralized servers 6-2, 6-3, 6-6
- window
 - log 5-8
 - Object Details 8-6
- Windows
 - compiling and linking on 3-3, 3-4, 17-24
- wrapper
 - deleting 17-8
 - Federate Ambassador 17-7
 - nesting or chaining 17-8
- wwwSpyServer application 6-7

X-Y-Z

- XML file 1-4



Link - Simulate - Visualize

RTI-4.0-1-100722