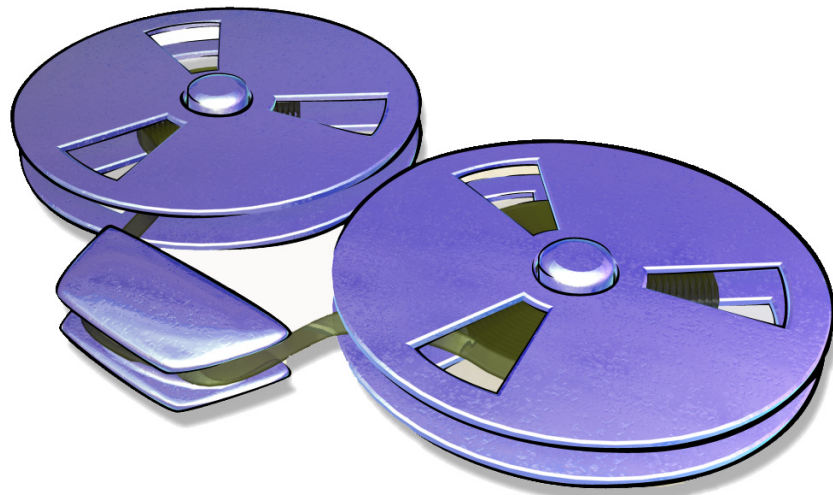




MÄK Data Logger

Users Guide



MÄK Data Logger



Users Guide

Copyright © 2011 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™, VR-TheWorld™, and VR-Vantage™ are trademarks of VT MÄK. MÄK Technologies®, VR-Forces®, RTIspy®, B-HAVE®, and VR-Link® are registered trademarks of VT MÄK.

All other trademarks are owned by their respective companies.

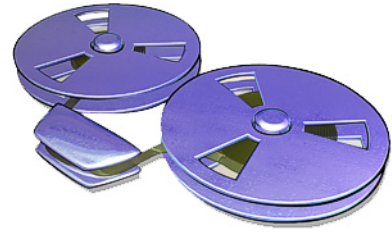
For third-party license information, please see “Third Party Licenses,” on page xvii.

VT MÄK
68 Moulton St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revision LOG-5.1-1-111005



Contents

Preface

How the Manual is Organized	xi
Logger Documentation	xii
MÄK Products	xiii
How to Contact Us	xv
Document Conventions	xvi
Mouse Button Naming Conventions.....	xvi
Third Party Licenses	xvii
Boost License.....	xvii
libXML and libICONV	xviii
pThreads Library.....	xviii
EHS, PCRE, and PME	xviii

Chapter 1 Introduction

1.1. Logger Features	1-2
1.1.1. Logger Interfaces	1-3
1.1.2. Filtering	1-3
1.1.3. Annotations	1-3
1.1.4. Editing	1-4
1.1.5. Data Export	1-4
1.1.6. Sample Logger Files	1-4
1.1.7. Sample Programs	1-5
1.2. Distributed Simulation Standards Supported	1-6
1.2.1. RTI Support	1-6
1.2.2. FOM Support	1-6

Chapter 2 Installing the Logger

2.1. Installing the MÄK Data Logger	2-2
2.1.1. Installing the MÄK Data Logger on Windows	2-2
2.1.2. Installing the Logger on a UNIX System	2-2
2.2. The Logger Directory Structure	2-3
2.3. Installing and Setting Up the License Manager	2-4
2.3.1. Specifying the License Server	2-5
2.4. Installing an RTI	2-7
2.4.1. Installing the MÄK RTI	2-7
2.5. Localizing the Graphical User Interface	2-8
2.5.1. Translating Other Interface Files	2-11
2.5.2. Applying the Language Files	2-11

Chapter 3 Logger Concepts

3.1. What the Logger Does During Recording and Playback	3-2
3.1.1. The Exercise Connection	3-2
3.1.2. What The Logger Does During Recording	3-3
3.1.3. What The Logger Does During Playback	3-3
3.1.4. Recording and Playback Using HLA Time Management	3-4
3.2. Timescale During Playback	3-6
3.2.1. Using Time Scaling with Time Management	3-6
3.2.2. Compensating for Timescale Effects During Playback	3-6
3.3. HLA Support	3-8
3.3.1. FOM Flexibility	3-8
3.3.2. Backward Compatibility	3-8

Chapter 4 Configuring the Logger Connection

4.1. Configuring the Exercise Connection	4-3
4.1.1. Viewing Connection Properties	4-3
4.1.2. Editing a Connection	4-4
4.2. Configuring the Logger for HLA	4-6
4.2.1. Specifying the Name of the Logger Federate	4-7
4.2.2. Specifying the Type of the Logger Federate (HLA Evolved Only)	4-7
4.2.3. Specifying the Federation Execution Name	4-8
4.2.4. Specifying a FED File	4-8
4.2.5. Specifying FOM Modules (HLA Evolved Only)	4-9
4.2.6. Specifying the MIM (HLA Evolved Only)	4-9
4.2.7. Specifying the RPR FOM Version	4-10
4.2.8. Specifying the RPR FOM Revision	4-10
4.2.9. Ignoring Advisory Messages	4-11
4.2.10. Specifying a FOM Mapper	4-11
4.2.11. Configuring Time Management	4-13
4.2.12. Specifying the Logger Subscription List	4-14

4.2.13. Subscribing Passively to Objects and Interactions	4-16
4.3. Configuring the Logger for DIS	4-17
4.3.1. Specifying a UDP Port	4-17
4.3.2. Setting the Exercise ID	4-18
4.3.3. Setting the Site ID	4-18
4.3.4. Setting the Application Number	4-18
4.3.5. Using Asynchronous I/O	4-19
4.3.6. Specifying a Destination Address for Outgoing and Incoming Data	4-19
4.3.7. Setting the Subnet Mask	4-20
4.3.8. Setting the Buffer Size for Sending and Receiving PDUs	4-20
4.3.9. Setting the Multicast Time to Live Value	4-21
4.3.10. Setting the Network Interface Address	4-21
4.3.11. Specifying the Heartbeat	4-22
4.3.12. Specifying the Timeout	4-23
4.3.13. Subscribing to Multicast Groups	4-24

Chapter 5 Configuring the Logger

5.1. Configuring the Logger	5-2
5.2. Configuring Indexing	5-2
5.2.1. Configuring Indexing Using the Command Interfaces	5-3
5.3. Configuring Faster Playback	5-4
5.3.1. The SetEntityTracking Command	5-5
5.3.2. The SetFastJumpsWithoutDiff Command	5-5
5.3.3. The SetPlaybackReadTimeLimit Command	5-6
5.3.4. Configuring HLA Heartbeating	5-7
5.3.5. Configuring HLA Timeouts	5-7
5.4. Configuring Network Read Time	5-9
5.5. Using Data Distribution Management	5-10
5.5.1. Specifying Subscription Regions	5-10
5.5.2. Using Conveyed Regions	5-13
5.5.3. Using Simulation Data	5-13
5.5.4. Configuring DDM	5-14
5.5.5. Configuring DDM in <i>lgrConfig.xml</i>	5-16
5.6. Setting the Notification Level	5-17
5.6.1. Specifying the Log File	5-17
5.7. Editing <i>lgrConfig.xml</i>	5-18
5.7.1. Adding a Command to <i>lgrConfig.xml</i>	5-21
5.8. Summary of Command-line Options for the Logger	5-23

Chapter 6 Recording and Playing Logger Files

6.1. Starting the Logger	6-3
6.1.1. Starting the Logger on Linux	6-3
6.1.2. Starting the Logger in Windows	6-3
6.2. The Logger Window	6-4
6.2.1. Hiding and Displaying Plug-in Views	6-5
6.2.2. Docking and Undocking GUI Plug-in Views and Toolbars ...	6-5
6.3. Connecting to the Simulation Network	6-6
6.3.1. Disconnecting from the Simulation Network	6-6
6.4. Recording Exercises	6-7
6.4.1. Recording Immediately (to a Temporary File)	6-7
6.4.2. Recording to a Specified File	6-7
6.4.3. Recording to Multiple Files	6-8
6.4.4. Pausing the Recording Process	6-10
6.4.5. Stopping the Recording Process	6-10
6.5. Opening Logger Files	6-11
6.5.1. Opening Incomplete Files	6-12
6.5.2. Closing a Logger File	6-12
6.6. Playing Logger Files	6-13
6.6.1. Playing a Logger File	6-13
6.6.2. Pausing Playback of a Logger File	6-13
6.6.3. Playing a Segment of a File	6-13
6.6.4. Looping Playback	6-15
6.6.5. Stopping Playback of a Logger File	6-15
6.6.6. Setting the Playback Speed	6-15
6.6.7. Using High Speed Playback	6-16
6.7. Saving Files	6-17
6.7.1. File Backups	6-17
6.7.2. Saving a File to a New Name	6-17
6.7.3. Saving a Portion of a File to a New File	6-17
6.7.4. Upgrading a Logger File's Format	6-18
6.8. Zooming the Timeline	6-18
6.8.1. Using the Zoom Sliders	6-18
6.8.2. Setting the Zoom from the Set Zoom Dialog Box	6-19
6.8.3. Resetting the Zoom Level	6-19
6.9. Moving the Timepoint to a Specific Time	6-20
6.9.1. Jumping to the Beginning or End of the Playback Range	6-20
6.9.2. Moving to a Specific Time in a File	6-20
6.9.3. Setting the Time by Dragging the Timeline Cursor	6-21
6.10. Viewing a File's Packet Count Histogram	6-21
6.11. Viewing Object Statistics	6-22

Chapter 7 Annotating Files

7.1. Annotating a Logger File	7-2
7.1.1. Adding an Annotation	7-2
7.1.2. Saving an Annotation File	7-4
7.1.3. Opening an Annotation File	7-5
7.1.4. Importing an Annotation File	7-5
7.1.5. Selecting an Annotation	7-5
7.1.6. Editing an Annotation	7-5
7.1.7. Deleting an Annotation	7-6
7.1.8. Moving the Timeline Cursor to the Start of an Annotation	7-6
7.1.9. Jumping to the Next or Previous Annotation	7-7
7.1.10. Zooming the Timeline to an Annotation	7-8
7.1.11. Bounding Playback to an Annotation	7-9

Chapter 8 Filtering Input and Output

8.1. Filtering Logger Input and Output	8-2
8.2. Filtering DIS PDUs by Entity Type	8-2
8.2.1. Removing Entity Types from the Entity Types Lists	8-2
8.2.2. Configuring the List of Common Entity Types	8-3
8.3. Filtering DIS PDUs by PDU Kind	8-3
8.4. Including Custom PDUs in Filter Lists	8-4
8.5. Filtering by Entity ID or Object Name	8-5
8.6. Filtering HLA Exercises by Class	8-6
8.7. Filtering VR-Forces Messages	8-7

Chapter 9 Editing Logger Files

9.1. Editing Files	9-2
9.1.1. Reverting to the Saved Version of a File	9-2
9.2. Clearing and Deleting Segments of a File	9-2
9.3. Merging Files	9-4
9.4. Concatenating Files	9-4
9.5. Trimming Files	9-5
9.6. Offsetting Files	9-5

Chapter 10 Exporting Logger Data

10.1. Exporting Data	10-2
10.2. Exporting Data to a Database	10-3
10.2.1. Configuring an ODBC Data Source	10-5
10.2.2. Exporting DIS Data to a Database	10-6
10.2.3. Exporting HLA Data to a Database	10-7
10.2.4. The Export Log	10-7
10.3. Configuring a Data Source for Microsoft Access	10-8
10.4. Configuring a Data Source for MySQL	10-11

10.5. Importing Logger Data into a Spreadsheet	10-14
10.5.1. Importing Data from MySQL to Microsoft Excel	10-14
10.5.2. Importing Data from Access to Excel	10-18

Chapter 11 Using Logger Commands

11.1. Using Logger Commands on the Command Line	11-2
11.2. Using Logger Commands in the Configuration File	11-2
11.3. Using Logger Commands in the Text Interface	11-3
11.3.1. Entering Commands	11-3
11.3.2. Displaying a List of Commands	11-4
11.4. Command Descriptions	11-4
11.4.1. Argument Types	11-5
11.4.2. System Domain Commands	11-6
11.4.3. Player Domain Commands	11-7
11.4.4. Recorder Domain Commands	11-8
11.4.5. Simulation Domain Commands	11-10
11.4.6. Editor Domain Commands	11-17
11.5. Command List	11-18

Chapter 12 Using The Logger API

12.1. Introduction	12-3
12.2. The Logger Architecture	12-4
12.2.1. Drivers	12-4
12.2.2. Factories	12-5
12.2.3. Interfaces	12-5
12.2.4. The Logger Message Pipeline	12-6
12.2.5. The Simulation Manager	12-6
12.3. Embedding the Logger in an Application	12-7
12.3.1. Embedding Plug-ins in an Application	12-7
12.3.2. The playFile Example	12-8
12.4. Creating a Stand-Alone Logger Application	12-10
12.5. The Logger Plug-in API	12-10
12.5.1. Writing a Plug-In	12-11
12.5.2. Loading Plug-Ins	12-12
12.6. Logger API Coding Conventions	12-13
12.6.1. Namespaces	12-13
12.6.2. Templates	12-13
12.6.3. Shared Pointers	12-14
12.6.4. Virtual Constructors	12-14
12.6.5. Runtime Values	12-15
12.6.6. Thread Safety	12-16
12.6.7. Interface Signal Slots	12-16
12.6.8. Observer Pattern	12-17
12.7. Logger Libraries	12-18
12.7.1. Other Libraries and Plug-ins	12-18

12.8. Compiling and Linking with the Logger API	12-19
12.8.1. Compiling and Linking for Linux	12-19
12.8.2. Compiling and Linking for Microsoft Windows	12-21
12.8.3. Building Examples	12-21

Chapter 13 Remote Control API

13.1. Overview	13-2
13.1.1. Using Multiple Loggers	13-3
13.2. Connecting to Loggers	13-3
13.3. Inspecting Logger State	13-4
13.4. Sending Commands Using the Remote Control Interface	13-5
13.5. Invoking Actions Using Domain Interfaces	13-6
13.6. Processing Responses	13-7
13.7. File Naming Conventions	13-8
13.8. Using the Remote Control Interface	13-8
13.9. Configuring the Logger to use the Remote Control API	13-9
13.10. Configuring the Remote Control Interface	13-10

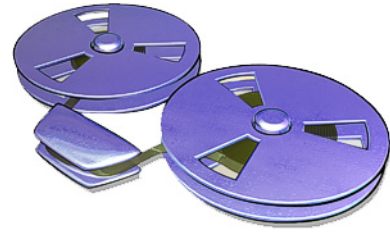
Appendix A Troubleshooting

A.1. Troubleshooting Logger Network Communications (DIS Only)	A-2
A.1.1. Using the Correct Broadcast Address and UDP Port	A-2

MÄK Products Glossary

Index

MÄK Data Logger Quick Reference Card



Preface

The MÄK Data Logger (Logger) records network activity of HLA or DIS exercises and plays it back for review. The Logger has a graphical user interface, a text interface, and an API.

This manual is for persons who will install the Logger, for persons who will record or play back simulations, and for persons who will develop plug-ins or applications using the Logger APIs. It assumes that you are familiar with your operating system and that you know how to work in your computer's windowing environment.

Please consult *MÄK Data Logger Release Notes* for your version of the Logger for release-specific information and updates that occurred after this manual went to press.

How the Manual is Organized

The manual is organized as follows:

Chapter 1, *Introduction*, describes the Logger's features.

Chapter 2, *Installing the Logger*, explains how to install the Logger and related software such as the FLEXlm license manager and an RTI.

Chapter 3, *Logger Concepts*, provides an overview of what the Logger does.

Chapter 4, *Configuring the Logger Connection*, explains how to configure the Logger for use with HLA or DIS.

Chapter 5, *Configuring the Logger*, describes how to configure the Logger for faster performance, how to configure HLA DDM, and how to edit the Logger configuration file.

Chapter 6, *Recording and Playing Logger Files*, shows how to record and play back HLA/DIS network activity using the Logger's graphical user interface.

Chapter 7, *Annotating Files* explains how to create and edit annotations in Logger files.

Chapter 8, *Filtering Input and Output* explains how to use the Logger's filtering capabilities to limit the messages being sent or received.

Chapter 9, *Editing Logger Files* explains how to edit Logger files, including cutting segments, inserting files, and offsetting packet timestamps.

Chapter 10, *Exporting Logger Data* explains how to export Logger files to other formats, such as text and to MySQL or Microsoft Access databases.

Chapter 11, *Using Logger Commands*, explains how to execute Logger commands from the command-line, from a console window, and from the configuration file.

Chapter 12, *Using The Logger API*, describes the Logger API, which allows you to add Logger functionality to your applications, and write plug-ins, which allow you to add new features to the Logger application. .

Chapter 13, *Remote Control API*, explains how to use the Logger Remote Control API to control the Logger from an external program or as an embedded application.

Appendix A, *Troubleshooting*, shows how to correct network problems that may prevent the Logger from communicating properly.

The *MÄK Products Glossary*, defines terms used in the manual.

The *MÄK Data Logger Quick Reference Card* lists command-line options, keyboard shortcuts, and the commands that you can use with the command-line interface.

Logger Documentation

The Logger documentation set is as follows:

- ♦ *MÄK Data Logger Users Guide* describes how to install and use the Logger. It is distributed as an online manual in Adobe Portable Document Format (PDF).
- ♦ A class reference based on the header files is provided in HTML format.
- ♦ Release documentation consists of release notes provided as a PDF file.
- ♦ Online help.
- ♦ *MÄK Products Interoperability Guide* describes general issues of HLA and DIS interoperability and specific issues for running MÄK products together.

The PDF files are in the *./doc* directory.

MÄK Products

The MÄK Data Logger 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) and the Distributed Interactive Simulation (DIS) protocol. 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 four end-user applications (VR-Vantage Stealth, VR-Vantage XR, VR-Vantage PVD, and VR-Vantage IG), the VR-Vantage Toolkit, and VR-Vantage FreeView.
 - 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.
- VR-Vantage PVD provides a 2D plan view display. It gives you 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.
- VR-Vantage Free View is a terrain and 3D model viewer that introduces some of the features of VR-Vantage applications in a useful, free utility.
- ♦ 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.
- ♦ 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.
- ♦ VR-TheWorld™ Server. VR-TheWorld Server is a simple, yet powerful, web-based streaming terrain server, developed in conjunction with Pelican Mapping. Delivered with a global base map, you can also easily populate it with your own custom source data through a web-based interface. The server can be deployed on private, classified networks to provide streaming terrain data to a variety of simulation and visualization applications behind your firewall.
- ♦ VR-inTerra. VR-inTerra is a C++ API for adding terrain agility to applications. It can load, page, or stream terrain from a wide variety of formats or sources into a single, consistent run-time representation, consisting of a collision graph and vector network.

How to Contact Us

For Logger 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/get-licenses.html
Product version and platform information:	www.mak.com/support/product-versions.html
For the free, unlicensed MÄK RTI:	www.mak.com/resources/bonus-material/cat_view/16-bonus-materials/24-mak-high-performance-rti.html
MÄK Community Forum:	www.mak.com/community-forum/1-forum.html

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:

Monospaced	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 Logger home directory. For example, the directory *logger/doc* appears in the text as *./doc*.

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/>

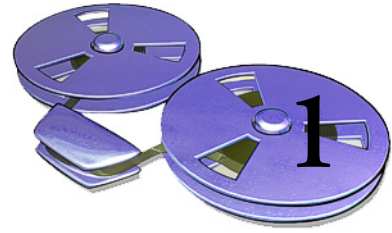
pThreads Library

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

This chapter describes the basic features of the Logger.

Logger Features	1-2
Logger Interfaces	1-3
Filtering.....	1-3
Annotations.....	1-3
Editing	1-4
Data Export.....	1-4
Sample Logger Files.....	1-4
Sample Programs	1-5
Distributed Simulation Standards Supported	1-6
RTI Support.....	1-6
FOM Support	1-6

1.1. Logger Features

The Logger records network activity of HLA and DIS exercises¹ and plays them back for review.

DIS only

In record mode, the Logger monitors all DIS traffic on a specified UDP port and stores it in a file. In playback mode, the Logger transmits DIS data packets stored in a Logger file. When the playback scale is less than or greater than 1.0, the Logger scales the dynamic quantities stored in the PDUs, such as velocity, acceleration, and so on.

HLA only

The HLA version of the Logger performs recording and playback while acting as a separate federate. In record mode, by default, it subscribes to all objects, attributes, and interactions in the RPR FOM, and records updates, interactions, and object removals that it perceives through the RTI. It records values for all published attributes of an object by requesting attribute updates as it discovers each new object.

In playback mode, the HLA Logger acts as a federate that owns each of the objects for which updates appear in the file. The Logger sends interactions and updates only for classes that other federates are subscribed to. The Logger serves as a proxy simulator, answering requests for attribute updates and making objectRemove RTI calls when it encounters removals in the recorded file. In addition, just as in DIS, velocities, accelerations, and angular velocities are scaled in outgoing messages when the Logger's time scale is not equal to zero.

If possible, the Logger uses the same object names as those used in the original simulation. If this is not possible, for example, if another federate is using the name, the Logger lets the RTI assign a name for the object in question.

1. The HLA counterpart to a DIS exercise is a federation execution. For simplicity, we use the term exercise to represent both.

1.1.1. Logger Interfaces

The Logger has the following interfaces:

- ♦ A graphical user interface (GUI)
- ♦ A text-based interface
- ♦ A remote control interface
- ♦ APIs that let you control the Logger through plug-ins, external applications, or over a network.

These interfaces have nearly identical capabilities. (Some features are only available through the GUI; others through the Logger API.) Common features include:

- ♦ Start, stop, or pause when playing a file
- ♦ Navigate to a specific time-point when playing a file
- ♦ Control playback speed and direction
- ♦ Load a specific file for playback.

You can use all the interfaces at the same time.

Localization

The Logger includes translation files that allow you to localize the Logger's GUI. Please see [“Localizing the Graphical User Interface,”](#) on page 2-8 for details.

1.1.2. Filtering

You can filter the messages that the Logger records and sends to reduce network traffic or focus on a particular set of objects or interactions. In HLA, you can specify which classes you want to subscribe to. For details, please see Chapter 8, [Filtering Input and Output](#).

1.1.3. Annotations

You can annotate Logger files. Point annotations identify a specific time in a file. Range annotations identify a segment of a file. You can easily zoom the display to a range annotation and can bind the playback segment to a range annotation. For details, please see Chapter 7, [Annotating Files](#).

1.1.4. Editing

The Logger lets you easily edit Logger files. It supports the following types of edits:

- ♦ Merge files
- ♦ Concatenate files
- ♦ Clear or delete a segment of a file
- ♦ Trim dead space at the beginning of a file
- ♦ Offset the timestamps of the packets in a file.

For details, please see Chapter 9, [Editing Logger Files](#).

1.1.5. Data Export

The Logger can export data to the following formats:

- ♦ Text
- ♦ Logger file
- ♦ MySQL database using ODBC driver
- ♦ Microsoft Access database using ODBC driver.

For details, please see Chapter 10, [Exporting Logger Data](#).

Developers can add additional export formats using the Logger API.

1.1.6. Sample Logger Files

The Logger includes several sample files, including:

- ♦ *berkeleyDemo*. DIS and HLA (1.3, RPR FOM 1.0). An exercise with ground vehicles, rotary-wing entities, fixed-wing entities, dismounted infantry, and a surface vessel (LCAC). The DIS version contains aggregate entity information. These files use the Berkeley database that is distributed with the MÄK's 2D and 3D visualization and computer generated forces products.
- ♦ *maklanddemo*. DIS and HLA (1.3 and 1516, RPR FOM 1.0). A complex scenario with land, air, surface, and sub-surface entities. This file uses the Makland database, which is distributed with VR-Forces and MÄK's 2D and 3D visualization products.
- ♦ *embarkedemo*. DIS and HLA (1.3, RPR FOM 2.0, draft 17). A complex scenario with land, air, and surface entities. It displays entities embarking and disembarking from parent entities. This file uses the Makland database.
- ♦ *VR-Village-CarBombDemo*. DIS and HLA (1.3, RPR FOM 2.0, draft 17). Two versions of a scenario with land and air entities. It shows civilian, dismounted infantry, and enemy combatants in an urban environment. It uses the VR-Village terrain, a fantasy terrain created by MÄK, which is distributed with MÄK's 2D and 3D visualization and computer generated forces products.

- ♦ *IISec_VPMM_Demo_Final-DIS*. DIS only. This is a scenario using ground and air entities in over California, U.S.A. This recording is suitable for use with geocentric terrains, such as those provided by the VR-TheWorld terrain server.
- ♦ *VR-TheWorld_FirstExperienceDIS*. DIS and HLA 1.3 RPR FOM 1.0. This recording is for use with *VR-TheWorld First Experience Guide*.



Most of these files contain View Control messages, which cause the MÄK's 3D visualization products to follow the action automatically, switching among many predetermined viewpoints. For information about manually controlling the eyepoint in this demo, please see the 3D product documentation.

1.1.7. Sample Programs

The Logger includes the following example programs:

- ♦ Simple examples of using the Logger API — `playFile`, `readFile`, `recordFile`, and `writeFile`. Compiled versions of these examples are provided in the `./bin` directories.
- ♦ Example plug-ins — `stateObserver` displays a graph of object statistics. `lgrDump` prints details of each packet to the console. `sampleFilter` shows how to filter out selected packets.
- ♦ An example Logger application - `loggerApp`. You can use this example as a starting point for changing and recompiling the Logger application.
- ♦ A remote control application — `lgrControl`. Compiled versions of the example are provided in the `./bin` directories.

1.2. Distributed Simulation Standards Supported

The Logger has built-in support for DIS and the HLA RPR FOM. Please see release documentation for a list of DIS protocols, RPR FOM versions, and RTI versions supported by your release of the Logger.

1.2.1. RTI Support

The Logger supports the HLA RTI 1.3 specification, the IEEE 1516 specification (SISO Dynamic Link Compatibility HLA API <http://www.sisostds.org/index.cfm>), and HLA Evolved (IEEE 1516-2010). The SISO 1516 API is an alternative mapping of the IEEE 1516 specification to C++. It supports dynamic link compatibility between different RTI implementations.

1.2.2. FOM Support

The Logger provides native support for several versions of the RPR FOM. (Please see the *MÄK Data Logger Release Notes* for a list of supported versions.) The Logger comes with FED files for HLA 1.3, HLA 1516, and HLA Evolved. The default FED file for HLA 1.3 is *VR-Link.fed*, which contains all of the RPR FOM 1.0 objects and interactions, plus a few extra classes described in the file's comments. For HLA 1516, the Logger uses *VR-Link.xml*. By default, the Logger joins the federation execution called VR-Link, and uses this FED file, which is distributed with the Logger, and with other MÄK tools.

The HLA Evolved version of the Logger uses *VR-Link_evolved.xml*.

Using FOMs other than the RPR FOM

For simple recording and playback, the Logger can use any FOM. If you want to use advanced Logger features with FOMs other than the RPR FOM, you can write a VR-Link FOM mapper to provide the information necessary to properly map between VR-Link's internal object model and your FOM. Please see *VR-Link Developer's Guide* for complete details about creating FOM mappers.



Installing the Logger

This chapter explains how to install the Logger and related software.

Installing the MÄK Data Logger	2-2
Installing the MÄK Data Logger on Windows.....	2-2
Installing the Logger on a UNIX System	2-2
The Logger Directory Structure	2-3
Installing and Setting Up the License Manager	2-4
Specifying the License Server	2-5
Installing an RTI.....	2-7
Installing the MÄK RTI	2-7
Localizing the Graphical User Interface.....	2-8
Translating Other Interface Files.....	2-11
Applying the Language Files.....	2-11

2.1. Installing the MÄK Data Logger

A full Logger installation includes three components: the application, the License Manager software, and for HLA operation, an RTI (or Runtime Infrastructure.)

Before you install the Logger, please read the Release Notes to see if there are any special instructions for installation.

After you install the Logger, as described in this section, open your *MÄK Data Logger Users Guide*, which is in `<product_dir>/doc`. Chapter 2 contains complete instructions for installing the License Manager and obtaining a product license.

2.1.1. Installing the MÄK Data Logger on Windows

Note the following:

- ♦ You must have administrator privileges to install MÄK products on Windows Vista.
- ♦ When you install large applications on Vista, such as VR-Forces, there may be a delay of up to several minutes from the time you try to run the setup program to the time that an installation dialog box is displayed. This is due to how Vista scans setup programs before it executes them. If you experience this problem, turning off User Access Control can reduce or eliminate this delay.

Windows versions of the Logger are provided as executable installer files on CD or DVD, or as downloaded files. The installers are named to indicate the compiler used to build that version of the Logger and whether it is a 32 bit or 64 bit version.

- To install the Logger, run the installer. Follow the instructions in the installation wizard.

2.1.2. Installing the Logger on a UNIX System

Before you install the Logger, please read *MÄK Data Logger Release Notes* to see if there are any special instructions for installation.

Linux versions of the Logger are provided as compressed tar files on CD or DVD, or as downloaded files.

To install the Logger on Linux:

1. Create the directory in which you want to install the product.
2. Copy the tar file to the install directory.
3. Uncompress and untar the file:

```
tar -vxzf application.tar.gz
```

where *application* is the product release identification.

2.2. The Logger Directory Structure

The Logger installation process creates a directory structure like the one shown in [Figure 2-1](#).

```

loggerx.x.x
├── bin
├── binDIS
├── binHLA13
├── binHLA1516
├── binHLA1516e
├── data
├── doc
├── examples
├── include
├── lib
├── logger_tapes
├── mkcmake
└── translations

```

Figure 2-1. Logger directory structure

[Table 2-1](#) describes the contents of the Logger directories.

Table 2-1: Logger directory structure

Directory	Contents
<i>bin</i>	Protocol-independent executables.
<i>binDIS</i>	Executable for DIS versions 2.0.4, 2.1.4, IEEE 1278, and IEEE 1278.1a.
<i>binHLA13</i> , <i>binHLA1516</i> , and <i>binHLA1516e</i>	Executables for use with the HLA RTI 1.3, 1516, and HLA Evolved (1516e).
<i>data</i>	Subdirectories contain default configuration files and images used in the GUI.
<i>doc</i>	Documentation.
<i>examples</i>	Examples of the Logger API and the Logger File API.
<i>include</i> and <i>lib</i>	Header files and libraries for the Logger API and Logger File API.
<i>logger_tapes</i>	Sample Logger files.
<i>mkcmake</i>	Build scripts.
<i>translations</i>	This directory contains translations files for localizing the Logger.

2.3. Installing and Setting Up the License Manager

Before you can use a MÄK product, you must obtain a valid license file, install the License Manager, and configure the license server and client machines. The License Manager uses a client-server architecture, so you do not need to install the License Manager on every computer on which you install MÄK products. You only need to install it on the computer that you will use as the license server.



If you have already installed the License Manager for another MÄK product, you do not have to install it again. You just need to make sure you have licenses for your newly installed products.

The License Manager installer is included on MÄK installation media. It is separate from the product installers. You can download the installers from our web site at <http://www.mak.com/support/get-licenses/license-support.html> or you can download directly from at:

- ♦ Windows: <ftp://ftp.mak.com/out/MAKLicenseManager-win-setup.exe>
- ♦ Linux: <ftp://ftp.mak.com/out/MAKLicenseManager-linux-setup.tar.gz>

Complete installation and configuration instructions are included with the license manager installer. Instructions are also available at the license support page mentioned in a previous paragraph.

Some customers use dongle licenses instead of running a license server. Instructions for using dongles are in the license manager documentation.

2.3.1. Specifying the License Server

The first time you run a MÄK application on a particular computer, the License Setup dialog box opens (Figure 2-2). It prompts you to enter the hostname of the license server and optionally, a port number.

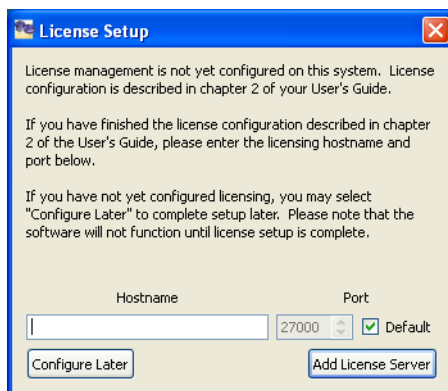


Figure 2-2. License Setup dialog box

If you do not know the hostname of the license server, click **Configure Later**. When you have the hostname, you can start the application again and complete the dialog box. You will not be able to run any MÄK applications until you set up license management.

If you know the hostname, type it in the **Hostname** box. Then click **Add License Server**. The application will start.

i

- ♦ If you are running MÄK products on the license server machine, it is also a client, so you must specify the license server on that machine too.
 - ♦ If you change the license server, the saved configuration will no longer be valid and the License Setup dialog box will open the next time you start a MÄK application.
 - ♦ You can clear the saved license configuration by deleting the cache file. On Windows it is *C:\Documents and Settings\user_name\Application Data\MAK\licenses<n>.xml*. On UNIX, it is *.mak/licenses<n>.xml*. (There may be more than one cache file, for example, *licenses1.xml* and *licenses2.xml*.)
-

The MAKLMGRD_LICENSE_FILE Environment Variable

As an alternative to using the License Setup procedure described in the previous section, you can configure the license server in an environment variable. The MAKLMGRD_LICENSE_FILE environment variable identifies the server machine. If you set this environment variable, it overrides the settings stored by the License Setup procedure.



To view a video that illustrates this procedure on Windows, please go to <http://www.mak.com/html/setenvvar.htm>

The syntax for the environment variable is: `@server_name`. For example, if the server machine is oak, set the environment variable to `@oak`.

The following sections explain how to set environment variables on the different platforms that MAK products run on.

Windows

To add the MAKLMGRD_LICENSE_FILE in Windows:

1. Open the Control Panel.
2. Click System. The System Properties dialog box opens.
3. Click the Advanced tab.
4. Click the Environment Variables button. The Environment Variables dialog box opens.
5. Click the New button. The New System Variable dialog box opens.
6. In the Variable Name field, enter MAKLMGRD_LICENSE_FILE.
7. In the Variable Value field, enter `@server_name`, where *server_name* is the name of the license server.
8. Click OK to back out of each dialog box and set the variable.

UNIX

On UNIX, you set environment variables in your `.cshrc` (or equivalent startup file). Set the variable similarly to the following example:

```
setenv MAKLMGRD_LICENSE_FILE @oak
```

If you are using the sh or bash shells, you set environment variables in your `.profile` file (or `.bashrc`). Set the variable similarly to the following example:

```
MAKLMGRD_LICENSE_FILE=@oak
export MAKLMGRD_LICENSE_FILE
```

Do not put spaces around the equal (=) sign.

You are ready to run the license server and use your new licenses or MÄK products.

2.4. Installing an RTI

An RTI is a software library (and perhaps supporting executables) that implements an HLA Interface Specification. In HLA, applications exchange FOM data through RTI calls, which means that all HLA applications need to use an RTI.



Because of differences in the low-level network mechanisms used by different RTI implementations (which include, but are not limited to packet layout), applications that want to interoperate in the same federation execution must use the same RTI implementation.

Because RTIs are usually provided as dynamic libraries that implement a fixed API, a federation can often switch from one RTI implementation to another between runs (without even recompiling the applications), but during each run, all participants must agree on which RTI to use, much as they must also agree on which FOM to use.

For the most recent information about the RTI versions supported by MÄK products, please see the release notes for your MÄK application.

2.4.1. Installing the MÄK RTI

To install the MÄK RTI, follow the instructions in Chapter 2 of *MÄK RTI Users Guide*.

Configuring Your System to Use the MÄK RTI

The RTI dynamic libraries must be located somewhere on your dynamic library search path. The path is specified in the PATH environment variable. The path is indicated by an environment variable as follows:

- ♦ Linux — LD_LIBRARY_PATH
- ♦ Windows — PATH.

The MÄK RTI needs to know where to find the following configuration files:

- ♦ The federation configuration file (*.fed*, *.fdd*, or *.xml*) for your federation execution (required)
- ♦ The RID file (*rid.mtl*) (optional).

Put the configuration files in the directory from which you are running, or set the environment variable RTI_CONFIG to the directory that contains them.

Running Applications with the MÄK RTI

In many cases, you do not need an RTI server, such as the rtiexec, or a central application to use the MÄK RTI, however you can run the rtiexec if you want to. (It is required to use certain features of the MÄK RTI.) Be sure the license server is running, then start the applications, and they should be able to communicate. For more information, please see your RTI documentation.

2.5. Localizing the Graphical User Interface

You can localize your MÄK product's graphical user interface so that it uses languages other than English. To do this, you edit a resource file using the following procedure:

1. Start the Qt Linguist utility. It is `./translations/linguist.exe`
2. In the Qt Linguist window, choose **File** → **Open**. Select the file `./translations/product_untranslated.ts`. The Settings for filename dialog box opens ([Figure 2-3](#)).

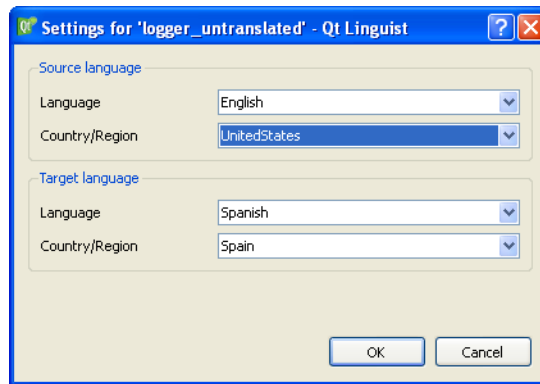


Figure 2-3. Linguist settings dialog box

3. Select values for the source language, target language, and target region or country.
4. Click OK. The translation file is opened and a list of classes is displayed on the left side of the window ([Figure 2-4](#)).

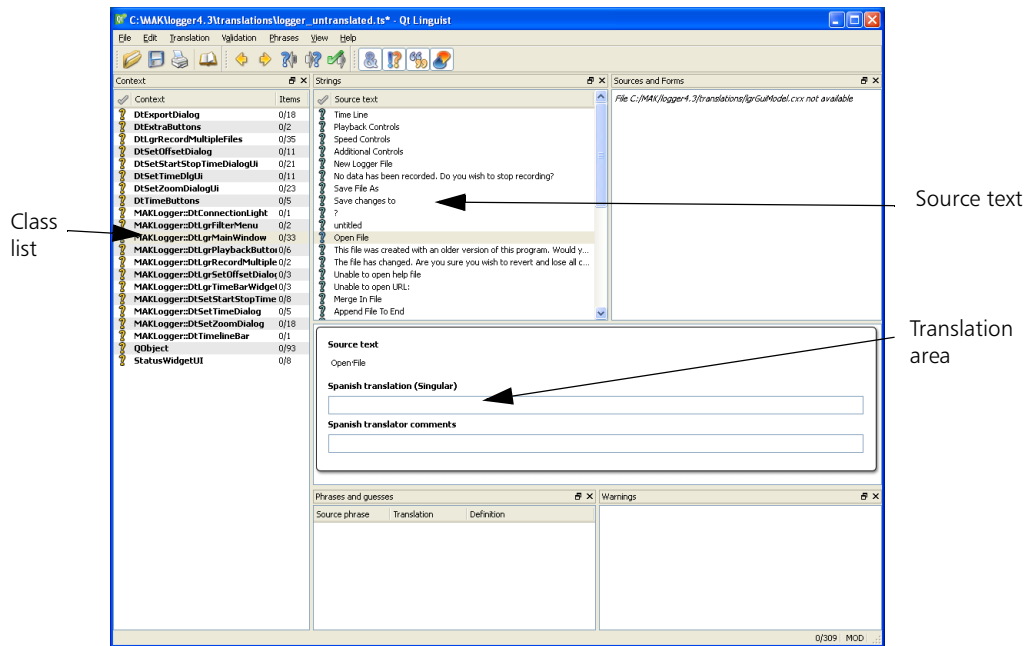


Figure 2-4. Linguist window

5. To translate text, you select each class in turn and translate the text associated with that class. You do not have to understand anything about what the classes mean, you just have to translate the text. In [Figure 2-4](#), *DtLgrMainWindow* is selected. The Strings window lists text that can be translated. Text that has not been translated has a question mark in the checkmark column.
6. In the Source Text list, select the word you want to translate. The text is displayed under Source Text in the translation area (the window below the source text list.) [Figure 2-5](#) shows the text Open File selected.

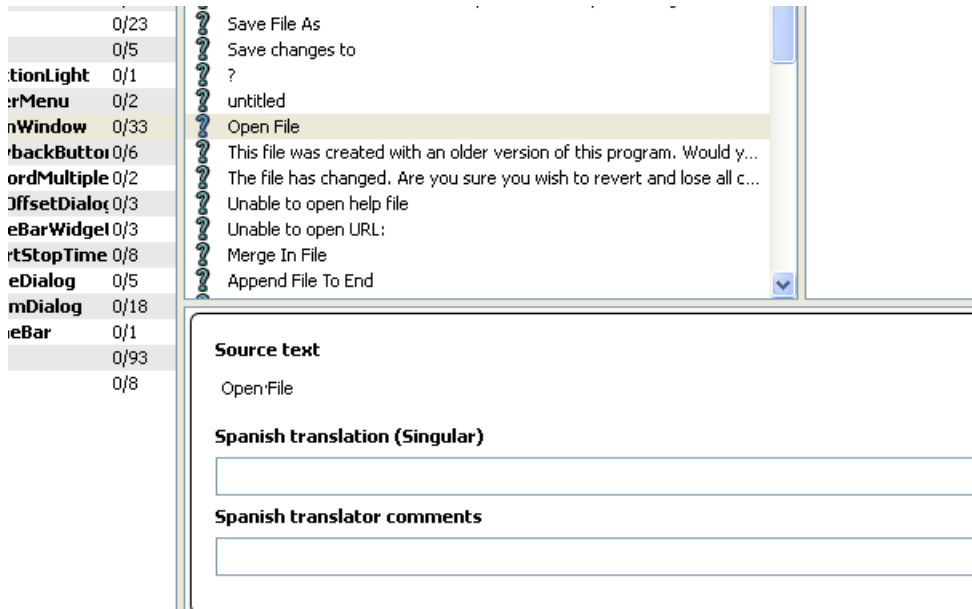


Figure 2-5. Text selected for translation

- Place the cursor in the text box below *Language* Translation, where *Language* is the target language that you selected when you opened the translation file.
- Type the translation text.
- When you are done and you are certain of the translation, select the Done and Next button on the toolbar. A green check mark replaces the question mark (Figure 2-6). If you move to a different text string without clicking Done and Next, a yellow question mark is displayed for the string you translated. When you translate all elements for a class, a green check mark replaces the question mark next to the class name.

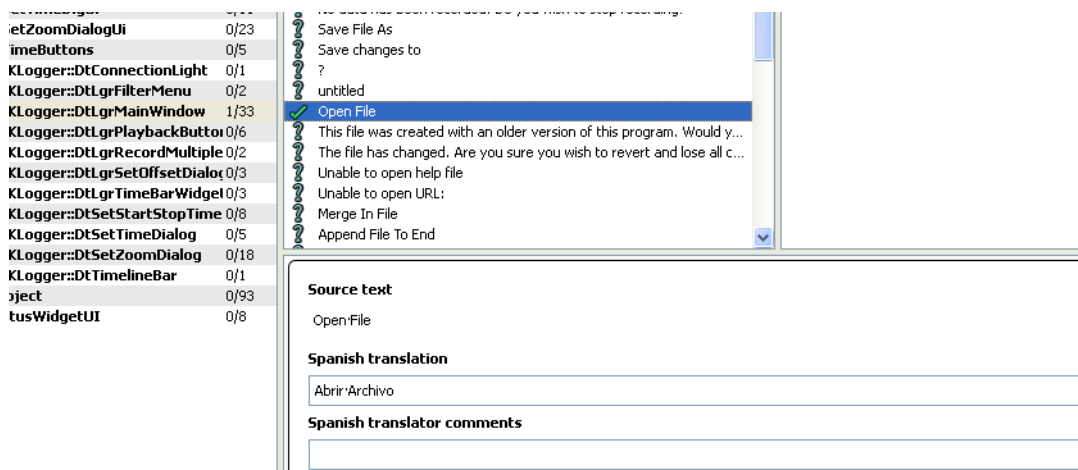


Figure 2-6. Completed translation for Open File

10. Repeat this process for each element of source text for each class.
11. Choose **File** → **Release**. The file is saved in the format *filename.qm*.
12. Rename the file to a name that indicates the language to which you have translated the file, using the two-character abbreviation typically used with your language. For example, a Spanish translation would be renamed *product_es.qm*.
13. Save your translated *.ts* file.

2.5.1. Translating Other Interface Files

The *product_untranslated.ts* file contains all the text used in the Logger interface. However, there may be other text and messages that are not specific to the Logger that are generated by the software that implements the Logger GUI. To translate this text, you need to translate the file *qt_untranslated.ts*. Follow the same procedure as you did for *logger_untranslated.ts*.

2.5.2. Applying the Language Files

To put the language file into effect, use one of the following methods:

- ♦ Set your computer's environment to the proper language (LANG environment variable).
- ♦ Set the locale to the proper language.
- ♦ Start the Logger with the `-G locale` command line option, where *locale* is the two-letter country code for the local language.



Logger Concepts

This chapter describes the Logger's basic functioning.

What the Logger Does During Recording and Playback	3-2
The Exercise Connection.....	3-2
What The Logger Does During Recording.....	3-3
What The Logger Does During Playback.....	3-3
Recording and Playback Using HLA Time Management.....	3-4
Timescale During Playback.....	3-6
Using Time Scaling with Time Management.....	3-6
Compensating for Timescale Effects During Playback.....	3-6
HLA Support.....	3-8
FOM Flexibility	3-8
Backward Compatibility.....	3-8

3.1. What the Logger Does During Recording and Playback

This section describes, in general, what happens when you record and play back exercises. [Figure 3-1](#) illustrates the Logger window with a file loaded.

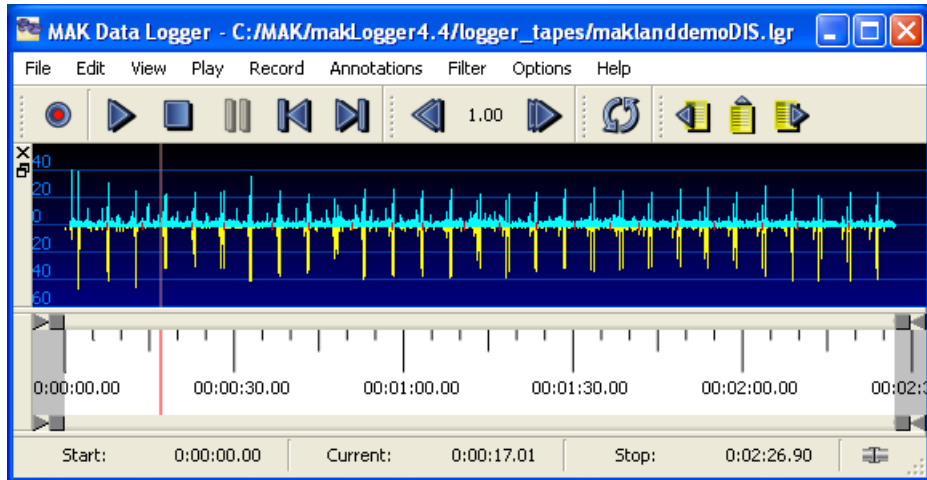


Figure 3-1. The Logger window

3.1.1. The Exercise Connection

Before you can record an exercise or play one back, you must connect to a DIS or HLA exercise. You can set connection parameters in *lgrConfig.xml*, from the command line, or through the GUI. For details, please see “[Configuring the Exercise Connection](#),” on page 4-3.

3.1.2. What The Logger Does During Recording

In record mode, the Logger is in either a recording state or paused state. When paused, the Logger does not record data and does not advance time.

DIS only

By default, the Logger connects to exercises using an exercise ID of 0, which means Any Exercise. When the exercise ID is 0, the Logger records all DIS packets it receives, without regard to exercise ID. With each PDU it records, the Logger stores a time-stamp indicating when the PDU occurred relative to the start of the file.

If you specify that the Logger connect with an exercise ID other than 0, it only records PDUs that have the same exercise ID.

HLA only

The Logger subscribes to objects, attributes, and interactions, and it records updates, interactions, and object removals that it perceives through the RTI. It requests updates as it discovers each new object, and records values for all published attributes of those objects.

If a recorded object has a name beginning with HLA_, the Logger adds LGR_ to the beginning of the name. This allows these objects to be properly published during playback. You can configure this behavior with the `SetDynamicNamePrefix` command.

3.1.3. What The Logger Does During Playback

When the Logger is in playback mode, it is in a playing state, a paused state, or a stopped state. When the Logger is playing, it reads messages from a file and transmits them in sequence over the network. When the Logger is paused, it halts at the current timepoint in the file and stops sending new data.

During playback, you can control the playback speed to produce real-time, fast motion, and slow-motion effects. For more information, please see [“Timescale During Playback,”](#) on page 3-6.

The Logger lets you navigate to any timepoint in a file. For procedures, please see [“Moving the Timepoint to a Specific Time,”](#) on page 6-20.

DIS only

When the Logger is paused, it repeatedly retransmits messages that represent the current state of each entity to maintain the DIS heartbeat. (This retransmission is not needed in HLA.)

In DIS, when you start the Logger and specify a file for playback, the Logger sends the recorded PDUs to the UDP port specified in *lgrConfig.xml*, the command line, or in the Connection dialog box. If the Logger’s exercise ID is 0, it sends each PDU with the same exercise ID as the PDU had when it was recorded. If the Logger’s exercise ID is set to a number other than 0, all of the PDUs get sent with the Logger’s exercise ID.

HLA only

In HLA, the Logger acts as a federate that owns each of the objects represented in the file. The Logger sends interactions and updates only for classes that other federates are subscribed to. The Logger serves as a proxy simulator, answering requests for attribute updates and making `objectRemove` RTI calls when it encounters removals in the recorded file.



By default, HLA 1516 and HLA Evolved use name reservation. When name reservation is in effect, the RTI does not allow the Logger to create entities with names that have previously been used. This becomes an issue when you loop a Logger file. The RTI can rename objects the second time they are created, but this renaming does not get passed along to the attributes of objects such as aggregates or to interaction parameters. They will look for objects that have the original names and will not be able to find them.

If you are using the MÄK RTI, you can work around this problem by disabling the `RTI_enableNameReservation`, `RTI_strictNameReservation`, and `throwExceptionForDisabledService` parameters in *rid.mtl*.

3.1.4. Recording and Playback Using HLA Time Management

The Logger supports HLA Time Management in both record and playback mode. In record mode, the Logger participates as a time-constrained federate. In playback mode, it participates as a time-regulating federate. The contents of the files created in time-managed mode are identical to those created when not using Time Management, however the arrival time recorded for each packet is determined differently when using Time Management.



HLA Time Management is not supported in the text interface.

Playback Using Time Management

You can play any Logger file with or without Time Management enabled. If you play a file without using Time Management, packets are sent as RO updates. If you are using Time Management, they are sent as TSO updates.

To play files with Time Management enabled, you must use a FED file that is coded to indicate which FOM attributes and parameters are to be sent in TSO mode. The Logger comes with a properly written FED file named *VR-LinkTM.fed* for use with Time Management.

When the Logger joins a federation in time-managed mode, it asks to be a time-regulating federate. Its initial time request is the current value of the lower bound time stamp (LBTS). If there are no other time regulating federates (LBTS is infinity), the initial time request is zero.

The Logger starts playback at the point in the file where the arrival time equals or exceeds the current federation time grant plus the lookahead. Therefore, it is possible that the Logger could skip significant portions of the file, or even all of it. To ensure that you play the entire file, make sure that the Logger is the first time regulating federate to join the federation execution, or offset the times in the file relative to an appropriate starting time value.

Once it starts playback, the Logger advances at whatever rate is determined by the time scale setting. For more details, see [“Timescale During Playback,”](#) on page 3-6.

Some Logger Features are Restricted When You Use Time Management

When you play a file with Time Management enabled, the following restrictions apply:

- ♦ You cannot jump to different points in the file
- ♦ You cannot pause playback
- ♦ You can stop, but you cannot restart unless you restart the entire federation execution.

3.2. Timescale During Playback

The Logger lets you play a file at any speed within the capabilities of your platform.

You determine the speed by specifying a timescale value: the factor by which time is multiplied during playback. The default timescale setting of 1.0 causes a Logger file to play forward at normal speed. A timescale of 0.5 results in half-speed playback.

For procedures, please see [“Setting the Playback Speed,”](#) on page 6-15.

3.2.1. Using Time Scaling with Time Management

When you play a Logger file using Time Management, the Logger acts as a time regulating federate. If the Logger sends updates more slowly than other time regulating federates, it may slow down the federation execution. If it sends updates much more quickly than the other federates, it could overwhelm the RTI. Therefore, you should try to set the time scale to a value that is synchronized with the overall speed of federation execution.

3.2.2. Compensating for Timescale Effects During Playback

The Logger has corrective features that adjust the content of outgoing messages for different playback speeds. Under DIS, the Logger generates interim PDUs that prevent receiving applications from inappropriately timing out entities during slow-speed playback. Under both DIS and HLA, velocity and acceleration scaling can adjust message content to prevent anomalies in vehicle movement that would otherwise occur at any playback speed other than real-time.

Generating Interim PDUs During Slow-Speed DIS Playback (Heartbeating)

In a DIS exercise, each participating entity sends PDUs to other applications at a regular time interval, called the heartbeat. A common heartbeat rate is 5 seconds.

Most receiving applications expect to receive PDUs from each entity at the agreed-upon heartbeat rate. If a receiving application does not receive a PDU from an entity for an unusually long time, it may conclude that the entity no longer exists, and then stop simulating (or time out) the entity. A typical timeout threshold is 12 seconds, or 2.4 times the heartbeat rate.

During slow-speed playback in DIS, the long time lapse between an entity's recorded PDUs might cause receiving applications to time the entity out prematurely. To prevent this, the Logger generates interim PDUs as needed to maintain each entity's heartbeat.

The Logger calculates the contents of an interim PDU by applying a dead-reckoning algorithm to an entity's most recent PDU.

By default, the Logger ensures that at least one PDU is transmitted for each entity every five seconds. You can specify a different heartbeat rate. For procedures, please see [“Specifying the Heartbeat,”](#) on page 4-22.

Scaling the Timeout Period During Slow-Speed Playback

When an entity's PDUs stop occurring in the file, the Logger continues to generate interim PDUs for the entity for the specified timeout period.

During slow-speed playback, the timeout period is extended accordingly. For example, when playing a file at half-speed, using the default timeout setting of 12 seconds, the Logger generates interim PDUs for an entity for 12 recorded seconds, or 24 real-time seconds before timing it out.

For procedures, please see [“Specifying the Timeout,”](#) on page 4-23.

Updating Entity Information Quickly After a Time Jump

During playback, you can jump to an arbitrary timepoint. After the time jump, the Logger resumes sending update messages for all entities, starting at the new timepoint. Some time may pass before a fresh update message is sent for each entity, so until updates arrive, receiving applications will reflect entity states that existed before the time jump.

DIS only

In DIS, these applications may time entities out.

After a time jump, the Logger can generate up-to-date interim messages to fill the period before the transmission of each entity's next recorded update. To create these interim updates, the Logger applies a dead-reckoning algorithm to each entity's most recent recorded message prior to the new time point.

Because additional processing is required, you will experience a delay when jumping to a new timepoint. The length of the delay depends on the number of entities processed and the size of the time jump.

Velocity and Acceleration Scaling

When the Logger plays at speeds above or below normal, it can adjust the velocity and acceleration values in entity state messages. This scaling allows the receiving applications to perform accurate dead-reckoning.

For example, if the Logger is playing at double speed, it can double the values in a message's velocity and acceleration fields. This adjustment causes a vehicle to move at double its original speed when viewed using an application such as VR-Vantage Stealth. Without this scaling, a moving vehicle would appear to be moving at its original slower speed, and would have to lurch forward with the arrival of each new state update message in order to catch up with the new location data contained in the update.

3.3. HLA Support

This section describes compatibility and support issues for running the Logger in HLA. Please refer to *MÄK Data Logger Release Notes* and any supplementary documentation for your version of the Logger for current information about FOM support.

3.3.1. FOM Flexibility

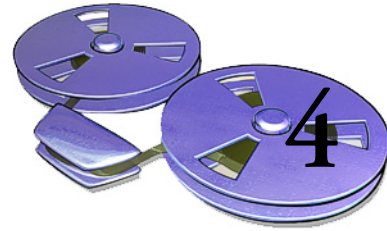
You can change the FED file between recording and playback. As long as the FED file you are using on playback contains object and interaction classes, and attributes and parameters with the same names and format as those that were recorded, you can play back your file.

Classes, attributes, and parameters can be rearranged in the FED file, and new ones can be added. If any class names have been deleted from the FED file between recording and playback, recorded interactions and updates for those classes will not be included in the Logger's outgoing data stream. If any attributes or parameter names have been deleted from the FED file, those attributes and parameters do not appear in outgoing updates and interactions.

3.3.2. Backward Compatibility

Different versions of the RPR FOM represent many attributes and parameters differently. They are essentially different protocols. Therefore, you may not be able to play back a Logger file recorded with previous versions of the Logger in the same federation execution as applications based on the current RPR FOM.

Please refer to the *MÄK Data Logger Release Notes* and any supplementary documentation for the latest release of the Logger for current information about backward compatibility.



Configuring the Logger Connection

This chapter explains how to configure connections for DIS and HLA. For general information about configuring the Logger, please see Chapter 5, *Configuring the Logger*.

Configuring the Exercise Connection	4-3
Viewing Connection Properties	4-3
Editing a Connection	4-4
Configuring the Logger for HLA	4-6
Specifying the Name of the Logger Federate	4-7
Specifying the Type of the Logger Federate (HLA Evolved Only)	4-7
Specifying the Federation Execution Name	4-8
Specifying a FED File	4-8
Specifying FOM Modules (HLA Evolved Only)	4-9
Specifying the MIM (HLA Evolved Only)	4-9
Specifying the RPR FOM Version	4-10
Specifying the RPR FOM Revision	4-10
Ignoring Advisory Messages	4-11
Specifying a FOM Mapper	4-11
Configuring Time Management	4-13
Specifying the Logger Subscription List	4-14
Subscribing Passively to Objects and Interactions	4-16
Configuring the Logger for DIS	4-17
Specifying a UDP Port	4-17
Setting the Exercise ID	4-18
Setting the Site ID	4-18
Setting the Application Number	4-18
Using Asynchronous I/O	4-19
Specifying a Destination Address for Outgoing and Incoming Data	4-19

Configuring the Logger Connection

Setting the Subnet Mask.....	4-20
Setting the Buffer Size for Sending and Receiving PDUs.....	4-20
Setting the Multicast Time to Live Value.....	4-21
Setting the Network Interface Address.....	4-21
Specifying the Heartbeat	4-22
Specifying the Timeout	4-23
Subscribing to Multicast Groups	4-24

4.1. Configuring the Exercise Connection

The Logger connects to exercises through an exercise connection. You can connect to and disconnect from an exercise. When you connect to an exercise you must specify the parameters for the protocol you are using (DIS or HLA). The connection status is displayed in the Logger status bar (Figure 4-1).

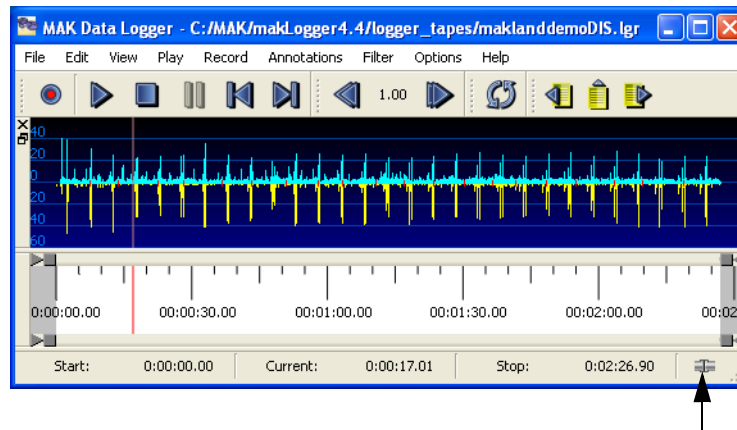


Figure 4-1. Connection status

4.1.1. Viewing Connection Properties

In the GUI, you can view the configuration of the current exercise connection for the Logger. When the Logger is connected to an exercise, the connection settings are read-only.

- To view connection properties, choose **File** → **Connection**, or right-click the connection status icon. The Connection dialog box opens (Figure 4-3).

4.1.2. Editing a Connection

If you are not connected to an exercise, you can set connection properties and connect to the network. For descriptions of the various properties, please see “[Configuring the Logger for HLA](#),” on page 4-6 and “[Configuring the Logger for DIS](#),” on page 4-17.



You cannot edit a connection while you are connected. To make a connection editable, disconnect from it.

To edit a connection:

1. Choose **File** → **Connection**. The Connection dialog box opens. It displays the current settings for the protocol you are using (HLA ([Figure 4-3](#)) or DIS ([Figure 4-3](#))).

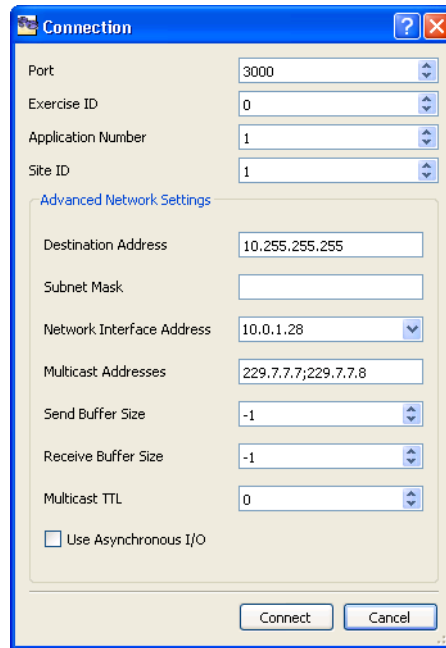


Figure 4-2. Connection dialog box for DIS

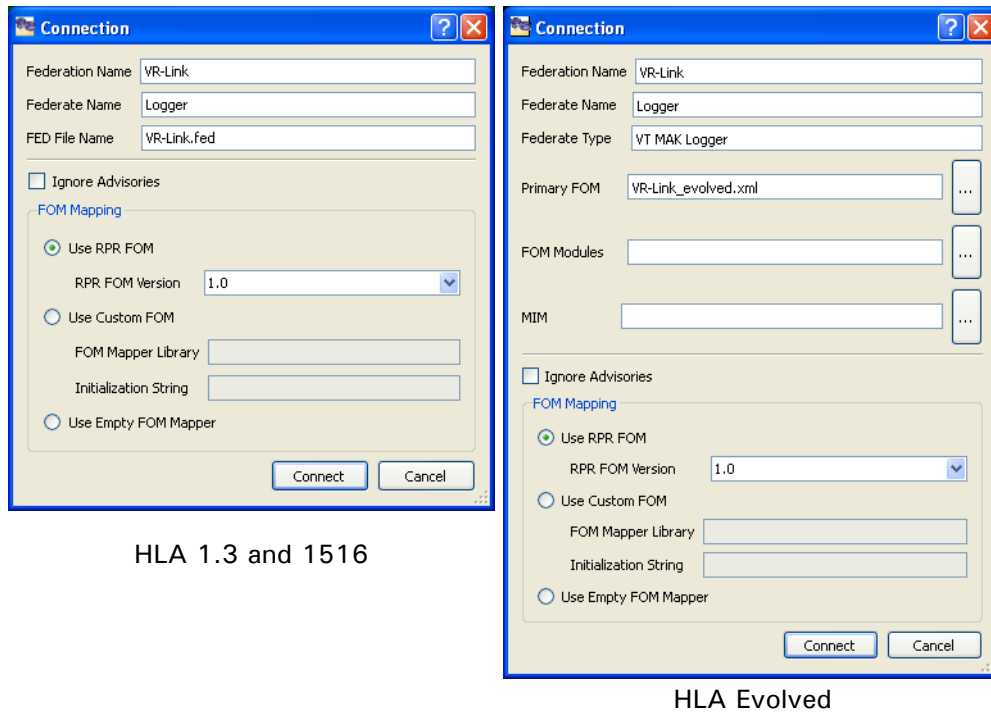


Figure 4-3. Connection dialog boxes for HLA

2. Change attribute values as desired.
3. Click Connect.

4.2. Configuring the Logger for HLA

This section describes how to configure the Logger for HLA exercises. For details about configuring connection settings in the GUI, please see [“Editing a Connection,”](#) on page 4-4.

Most of the HLA connection settings can be set using individual command-line options or by using the `SetConnectionParameters` command in the text interface.

The syntax for `SetConnectionParameters` for HLA 1.3 and HLA 1516 is:

```
SetConnectionParameters federateName federationName
    fedFile ignoreAdvisories fomVersion fomRevision
    fomMapper fomMapperData useEmptyFomMapper
```

For example:

```
SetConnectionParameters Logger VR-Link VR-Link.fed False
    1.0 1 myFomMapper "myInitParameter1 myInitParameter2 ..."
    False
```

The syntax for `SetConnectionParameters` for HLA Evolved is:

```
SetConnectionParameters federateName federateType
    federationName fedFile "fomModules" MIM ignoreAdvisories
    fomVersion fomRevision fomMapper fomMapperData
    useEmptyFomMapper
```

For example:

```
SetConnectionParameters Logger "VT MAK Logger" VR-Link VR-
    Link_evolved.xml "VR-Link_evolved_module.xml
    extendF18Module.xml" HLAstandardMIM.xml False 1.0 1
    myFomMapper "myInitParameter1 myInitParameter2 ..." False
```

The `fomMapperData` argument can be a list. If so, you must wrap it in quotes.

When you use the `SetConnectionParameters` command, you must specify all of the arguments, whether you want to change them from the default values or not. For example, if you just want to change the name of the Logger federate, even though it is the first argument in the list, you still must enter a complete command as follows:

```
SetConnectionParameters myLogger VR-Link VR-Link.fed ""
    1.0 1 "" "" ""
```

For details about commands and command syntax, please see Chapter 11, [Using Logger Commands](#).

4.2.1. Specifying the Name of the Logger Federate

By default, the Logger federate is named `Logger`.

- To specify the federate name, use the `-N` command-line option or the `SetConnectionParameters` command.

Table 4-1: Specifying the Logger federate

Interface	Command
GUI	Connection dialog box: Federate Name attribute.
Command line	<code>logger -N myLogger</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>federateName</code> variable is in the default configuration file.

4.2.2. Specifying the Type of the Logger Federate (HLA Evolved Only)

If you are using HLA Evolved, you must specify the federate type. By default, the Logger federate type is “VT MAK Logger”.

- To specify the federate name, use the `--federateType` command-line option or the `SetConnectionParameters` command.

Table 4-2: Specifying the Logger federate

Interface	Command
GUI	Connection dialog box: Federate Type attribute.
Command line	<code>logger --federateType "my custom Logger"</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>federateType</code> variable is in the default configuration file.

4.2.3. Specifying the Federation Execution Name

By default, the Logger uses the federation execution name VR-Link.



By default, the Logger assumes that the federation configuration file (FED, FDD) file has the same name as the federation execution, plus the appropriate extension. For details, please see [“Specifying a FED File,”](#) on page 4-8.

- To specify a federation execution name, use the `-x` command-line option or the `SetConnectionParameters` command.

Table 4-3: Specifying the federation execution

Interface	Command
GUI	Connection dialog box: Federation Name attribute.
Command line	<code>logger -x VR-Link</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>execName</code> variable is in the default configuration file.

4.2.4. Specifying a FED File

By default the Logger uses a FED file called *VR-Link.fed*. The Logger also accepts federation description files with the *.fdd* and *.xml* extensions. MÄK products come with a FED (and XML) file for each version of the RPR FOM that they support.



If you are using HLA Evolved, the FED (FDD) file defines the primary FOM, to which FOM modules may be added.

- To specify a FED file, use the `-F` command-line option or the `SetConnectionParameters` command.

Table 4-4: Specifying the FED file

Interface	Command
GUI	Connection dialog box: FED File Name attribute.
Command line	<code>logger -F VR-Link20017-1.fed</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fedFileName</code> variable is in the default configuration file.

4.2.5. Specifying FOM Modules (HLA Evolved Only)

If you are using HLA Evolved, you can specify FOM modules. FOM modules get added to the primary FOM, which is specified by the FED file. By default the Logger does not use additional FOM Modules. The Logger also accepts FOM Module files with the *.xml* extension. MÄK products come with a few FOM modules that define the VR-Link specific classes that are used by our products.

- To specify a FED file, use the `--fomModules` command-line option or the `SetConnectionParameters` command.

Table 4-5: Specifying the FED file

Interface	Command
GUI	Connection dialog box: FOM Modules attribute.
Command line	<code>logger --fomModules "module1.xml module2.xml ... moduleN.xml"</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fomModules</code> variable is in the default configuration file.

4.2.6. Specifying the MIM (HLA Evolved Only)

If you are using HLA Evolved, you can specify a MOM and Initialization Module (MIM). By default the Logger does not use a MIM. The Logger accepts MIM files with the *.xml* extension. MÄK products come with the standard MIM.

- To specify a FED file, use the `--MIM` command-line option or the `SetConnectionParameters` command.

Table 4-6: Specifying the FED file

Interface	Command
GUI	Connection dialog box: MIM attribute.
Command line	<code>logger --MIM HLAstandardMIM.xml</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>MIM</code> variable is in the default configuration file.

4.2.7. Specifying the RPR FOM Version

The Logger has built-in support for the RPR FOM. The default version is 1.0. You can specify a different RPR FOM version (0.5, 0.7, 0.8, 2.0006, 2.0014, or 2.0017). If you specify a RPR FOM version other than the default, you must also specify the correct FED file.



Please see *MÄK Data Logger Release Notes* for a list of the RPR FOM versions supported by your version of the Logger.

- To specify the RPR FOM version, use the `--rprFomVersion` command-line option or the `SetConnectionParameters` command.

Table 4-7: Specifying the RPR FOM version

Interface	Command
GUI	Connection dialog box: Use RPR FOM option and RPR FOM Version attribute.
Command line	<code>logger --rprFomVersion 2.0017 -F VR-Link20017-1.fed</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>rprFomVersion</code> variable is in the default configuration file.

4.2.8. Specifying the RPR FOM Revision

If you use RPR FOM version 2, draft 17, you can specify a RPR FOM revision (1, 2, or 3). These revision numbers only apply to the RPR FOM 2.0017. The FED file can be the same for the different revisions; only the data format changes. Default: 1.

- To specify the RPR FOM revision, use the `--rprFomRevision` command-line option or the `SetConnectionParameters` command.

Table 4-8: Specifying the RPR FOM version

Interface	Command
GUI	Connection dialog box: Use RPR FOM option and RPR FOM Revision attribute.
Command line	<code>logger --rprFomVersion 2.0017 --rprFomRevision 1 -F VR-Link20017-1.fed</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>rprFomRevision</code> variable is in the default configuration file.

4.2.9. Ignoring Advisory Messages

- To have the Logger ignore advisory messages, use the `SetIgnoreAdvisories` command.

Table 4-9: Specifying the response to advisory messages

Interface	Command
GUI	Connection dialog box: Select Ignore Advisories.
Command line	<code>logger --ignoreAdvisories 1</code>
Text interface	Use the <code>SetConnectionParameters</code> command, or use <code>SetIgnoreAdvisories true</code>
<i>lgrConfig.xml</i>	The <code>ignoreAdvisories</code> variable is in the default configuration file. You can also add <code>SetIgnoreAdvisories</code> to the configuration file.

4.2.10. Specifying a FOM Mapper

The Logger supports different FOMs using VR-Link FOM Mappers. The Logger chooses the FOM Mapper based on the following criteria:

- If a FOM mapper library is specified, it is used.
- If Use Empty FOM Mapper is specified, the empty FOM Mapper is used.
- If neither of the first two options is specified, a default RPR FOM FOM Mapper is used.



The different versions of the RPR FOM that you can specify with the `rprFomVersion` parameter are actually designating one of several default FOM Mappers supplied with the Logger.

- To specify a FOM Mapper, use the `-f` command-line option or the `SetConnectionParameters` command.

Table 4-10: Specifying a FOM Mapper

Interface	Command
GUI	Connection dialog box: Select Use Custom FOM and specify FOM Mapper Library and, optionally, Initialization String. Select Use Empty FOM Mapper if the FOM you are using is not a derivative of the RPR FOM and no appropriate FOM Mapper library is available.
Command line	<code>logger -f myFomMapper.dll</code> or <code>logger --useEmptyFomMapper 1</code>

Table 4-10: Specifying a FOM Mapper

Interface	Command
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fomMapperLib</code> and <code>useEmptyFomMapper</code> variables are in the default configuration file.

Specifying a FED File that is Appropriate for the FOM Mapper



The `-f` option only tells the application which FOM Mapper to use. It does not ensure that you are using the correct FED file for the FOM Mapper. Make sure that you are using a FED file that is consistent with the FOM Mapper you have chosen.

By default the Logger uses a FED file called *VR-Link.fed*, but you can specify an alternate FED file using the `-F` option.

The following example instructs the application to use an alternative FOM Mapper and FED file:

```
logger -f myFomMapper -F myFedFile.fed
```

Specifying FOM Mapper Initialization Data

Some FOM Mappers require initialization data.

- To specify FOM Mapper initialization data, use the `--fomMapperInitData` command-line option or the `SetConnectionParameters` command.

Table 4-11: Specifying FOM Mapper initialization data

Interface	Command
GUI	Connection dialog box: Set Use FOM Mapper to True, specify a FOM Mapper File, and set the FOM Mapper Init Data attribute.
Command line	<code>logger -f myFomMapper.dll --fomMapperInitData ABC</code> where ABC is passed to the FOM Mapper creator function in the shared library as an initialization parameter.
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fomMapperInitData</code> variable is in the default configuration file.

4.2.11. Configuring Time Management

The Logger supports time-constrained recording and time-regulating playback. To configure time management in the Logger, it must be enabled in your RTI. If time management is not enabled in the RTI, the configuration options are not available in the Preferences dialog box.

To configure time management in the GUI:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens (Figure 4-4).

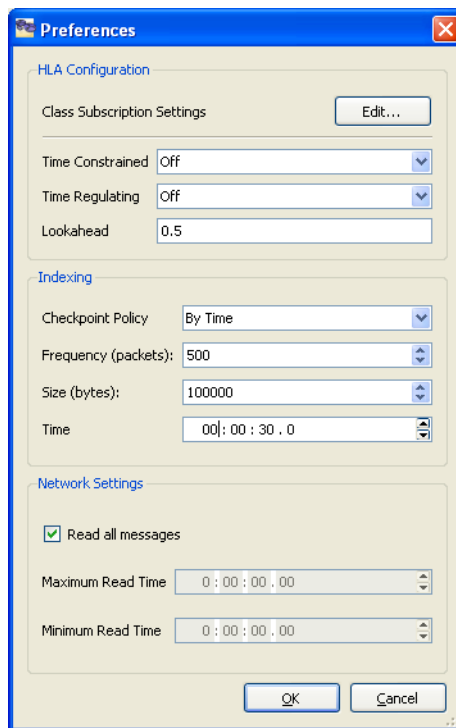


Figure 4-4. Preferences dialog box for HLA

2. To configure the Logger as a time-constrained federate, in the HLA Configuration group box, set the Time Constrained value to Recording.
 3. To configure the Logger as a time-regulating federate, in the HLA Configuration group box, set the Time Regulating value to Playback.
 4. Optionally, set the Lookahead value.
 5. Click OK.
- To specify time management from a command interface, use the `SetTimeManagement` command.

Table 4-12: Specifying time management

Interface	Command
GUI	See procedure in this section.
Command line	<code>logger -r "SetTimeManagement true false false true 0.5"</code>
Text interface	<code>SetTimeManagement true false false true 0.5</code>
<i>lgrConfig.xml</i>	Add <code>SetTimeManagement</code> to the configuration file.

4.2.12. Specifying the Logger Subscription List

By default the Logger subscribes to all classes in your FED file except *RTIPrivate* and any MOM object or interaction. However, you can specify the set of classes and interactions that the Logger subscribes to by listing them in a command or creating a subscription control list.

Creating a Subscription Control List

In the Logger GUI, you can specify a list of classes to subscribe to and you can save that list to a file.

To create a subscription control list in the GUI:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens (Figure 4-4).
2. In the HLA Configuration group box, click Edit. The Change HLA Class Subscription dialog box opens (Figure 4-5). The dialog box has a tab that lists object classes and a tab that lists interaction classes. Classes that have a check mark next to their name are filtered out.

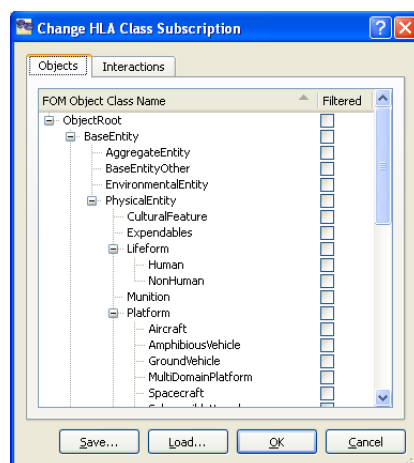


Figure 4-5. Change HLA Class Subscription dialog box

3. Edit the subscription list by checking the box in the Filtered column for classes that you want to filter out.
 4. Click OK. The subscription list takes effect the next time you enter record mode.
- To specify a subscription list in a command interface, use the `SetSubscriptionLists` command.

Table 4-13: Specifying response to advisory messages

Interface	Command
Command line	<code>logger -r "SetSubscriptionLists \"ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft ObjectRoot.BaseEntity.PhysicalEntity.Radio InteractionRoot.WeaponFire\""</code>
Text interface	<code>SetSubscriptionLists "ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft ObjectRoot.BaseEntity.PhysicalEntity.Radio InteractionRoot.WeaponFire"</code>
<i>lgrConfig.xml</i>	Add <code>SetSubscriptionLists</code> to the configuration file. For details about adding a string list to the configuration file, please see "Adding a String List to lgrConfig.xml," on page 5-22.
	 The <code>SetSubscriptionLists</code> command must come after the <code>Connect</code> command.

Saving A Subscription List

To save a subscription list in the GUI:

1. Edit a subscription list as described in ["Specifying the Logger Subscription List,"](#) on page 4-14.
 2. In the Change HLA Class Subscription dialog box, click Save. The Save As dialog box opens.
 3. Enter a name for the subscription list and click Save. By default, subscription lists are saved in the appropriate *bin* directory with the extension *.subx*.
- To save a subscription list in a command interface, use the `SaveSubscriptionListFile` command.

Table 4-14: Saving a subscription list

Interface	Command
Command line	<code>logger -r "SaveSubscriptionListFile mySubList.subx"</code>
Text interface	<code>SaveSubscriptionListFile mySubList.subx</code>

Loading A Subscription List

To load a subscription list in the GUI:

1. In the Change HLA Class Subscription dialog box, click Load. An Open dialog box opens.
 2. Select the subscription list file you want to load.
 3. Click Open.
 4. Click OK to apply the subscription list.
- To load a subscription list in a command interface, use the `LoadSubscriptionListFile` command.

Table 4-15: Loading a subscription list

Interface	Command
Command line	<code>logger -r "LoadSubscriptionListFile \"../config/mySubList.subx\""</code>
Text interface	<code>LoadSubscriptionListFile ../config/mySubList.subx</code>
<i>IgrConfig.xml</i>	Add <code>LoadSubscriptionListFile</code> to the configuration file.

4.2.13. Subscribing Passively to Objects and Interactions

The Logger can subscribe passively to objects and interactions. In passive mode, the Logger records the updates that other federates subscribe to. It does not subscribe to any specific objects or interactions. If you want to use passive subscription, you must specify it before you connect to an exercise.

- To have the Logger subscribe passively, use the `SetPassiveSubscription` command.

Table 4-16: Specifying passive subscription

Interface	Command
Command line	<code>logger -r "SetPassiveSubscription true"</code>
Text interface	<code>SetPassiveSubscription true</code>
<i>IgrConfig.xml</i>	Add <code>SetPassiveSubscription</code> to the configuration file.

4.3. Configuring the Logger for DIS

This section describes how to configure the Logger for DIS exercises.

Most of the DIS connection settings can be set using individual command-line options or by using the `SetConnectionParameters` command in the text interface. The syntax for this command is:

```
SetConnectionParameters port exerciseId siteId appNumber
                        useAsynchIO sendBufferSize recvBufferSize multicastTtl
                        destAddr subnetMask deviceAddr mcastAddresses
```

For example:

```
SetConnectionParameters 3000 1 1 2 True -1 -1 -1
10.255.255.255 "" 10.0.1.57 "225.0.0.118 225.0.0.202"
```

When you use the `SetConnectionParameters` command, you must specify all of the arguments, whether you want to change them from the default values or not. For example, if you just want to change the port, even though it is the first argument in the list, you still must enter a complete command as follows:

```
SetConnectionParameters 3000 1 1 2 True -1 -1 -1
10.255.255.255 "" 10.0.1.57 "225.0.0.118 225.0.0.202"
```



The Logger supports IPv4 and IPv6. For any particular connection, all addresses must be of the same type.

For details about commands and command syntax, please see Chapter 11, [Using Logger Commands](#).

4.3.1. Specifying a UDP Port

By default, the Logger sends and receives data over UDP port 3000.

- To specify a UDP port, use the `-P` command-line option or the `SetConnectionParameters` command.

Table 4-17: Specifying the UDP port

Interface	Command
GUI	Connection dialog box: Port attribute.
Command line	<code>logger -P 2345</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>disPort</code> variable is in the default configuration file.

4.3.2. Setting the Exercise ID

- To specify the exercise ID, use the `-e` command-line option or the `SetConnectionParameters` command.

Table 4-18: Specifying the exercise ID

Interface	Command
GUI	Connection dialog box: Exercise ID attribute.
Command line	<code>logger -e 3</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>exerciseld</code> variable is in the default configuration file.

4.3.3. Setting the Site ID

- To specify the site ID, use the `-s` command-line option or the `SetConnectionParameters` command.

Table 4-19: Specifying the site ID

Interface	Command
GUI	Connection dialog box: Site ID attribute.
Command line	<code>logger -s 3</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>siteid</code> variable is in the default configuration file.

4.3.4. Setting the Application Number

- To specify the application number, use the `-a` command-line option or the `SetConnectionParameters` command.

Table 4-20: Specifying the application number

Interface	Command
GUI	Connection dialog box: Application Number attribute.
Command line	<code>logger -a 2</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>appNumber</code> variable is in the default configuration file.

4.3.5. Using Asynchronous I/O

You can configure the Logger to use asynchronous I/O. Use of asynchronous I/O can help reduce dropped packets during times of high traffic.

- To specify use of asynchronous I/O, use the `SetConnectionParameters` command.

Table 4-21: Specifying use of asynchronous I/O

Interface	Command
GUI	Connection dialog box: Use Asynchronous IO attribute.
Command line	Use the <code>SetConnectionParameters</code> command.
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>useAsyncIO</code> variable is in the default configuration file.

4.3.6. Specifying a Destination Address for Outgoing and Incoming Data

Normally, the Logger sends outgoing, and receives incoming, data packets using the host machine's default broadcast address. The specified destination address can be a broadcast, multicast, or unicast address.

- To specify a destination address, use the `-A` command-line option or the `SetConnectionParameters` command.

Table 4-22: Specifying the destination address

Interface	Command
GUI	Connection dialog box: Destination Address attribute.
Command line	<code>logger -A 205.205.205.205</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>destAddrString</code> variable is in the default configuration file.

4.3.7. Setting the Subnet Mask

When the destination address is not specified ("0.0.0.0"), the subnet mask is used in conjunction with the network interface address to determine the destination broadcast address. If neither is specified (subnet mask is blank), the network interface's default broadcast address is used.

- To specify a subnet mask, use `SetConnectionParameters` command.

By default, the logger will use the subnet mask associated with the IP class.

Table 4-23: Specifying the subnet mask

Interface	Command
GUI	Connection dialog box: Subnet Mask attribute.
Command line	Use the <code>SetConnectionParameters</code> command.
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>subnetMask</code> variable is in the default configuration file.

4.3.8. Setting the Buffer Size for Sending and Receiving PDUs

You can specify the size of the buffers used for sending and receiving DIS packets. A larger receive buffer reduces the likelihood of dropping packets. Buffer size is specified in bytes. A value of -1 means use the system default value.

- To specify send and receive buffer sizes, use the `--receiveBufferSize` and `--sendBufferSize` command-line options or the `SetConnectionParameters` command.

Table 4-24: Specifying the send and receive buffers

Interface	Command
GUI	Connection dialog box: Send Buffer Size and Receive Buffer Size attributes.
Command line	<code>logger --receiveBufferSize 64000 --sendBufferSize 64000</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>sendBufferSize</code> and <code>recvBufferSize</code> variables are in the default configuration file.

4.3.9. Setting the Multicast Time to Live Value

The multicast time to live value specifies the number of router jumps a message can take.

- To specify the multicast time-to-live value, use the `--mcastTtl` command-line option or the `SetConnectionParameters` command.

Table 4-25: Specifying the multicast time-to-live

Interface	Command
GUI	Connection dialog box: Multicast TTL attribute.
Command line	<code>logger --mcastTtl 3</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>IgrConfig.xml</i>	The <code>mcastTtl</code> variable is in the default configuration file.

4.3.10. Setting the Network Interface Address

The network interface address specifies the address to use when a computer has multiple network cards.

- To specify the network interface address, use the `--deviceAddrString` command-line option or the `SetConnectionParameters` command.

Table 4-26: Specifying the multicast time-to-live

Interface	Command
GUI	Connection dialog box: Network Interface Address.
Command line	<code>logger --deviceAddrString string</code>
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>IgrConfig.xml</i>	The <code>deviceAddrString</code> variable is in the default configuration file.

4.3.11. Specifying the Heartbeat

The heartbeat value determines how frequently (in seconds) the Logger generates interim PDUs for each entity during slow-speed playback. For more information, please see [“Generating Interim PDUs During Slow-Speed DIS Playback \(Heartbeating\),”](#) on page 3-6.

To set the heartbeat in the GUI:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens ([Figure 4-6](#)).

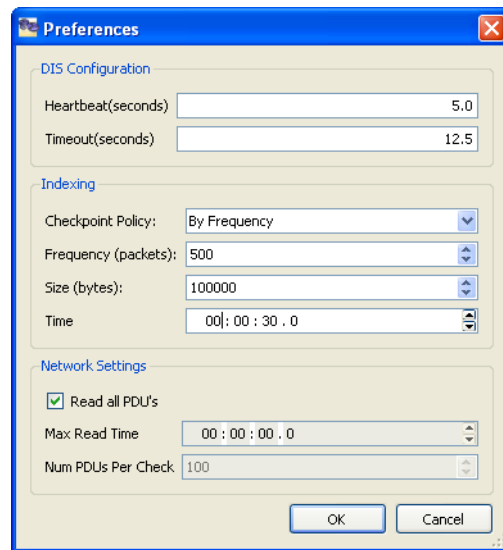


Figure 4-6. Preferences dialog box for DIS

2. In the DIS Configuration group box, type a value in the Heartbeat text box.
 - To specify the heartbeat from the command interface, use the `SetHeartbeatRate` command.

Table 4-27: Specifying the heartbeat

Interface	Command
Command line	<code>logger -r "SetHeartbeatRate 5.0"</code>
Text interface	<code>SetHeartbeatRate 5.0</code>
<i>lgrConfig.xml</i>	Add <code>SetHeartbeatRate</code> to the configuration file.

4.3.12. Specifying the Timeout

The timeout value determines how long the Logger continues to generate interim PDUs for an entity once that entity's PDUs have stopped occurring in the file. For more information, please see [“Generating Interim PDUs During Slow-Speed DIS Playback \(Heartbeating\),”](#) on page 3-6.

To set the timeout in the GUI:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens ([Figure 4-6](#)).
 2. Type a value in the Timeout text box.
- To specify the DIS timeout from the command interface, use the `SetTimeoutRate` command.

Table 4-28: Specifying the timeout

Interface	Command
Command line	<code>logger -r "SetTimeoutRate 12.0"</code>
Text interface	<code>SetTimeoutRate 12.0</code>
<i>lgrConfig.xml</i>	Add <code>SetTimeoutRate</code> to the configuration file.

4.3.13. Subscribing to Multicast Groups

In record mode, the Logger can subscribe to multicast groups. When the Logger is subscribed to multicast groups, it also continues to receive data through the machine's broadcast and unicast addresses.

Subscription applies to record mode only.

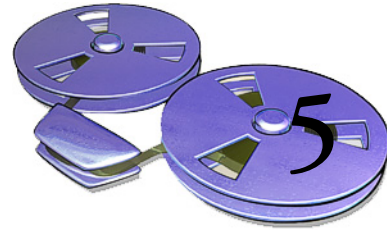
i

Multicast subscription is a property of the host machine, not of individual applications. If several applications are running on the same machine using the same port, and those applications subscribe to multicast groups, all the applications will receive packets for all multicast groups to which any application is subscribed. Therefore, if the Logger is using the same port as applications that subscribe to multicast groups, the Logger may receive traffic intended for the other applications.

- To subscribe to multicast groups, use the `-S` command-line option or the `SetConnectionParameters` command.

Table 4-29: Subscribing to multicast groups

Interface	Command
GUI	Connection dialog box: Multicast Addresses attribute.
Command line	<code>logger -S "225.0.0.118 225.0.0.202"</code>
Text interface	<code>SetConnectionParameters port exerciseId siteId appNumber useAsynchIO sendBufferSize recvBufferSize multicastTtl destAddr deviceAddr "mcastAddresses"</code>
<i>lgrConfig.xml</i>	The <code>mcastAddresses</code> variable is in the default configuration file.



Configuring the Logger

This chapter explains how to configure various Logger features and how to edit the *lgrConfig.xml* file.

Configuring the Logger.....	5-2
Configuring Indexing	5-2
Configuring Indexing Using the Command Interfaces	5-3
Configuring Faster Playback	5-4
The SetEntityTracking Command.....	5-5
The SetFastJumpsWithoutDiff Command	5-5
The SetPlaybackReadTimeLimit Command.....	5-6
Configuring HLA Heartbeating	5-7
Configuring HLA Timeouts.....	5-7
Configuring Network Read Time.....	5-9
Using Data Distribution Management.....	5-10
Specifying Subscription Regions	5-10
Using Conveyed Regions.....	5-13
Using Simulation Data	5-13
Configuring DDM.....	5-14
Configuring DDM in lgrConfig.xml.....	5-16
Setting the Notification Level.....	5-17
Specifying the Log File	5-17
Editing lgrConfig.xml	5-18
Adding a Command to lgrConfig.xml	5-21
Summary of Command-line Options for the Logger.....	5-23

5.1. Configuring the Logger

You can configure the Logger by editing its configuration file, *lgrConfig.xml*, with command-line options, in the text interface, and in the GUI. With the exception of some of the command-line options, all of the ways that you can configure the Logger are essentially alternate ways of implementing Logger commands. For a complete list of command-line options, please see “[Summary of Command-line Options for the Logger](#),” on page 5-23 or the Quick Reference Card. For descriptions of Logger commands and how to use them, please see Chapter 11, *Using Logger Commands* or the Quick Reference Card. For details about the configuration file, please see “[Editing lgrConfig.xml](#),” on page 5-18.

This chapter describes the configuration options and shows how to implement them in the various interfaces.

5.2. Configuring Indexing

Logger files contain index data (checkpoints) that allow the Logger to quickly and efficiently jump from one time point to another.

When you jump the Logger to a time point, it jumps to the nearest checkpoint before the selected time point and then processes the packets in the file to the specified time point. (The packets are not sent over the network.) Therefore, the efficiency of time jumps is determined by the frequency of checkpoints. Having more frequent checkpoints speeds up time jumps. However, more checkpoints increases file size and can slow down recording, particularly if the exercise has many entities. So when you specify your checkpoint policy, you must balance your priorities.

When you record, indexing is always enabled. You can set the checkpoint policy (the criteria for how often the Logger adds index data) to one of the following:

- ♦ By Frequency, in packets received (default)
- ♦ By Size, in bytes
- ♦ By Time.



If you are currently recording an exercise, changing the preferences does not affect the recording process. It takes effect for the next new file.

To configure indexing in the GUI:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens.

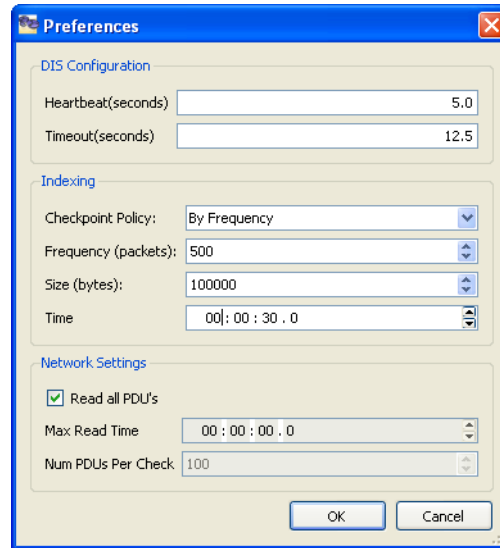


Figure 5-1. Preferences dialog box (DIS)

2. In the Indexing group box, select the checkpoint policy.
3. Edit the appropriate value for the checkpoint policy you chose (frequency, size, or time).
4. Click OK.

5.2.1. Configuring Indexing Using the Command Interfaces

- To configure indexing using a command interface, issue the `SetCheckpointParameters` command.

Table 5-1: Checkpointing command

Interface	Command
Command line	<code>logger -r "SetCheckpointParameters 0 0 500 0"</code> For details about the parameter values, please see "Recorder Domain Commands," on page 11-8.
Text interface	<code>SetCheckpointParameters 0 0 500 0</code>
<i>lgrConfig.xml</i>	Add <code>SetCheckpointParameters</code> to the configuration file.

5.3. Configuring Faster Playback

There are some processes that you can modify so that the Logger can play back files more quickly at the expense of some data or feature loss. In certain cases, the data loss is acceptable and the improved performance is required because of the density of data or packets in the tape. For example, in DIS and some HLA federations, there are entity heartbeats and timeouts. So if you drop some entity state information or miss the removal of an entity, that information can be retrieved by subsequent state updates or entity timeouts.

The loss of data is likely to have a greater effect on HLA federations, which usually do not have an entity heartbeat mechanism. In HLA the object management services inform federates about object discoveries and removals and there are services for requesting missing data. As a result, if the Logger skips data, there is no certainty that future updates will provide the missing data. More importantly, since in HLA the Logger only removes entities when it processes a removal packet, if one is skipped, an object that should have been removed will erroneously continue to persist in the federation.

In both DIS and HLA, all skipped interactions are lost. Since interactions are normally transient events, they can often be missed without great impact. However, if the exchange of interactions contains some state information, for example, transitions in a transaction protocol such as DIS logistics, then a missed interaction may leave the transaction in a bad state.

The Logger uses the following commands to configure fast playback:

- ♦ `SetEntityTracking`
- ♦ `SetFastJumpsWithoutDiff`
- ♦ `SetPlaybackReadTimeLimit.`

In conjunction with fast playback, in HLA the following commands let you configure the Logger to perform entity heartbeats and timeouts and support federations that have these requirements:

- ♦ `SetHeartbeatIntervals`
- ♦ `SetTimeoutIntervals.`

5.3.1. The SetEntityTracking Command

The `SetEntityTracking` command enables or disables the entity tracking process during playback. Entity tracking keeps track of the entities in a logger file and supplies the list of entities for the entity filter. The entity tracker is periodically updated. Eliminating this overhead speeds up playback. However, entity filtering depends on the entity tracker information and no entities are displayed when entity tracking is disabled. Default: entity tracking is enabled.

The syntax for this command in the text interface is:

```
SetEntityTracking isEnabledValue
```

For example:

```
SetEntityTracking false
```

The syntax for using this command from the command-line is:

```
logger -r "SetEntityTracking isEnabledValue"
```

For example:

```
logger -r "SetEntityTracking false"
```

5.3.2. The SetFastJumpsWithoutDiff Command

The `SetFastJumpsWithoutDiff` command changes how the Logger jumps to a particular time in the file during playback. The default behavior is to first get the nearest checkpoint data. This establishes the initial conditions of the jump (that is, current entities and their state). The Logger then processes the packets from the checkpoint time up to the jump time; however, the output is disabled. The result is that the internal object state is updated, but no data is sent to the network. This two step process establishes the current state of the entities at the selected jump time. When `SetFastJumpsWithoutDiff` is enabled, the Logger jumps immediately from the checkpoint to the jump time without processing intermediate packets. Playback in DIS can handle this fast jump because of entity state updates and entity timeouts. Default: fast jumps are disabled.

The syntax for this command in the text interface is:

```
SetFastJumpsWithoutDiff isEnabledValue
```

For example:

```
SetFastJumpsWithoutDiff true
```

The syntax for using this command from the command-line is:

```
logger -r "SetFastJumpsWithoutDiff isEnabledValue"
```

For example:

```
logger -r "SetFastJumpsWithoutDiff true"
```

5.3.3. The SetPlaybackReadTimeLimit Command

The `SetPlaybackReadTimeLimit` command limits the time the Logger can read packets from a file and drops the remaining packets up to the current time. The Logger records each packet with a relative timestamp from the beginning of the file. When playing back a tape, the Logger reads all of the packets that have timestamps less than or equal to the current time. There is no time limit for the length of this read operation. It is possible that the density of a tape may be such that the Logger takes too long to read all of the packets. The result is that the Logger begins to lag behind the selected playback speed.

A simple example of how this can happen is if the Logger playback speed is increased such that the Logger cannot read all the packets and maintain the selected speed. This lagging behavior could also happen if a very fast machine records the tape and a slow machine is used to play it back or a lightly loaded and then heavily loaded machine. In any case, if the file or playback speed is consistently too dense for the Logger to play back, it falls farther and farther behind.

Setting a read time limit causes the Logger to cut off the read time if needed to maintain the selected speed. However, like the fast jump, the Logger discards data and it should only be employed when dropped data is not a concern or can be recovered by subsequent updates. This trade-off is a bit more drastic in that the Logger may continually drop packets, and while unlikely, the distribution of the packets in the file may be such that some entities may get updated sporadically or not at all. The default configuration disables the read time limit and allows the Logger to take whatever time necessary to read all the packets up to the current time.

When you set the read time limit, it is important to consider the tick rate. The Logger uses the tick rate (really a tick interval) to periodically schedule Logger activity. The Logger starts a read operation (which includes playback of those packets) and when complete, it sleeps a period of time. The sleep allows other processes to run, like the Logger GUI. The sleep interval is established by the tick rate minus the time required to process the read operation. In this way, the Logger performs reads at regular intervals. A candidate value for the read time limit is a value slightly less than the tick rate. In this way, the Logger is assured of completing the read operation before the next scheduled tick time.

This command takes a floating point argument. A value less than or equal to zero disables the behavior. The syntax for this command in the text interface is:

```
SetPlaybackReadTimeLimit timeLimitValue
```

For example:

```
SetPlaybackReadTimeLimit 0.2
```

The syntax for using this command from the command-line is:

```
logger -r "SetPlaybackReadTimeLimit timeLimitValue"
```

For example:

```
logger -r "SetPlaybackReadTimeLimit 0.2"
```

5.3.4. Configuring HLA Heartbeating

HLA entity heartbeating sends a complete update of all the object attributes at a specified interval. The Logger will perform dead-reckoning for entity and aggregate publishers if it is using the internal RPR FOM Mapper or a FOM Mapper library that maps to these publishers. If not, the last known values are used.

HLA heartbeating is configured with the `SetHeartbeatIntervals` command. `SetHeartbeatIntervals` takes as an argument a space-separated set of class heartbeat intervals. Each interval is specified in the form "`className:interval:inheritable`", for instance "ObjectRoot:5.0:true".

The Logger sends a heartbeat for each object as specified by its class interval parameter. If an interval is not specified for an object class, the Logger uses the closest parent class for which there is an interval and the class specified inheritable as true. For example, if all classes are a subclass of ObjectRoot and you want all classes to heartbeat every 5.0 seconds, "ObjectRoot:5.0:true" configures this without requiring every class to be individually listed.

If an object's class specifies a zero interval or the interval is not specified and there is no inheritable interval available, objects of that class do not heartbeat.

The syntax for this command in the text interface is:

```
SetHeartbeatIntervals "className:interval:inheritable ..."
```

You can specify as many classes as needed. Each specification must be separated by a space. For example:

```
SetHeartbeatIntervals "ObjectRoot:5.0:true Object-
Root.BaseEntity.PhysicalEntity:4.0:true"
```

5.3.5. Configuring HLA Timeouts

You can configure the HLA Logger to timeout entities. When recording, if the Logger does not receive an update from a known entity within its timeout interval, it acts as if the entity has been removed. The timeout interval is conventionally the heartbeat interval multiplied by 2.5.

To configure HLA timeouts use the `SetTimeoutIntervals` command. This command takes as an argument a space-separated set of class timeout intervals. Each interval is specified in the form "`className:interval:inheritable`", for instance "ObjectRoot:12.5:true".

The Logger removes known objects based on the object's class timeout parameter. If a timeout is not specified for an object class, the Logger uses the closest parent class for which there is a timeout and the class specified inheritable as true. For example, if all classes are a subclass of ObjectRoot and you want all classes to timeout after 12.5 seconds, "ObjectRoot:12.5:true" specifies this without requiring every class to be individually listed.

If an object's class specifies a zero timeout or the timeout is not specified and there is no inheritable timeout available, objects of that class do not timeout.

The syntax for this command in the text interface is:

```
SetTimeoutIntervals "className:interval:inheritable ..."
```

You can specify as many classes as needed. Each specification must be separated by a space. For example:

```
SetTimeoutIntervals "ObjectRoot:12.5:true Object-  
Root.BaseEntity.PhysicalEntity:15.0:true"
```

5.4. Configuring Network Read Time

You can specify how much time to spend reading messages from the network. Setting a maximum time can prevent the Logger from becoming unresponsive when it is flooded with messages. The options for Network Settings are as follows:

- Read All Messages (HLA) or Read All PDUs (DIS). When selected, during each tick the Logger reads all messages in the queue.
- Max Read Time. Specifies the maximum time to spend reading messages each tick.
- Min Read Time (HLA only). Specifies the minimum amount of time to spend reading messages each tick.

Num PDUs Per Check (DIS only). Specifies the maximum number of PDUs to read before checking to see if the Max Read Time has been reached.



Max Read Time, Min Read Time, and Num PDUs Per Check are not available when Read All Messages is selected.

To configure network read settings:

1. Choose **Options** → **Preferences**. The Preferences dialog box opens (Figure 5-2).

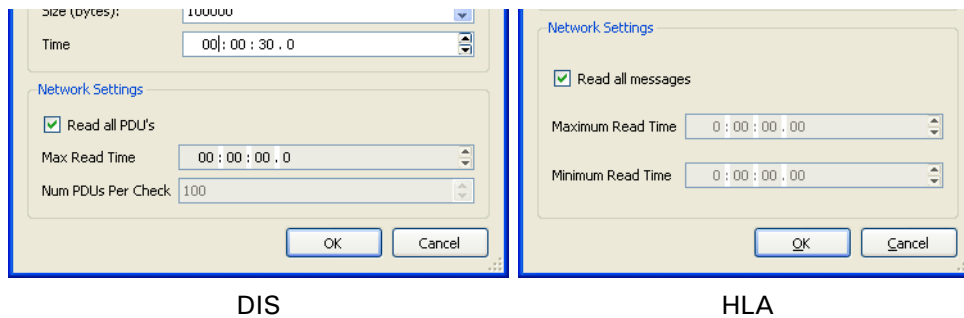


Figure 5-2. Preferences dialog box

2. To read all messages in each tick, select Read All Messages (for HLA; for DIS, select Read All PDUs.)
3. To specify the maximum amount of time to read messages, clear Read All Messages and type a value for Max Read Time.
4. To specify the minimum amount of time to read messages (HLA only), clear Read All Messages and type a value for Min Read Time.
To specify the maximum number of PDUs to read before checking Max Read Time (DIS only), clear Read All Messages and type a value for Num PDUs Per Check.
5. Click OK.

5.5. Using Data Distribution Management

The Logger can record and play back using HLA Data Distribution Management (DDM). There are two general use cases for configuring DDM with the Logger. The first case uses the HLA DDM conveyed region information to record and play back using DDM regions. The second case supports creating a plug-in to inspect the simulation data to create and update regions. The DDM configuration parameters control which of these approaches is supported and they allow some variation in how they are supported.

In both cases, you can configure the Logger to use either global region subscriptions (*subscribeMode* = **SubscribeRegionsAreGlobal**) or specific region subscriptions (*subscribeMode* = **SubscribeRegionsAreSpecified**). When the Logger subscribes using global regions, it uses an appropriate region with global extents for each class subscription to receive all possible data from the federation. This subscription mode requires no further configuration other than setting the *subscribeMode* parameter in the **SetDdmParameters** command.

You can configure the Logger with specific region subscriptions to restrict the data that the Logger records, or segregate the data among several Loggers, each specifying different region extents. This subscription mode requires a subscription specification file, as described in [“Specifying Subscription Regions,”](#) on page 5-10.

5.5.1. Specifying Subscription Regions

The specification of subscription regions can be read from a configuration file using the **LoadSubscriptionListFile** command. This command can be included in the Logger configuration file in the startup section, as follows:

```
<command domain="Simulation" protocol="HLA 13"
  command="LoadSubscriptionListFile">
  <param name="filename" var="subscriptionListFile"></param>
</command>
```

A subscription list file has a list of region descriptors, a list of region sets, and a list of associations between region sets and object and interaction classes. Region descriptors specify the lower and upper bounds of each dimension in the region. The bound values are the normalized values used by the RTI. For the HLA 1.3 specification, the region descriptor includes the routing space name and must specify the bounds for all the dimensions in the routing space.

The following example configuration associates regions with object and interaction classes:

```
<?xml version="1.0" encoding="utf-8"?>
<subscriptionList>
  <regionList name="subRegionList">
    <region name="Region1">
      <space>RoutingSpaceName</space>
      <extent>
        <dimension name="Dim1">
          <lowerBound>0</lowerBound>
          <upperBound>20000</upperBound>
        </dimension>
        <dimension name="Dim2">
          <lowerBound>0</lowerBound>
          <upperBound>20000</upperBound>
        </dimension>
      </extent>
    </region>
    <region name="GlobalRegion">
      <space> RoutingSpaceName </space>
      <extent>
        <dimension name="Dim1">
          <lowerBound>0</lowerBound>
          <upperBound>4294967293</upperBound>
        </dimension>
        <dimension name="Dim2">
          <lowerBound>0</lowerBound>
          <upperBound>4294967293</upperBound>
        </dimension>
      </extent>
    </region>
  </regionList>

  <!-- Each region set contains one or more regions. -->

  <regionSetList name="subRegionSetList">
    <regionSet name="Region1Set">
      <region>Region1</region>
    </regionSet>

    <regionSet name="GlobalRegionSet">
      <region>GlobalRegion</region>
    </regionSet>
  </regionSetList>

  <!-- In the object and interaction class lists, if a class is not -->
  <!-- associated with a specific region set, it uses the region set -->
  <!-- specified by allObjectClasses or allInteractionClasses. -->
  <!-- If the all*Classes parameter is omitted, and SetDdmParameters -->
  <!-- subscribeUnspecifiedAttributes is disabled, only those classes -->
  <!-- that are explicitly associated with a region set are subscribed. -->

  <classList name="objectClassList">
    <allObjectClasses regionSet="GlobalRegionSet"/>
    <objectClass regionSet="RegionSet1">ObjectRoot.Class1</objectClass>
    <objectClass regionSet="RegionSet1"> ObjectRoot.Class2</objectClass>
  </classList>

  <classList name="interactionClassList">
```

```
<allInteractionClasses regionSet="GlobalRegionSet"/>
<interactionClass regionSet="RegionSet1">InteractionRoot.Class1
</interactionClass>
<interactionClass regionSet="RegionSet1">InteractionRoot.Class2
</interactionClass>
</classList>
</subscriptionList>
```

If individual attributes of a class must be associated with specific regions, then the regions can specify those attributes, as follows:

```
<region name="Region1">
  <space>RoutingSpaceName</space>
  <extent>
    <dimension name="Dim1">
      <lowerBound>0</lowerBound>
      <upperBound>20000</upperBound>
    </dimension>
    <dimension name="Dim2">
      <lowerBound>0</lowerBound>
      <upperBound>20000</upperBound>
    </dimension>
  </extent>
  <attribute class="ObjectRoot.Class1" name="Attribute1"/>
  <attribute class="ObjectRoot.Class1" name="Attribute2"/>
  <attribute class="ObjectRoot.Class1" name="Attribute3"/>
</region>
```

You must define a separate region for each set of class attributes for a given class and the class must still be associated with the region in the object class list. The class is subscribed with the specified list of attributes and region. If any attributes of the class are left unspecified, and the `subscribeUnspecifiedAttributes` parameter is disabled, they are not subscribed.

Loading the Subscription List Using the API

The subscription list file described in the previous section can be loaded through the API as follows:

```
void loadSubscriptionFile(DtLogger* logger, const DtString& fileName)
{
  DtSubscriptionListFile* sublist = DtSubscriptionListFile::create();
  sublist->readFromFile(fileName);
  logger->simManager()->setSubscriptionList(*sublist);
  delete sublist;
}
```

Alternatively, the subscription list can be constructed incrementally. For details, please contact support@mak.com.

5.5.2. Using Conveyed Regions

The HLA DDM service supports the concept of conveyed regions, which provides to the receiving federate the region extents associated with the reflected attributes or received interactions at the time the data was sent. By recording these conveyed region extents, the Logger can reproduce the regions during playback.

You can configure the Logger to record the conveyed region information. This region information can then be used during playback to create regions, update regions, and switch object attributes between regions. The conveyed regions provided by the RTI during Federate Ambassador callbacks specify the region extents that were active when the object instance attributes were updated or the interaction was sent. The Logger records these region extents with the received data. When playing back a file with region information, the Logger can send the data using the recorded region extents. Because the region information was provided by the RTI and recorded by the Logger, it is not necessary to inspect the data in order to create and update regions. The Logger replicates the same region extents as when the data was sent by the original federate.



To record conveyed region information, the RTI must be configured to deliver the conveyed region information in the Federate Ambassador callbacks. See the appropriate RTI documentation for enabling this HLA feature.

5.5.3. Using Simulation Data

The Logger plug-in API provides class methods used in the playback process that can be overridden to inspect the outgoing data to create and update regions to support DDM. DDM regions are used to send interactions and they are associated with object instance attributes. The interaction data can be inspected to select a region and then the region extents can be modified prior to sending the interaction. The object data can be inspected when the object publisher is created in order to select the regions used to register the object. As the object is updated, the region extents can be modified or the attributes can be associated with different regions.

5.5.4. Configuring DDM

The DDM settings are set using the `SetDdmParameters` command in the Logger configuration file, text interface, or on the command line. The syntax for this command in the text interface is:

```
SetDdmParameters enableDdm playbackMode subscribeMode  
playbackDynamicAssociation playbackDynamicExtents  
allowPlaybackWithoutRegions recordConveyedRegions  
requestUpdates subscribeUnspecifiedAttributes
```

For example:

```
SetDdmParameters true PlaybackRegionsUseConveyedExtents  
SubscribeRegionsAreGlobal true false true true true true
```

The parameters for the `SetDdmParameters` command are as follows:

- ♦ *enableDdm*. Enable record and playback using DDM. Default: false.
- ♦ *playbackMode*. Supported modes are:
 - **PlaybackRegionsUseConveyedExtents**. During playback, regions are created and updated using recorded region data.
 - **PlaybackRegionsUseData**. Regions are created and updated by inspecting simulation data (. Requires plug-in extension.
 - **PlaybackRegionNotEnabled** (default). Updates and interactions are sent without regions.
- ♦ *subscribeMode*. Supported modes:
 - **SubscribeRegionsAreGlobal**. DDM subscriptions are made with global regions with extents of global min and max.
 - **SubscribeRegionsAreSpecified**. The regions are specified through subscription configuration files. (For details, please see the `LoadSubscriptionListFile` command.) The subscription region configuration can also be specified programmatically through the Logger plug-in API.
 - **SubscribeRegionNotEnabled** (default). Subscriptions are performed without regions.
- ♦ *playbackDynamicAssociation*. During playback, the association of regions with object instance attributes can be fixed or dynamic. If it is known that region associations with attribute instances never change, the overhead of checking for changes can be eliminated. This parameter should be disabled if possible. Default: false.
- ♦ *playbackDynamicExtents*. During playback, the region extents can be fixed or dynamic. The conventional notion of DDM regions regards them as dynamically changing extents to reflect the state of simulation data (for example, surrounding an entity's position). An alternative approach is for the regions to be fixed and the object attribute instances move from region to region. If it is known that region extents never change, the overhead of checking for changes can be eliminated. This parameter should be disabled if possible. Default: false.

- ♦ *allowPlaybackWithoutRegions*. During playback, it may happen that a region association cannot be established. In these cases, the Logger sends the data without region association. Disable this parameter if the federation agreement requires that all data must be transmitted with associated regions. Default: true.
- ♦ *recordConveyedRegions*. Enable this parameter to record conveyed regions. Default: True.
- ♦ *requestUpdates*. Requesting updates can result in spikes in network traffic, especially since it is often performed by late joining federates when the exercise is already populated with many objects. In some DDM federations, object heartbeats are used instead of update requests. Disable this parameter to prevent the Logger from making requests for updates.

i

The Logger does not use region information to perform update requests. The default value is True.

- ♦ *subscribeUnspecifiedAttributes*. The Logger may not be able to determine a region association to apply to an interaction class or object class attribute subscription. This situation may occur if the interaction or class does not specify a routing space or dimension in the FOM. It can also happen if the *subscribeMode* is configured as **SubscribeRegionsAreSpecified**, but some classes or attributes are left unspecified. By default, the Logger subscribes to these interaction classes and object class attributes without using a region association (in effect a global subscription). Disable this parameter if the Logger should only subscribe when a region association can be used. Default: true.

5.5.5. Configuring DDM in *lgrConfig.xml*

The following is an example of configuring DDM in the Logger configuration file:

```
<defaults>
*
*
*
  <var name="playbackModeString" type="string"
    value="PlaybackRegionsUseConveyedExtents"/>
  <!-- <var name="playbackModeString" type="string"
    value="PlaybackRegionUsesData"/> -->
  <var name="subscribeModeString" type="string"
    value="SubscribeRegionsAreGlobal"/>
  <!-- <var name="subscribeModeString" type="string"
    value="SubscribeRegionsAreSpecified"/> -->
  <var name="True" type="bool" value="true"/>
  <var name="False" type="bool" value="false"/>
</defaults>

<commands name="startup">
*
*
*
  <command domain="Simulation" protocol="HLA 13"
    command="SetDdmParameters">
    <param name="enableDdm" var="False"></param>
    <param name="playbackMode" var="playbackModeString"></param>
    <param name="subscribeMode" var="subscribeModeString"></param>
    <param name="playbackDynamicAssociation" var="True"></param>
    <param name="playbackDynamicExtents" var="False"></param>
    <param name="allowPlaybackWithouRegions" var="True"></param>
    <param name="recordConveyedRegions" var="True"></param>
    <param name="requestUpdates" var="True"></param>
    <param name="subscribeUnspecifiedAttributes" var="True"></param>
  </command>
</commands>
```

5.6. Setting the Notification Level

The notification level determines the verbosity of informational messages that the Logger generates. The range is 0-4, with 0 being least verbose. Some informational messages are generated by the Logger application and some are generated by the VR-Link libraries it is built against. You can set the notification level for each message source independently.

- To configure the Logger notification level using a command interface, issue the `-m` and `-n` command-line options or the `SetNotifyLevel` command.

Table 5-2: Notification level commands

Interface	Command
Command line	<code>logger -m 2 -n 2</code> or <code>logger -r "SetNotifyLevel 2 2"</code>
Text interface	<code>SetNotifyLevel 2 2</code>
<i>lgrConfig.xml</i>	The <code>SetNotifyLevel</code> command is in the default configuration file.

5.6.1. Specifying the Log File

The Logger sends notification messages to a log file. The default log file is *logger.log*. If you assign a log file a name with the extension *.lgr*, the Logger adds a *.log* extension to ensure that the log file is not confused with a Logger file, or worse, overwrites a file.

- To configure the log file, issue the `-l` command-line option.

Table 5-3: Specifying the log file

Interface	Command
Command line	<code>logger -l mylogfile.log</code>
Text interface	None.
<i>lgrConfig.xml</i>	The <code>logFileName</code> variable is in the default configuration file.

5.7. Editing *lgrConfig.xml*

The *lgrConfig.xml* file contains commands that get run when you start the Logger. The default configuration file starts the Logger with initialization commands to load plugins and connect to the DIS or HLA network.

Each *.bin* directory has a version of *lgrConfig.xml* that is customized to the protocol used by the version of the Logger in that directory. Default versions of *lgrConfig.xml* are in *.data/backups*. These files are named by protocol so that you know which one to use if you need to revert a particular *lgrConfig.xml* to the default values. You can also use a default version by specifying it with the `--configFile` command-line option.

The file is written using XML syntax. It has essentially two sections – variable definitions and commands. Variable definitions are specified in the `<defaults>` block. Commands are specified with a series of `<command>` entries within one of two `<commands>` blocks. You can use any Logger command in a configuration file. (For a list of commands, please see “[Command Descriptions](#),” on page 11-4.)



Some command line options have a matching variable name in the defaults section of the configuration file. If you change or remove the name of one of the variables in the default configuration file, it will break the command line option associated with it.

Additionally, if you add a new variable and give it a name that matches an existing command line argument, you will get unexpected behavior. Being careful about how you name new variables should eliminate any risk of this error occurring.

The following is a heavily edited version of a configuration file for HLA that shows the main components:

```
<lgrConfig>
  <!-- Variable formatting -->
  <defaults>
    <var name="messageLevel" type="int" value="2"/>
    <!-- HLA Settings -->
    <var name="execName" type="string" value="VR-Link"/>
    <var name="federateName" type="string" value="Logger"/>
    <var name="fedFileName" type="string" value="VR-Link.fed"/>
    <var name="fomMapperLib" type="string" value=""/>
    <var name="fomMapperInitData" type="string" value=""/>
    <var name="rprFomVersion" type="double" value="1.0"/>
    <var name="rprFomRevision" type="int" value="1"/>
    <var name="ignoreAdvisoriesValue" type="bool" value="false"/>
    <var name="useEmptyFomMapperValue" type="bool" value="false"/>
  </defaults>

  <!-- Initialize commands executed after the command line is parsed -->
  <!-- Parameter variables must be defined in the defaults group -->
  <commands name="init">
    <command domain="System" protocol="None" command="SetNotifyLevel">
      <param name="lgrLevel" var="messageLevel"/></param>
    </command>
  </commands>

  <commands name="startup">
    <command domain="Simulation" protocol="HLA 13"
      command="SetConnectionParameters">
      <param name="federateName" var="federateName"/></param>
      <param name="federationName" var="execName"/></param>
      <param name="fedFile" var="fedFileName"/></param>
      <param name="fomVersion" var="rprFomVersion"/></param>
      <param name="fomRevision" var="rprFomRevision"/></param>
      <param name="fomMapper" var="fomMapperLib"/></param>
      <param name="fomMapperData" var="fomMapperInitData"/></param>
      <param name="ignoreAdvisories" var="ignoreAdvisoriesValue">
        </param>
      <param name="useEmptyFomMapper" var="useEmptyFomMapperValue">
        </param>
      </command>
    <command domain="Simulation" protocol="None" command="Connect"/>
  </commands>
</lgrConfig>
```

Command Syntax

Commands that do not take arguments have the format:

```
<command domain="domain" protocol="protocol" command="command"/>
```

where:

- ♦ *domain* is System, Player, Recorder, Editor, or Simulation. Please see “[Command Descriptions](#),” on page 11-4 for a list of commands by domain.
- ♦ *protocol* is None, DIS, HLA 13, HLA 1516, or HLA 1516e.
- ♦ *command* is the command you want to run, from the list of supported commands in Chapter 11, [Using Logger Commands](#).

The following is an example of a command that does not take arguments:

```
<command domain="Simulation" protocol="None" command="Connect"/>
```

Commands that take arguments have the format:

```
<command domain="domain" protocol="protocol" command="command">
  <param name="name" var="variableName"></param>
</command>
```

where:

- ♦ *name* is the name of a command argument as listed in the command descriptions in “[Command Descriptions](#),” on page 11-4.
- ♦ *variableName* is the user-defined name of a variable that is to be defined in the <defaults> section of the configuration file.

The following is an example of a command that takes arguments:

```
<command domain="System" protocol="None" command="SetNotifyLevel">
  <param name="lgrLevel" var="messageLevel"></param>
</command>
```

Variable Syntax

Variable entries take the format:

```
<var name="variableName" type="type" value="value"/>
```

where:

- ♦ *variableName* is a variable that is specified by a command in the commands section of the configuration file.
- ♦ *type* is the argument type for the variable, as defined for each command argument in “[Command Descriptions](#),” on page 11-4.
- ♦ *value* is the value you want to assign to the variable.

For example:

```
<var name="messageLevel" type="int" value="2"/>
```

5.7.1. Adding a Command to *lgrConfig.xml*

When you add a command to the configuration file, you will usually add it to the startup commands section. The only time you would add a command to the init section is if you are adding a command to load a plug-in.

Suppose that you want to add the `NewFile` command to *lgrConfig.xml*. This command creates a new file for recording. If you add it to the configuration file, every time you start the Logger, it will create a new file with the specified name.

1. Add a new command in the `<commands name="startup">` section, similar to the following:

```
<commands name="startup">
  <command domain="Recorder" protocol="None" command="NewFile">
    <param name="filename" var="newfile"></param>
  </command>
</commands>
```

In the command definition, the domain is `Recorder`, which you know because `NewFile` is listed with the commands in the `Recorder` domain in “[Command Descriptions](#),” on page 11-4. Protocol is `None`, because the command is protocol independent. The command is `NewFile`.

The command definition shows that `NewFile` takes an argument, `filename`, of type `string`. Therefore, add a parameter definition to the command. The name is `filename`. This must be exactly the same spelling as in the definition. The variable name is a user-defined name that will be defined in the variable definition.

2. Add a variable definition for the variable `newfile`, similar to the following:

```
<var name="newfile" type="string" value="newfilename.lgr"/>
```

The name, `newfile`, matches the variable name in the command entry. The type is the type listed in the command definition. The value is whatever value you want to assign, in this case, the name you want to give to the Logger file that will be created on startup.

Adding a String List to *lgrConfig.xml*

Some commands require specification of multiple strings, for example, the list of common entity types in the configuration file for DIS. As with other commands, you specify a variable to take the data. Then you provide the strings as a list of listitems. For example, here is how you could specify a subscription list for HLA:

1. In the command section of the configuration file, add an entry for the `SetSubscriptionLists` command, as follows:

```
<!-- To Disable auto-connection remove this command -->
  <command domain="Simulation" protocol="None" command="Connect"/>
</commands>

<command domain="Simulation" protocol="HLA 13"
  command="SetSubscriptionLists">
  <param name="objectClasses" var="objectSubscriptionList"></param>
  <param name="interactionClasses" var="interactionSubscriptionList">
    </param>
</command>
```

Since the `SetSubscriptionLists` command takes two arguments, we define two parameters for it.



We include the `Connect` command, which is in the default configuration file, because the Logger must be connected before you can issue a `SetSubscriptionLists` command. When you use Logger commands, you must take into account any prerequisites that they have.

2. Add a variable definition for each of the parameters defined in the command. Within each variable definition, include a listitem for each string you want to include in the list.

```
<var name="objectSubscriptionList" type="list" listtype="string">
  <listitem>ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft
  </listitem>
  <listitem>ObjectRoot.BaseEntity.PhysicalEntity.Radio</listitem>
</var>

<var name="InteractionSubscriptionList" type="list" listtype="string">
  <listitem>InteractionRoot.WeaponFire</listitem>
</var>
```



The subscription list created by this example is identical to the one that would be created by the example command in [“Using Logger Commands on the Command Line,”](#) on page 11-2 and [“Using Logger Commands in the Text Interface,”](#) on page 11-3.

5.8. Summary of Command-line Options for the Logger

Table 5-4 lists command-line options. Command-line options override the comparable settings in the *lgrConfig.xml* file.

Table 5-4: Command-line options (Sheet 1 of 3)

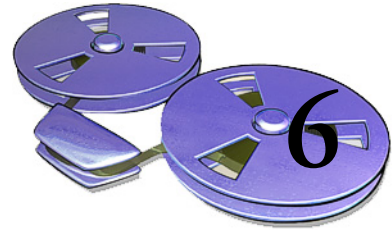
Option	Description
<code>--allowAllRemoteControl mode</code>	Specifies the response to remote control PDUs. 1 = Allow 2 = Do not allow.
<code>{-c --showConsole} mode</code>	Enables (1) or disables (0) display of a console window.
<code>--configFile filename</code> <i>filename</i>	Specifies the configuration file to load. Specifies a Logger file to load. This option uses an implicit <code>-r OpenFile</code> . If a filename starts with a hyphen (-), you must use the <code>-r OpenFile</code> syntax.
<code>{-- --ignore_rest}</code>	Ignore all arguments after this one.
<code>{-h --help}</code>	Displays a brief summary of Logger command-line arguments.
<code>{-l --logFileName} filename</code> <i>filename</i>	Specifies a file to send notification messages to. Default: <i>logger.log</i> .
<code>--loggerTickRate rate</code> <i>rate</i>	The interval for the Logger tick rate, in milliseconds. For more information, please see “The SetPlayback-ReadTimeLimit Command,” on page 5-6
<code>{-m --messageLevel} level</code> <i>level</i>	Specifies the notification level for Logger messages for the log file. Range: 0-4.
<code>{-n --notifyLevel} level</code> <i>level</i>	Specifies the notification level for VR-Link messages for the log file. Range: 0-4.
<code>{-r --runCommand} command</code> <i>command</i>	Specifies a Logger command to run. You can specify <code>-r</code> multiple times. For more information, please see “Using Logger Commands on the Command Line,” on page 11-2.
<code>{-v --version}</code>	Displays version information and exits.
HLA Only	
<code>{-f --fomMapperLib} libname</code> <i>libname</i>	Specifies the FOM Mapper to use if you are not using the default FOM. For details about how the Logger chooses the FOM Mapper, please see “Specifying a FOM Mapper,” on page 4-11.
<code>--federateType type</code> <i>type</i>	Specifies the federate type for the Logger federate. (HLA Evolved only)
<code>--fomMapperInitData data</code> <i>data</i>	Specifies FOM Mapper initialization data, if required.

Table 5-4: Command-line options (Sheet 2 of 3)

Option	Description
<code>--fomModules "module1 module2 ... moduleN"</code>	Specifies one or more FOM modules for the exercise. (HLA Evolved only)
<code>{-F --fedFileName} fedfile</code>	Specifies the FED file for the exercise.
<code>--ignoreAdvisories mode</code>	If 1, ignore HLA advisory messages. If 0, do not ignore.
<code>--MIM MIM_file</code>	Specifies the MIM (MOM and Initialization Module) for the exercise. (HLA Evolved only)
<code>{-N --federateName} federate_name</code>	Specifies the federate name for the Logger federate.
<code>--rprFomVersion version</code>	Specifies the version of the RPR FOM to use.
<code>--rprFomRevision revision</code>	If using RPR FOM 2.0017, specifies the revision to use.
<code>--ignoreAdvisories mode</code>	Specifies whether (1) or not (0) the Logger ignores advisory messages.
<code>--useEmptyFomMapper mode</code>	Specifies whether (1) or not (0) the Logger uses an empty FOM Mapper. For details about how the Logger chooses the FOM Mapper, please see "Specifying a FOM Mapper," on page 4-11.
<code>{-x --execName} ExName</code>	Specifies the federation execution name. Default: VR-Link.
DIS Only	
<code>{-A --destAddrString} address</code>	Specifies the default IP address to which the Logger sends PDUs. If this option is not used, the Logger uses the computer's broadcast address. Please see "Specifying a Destination Address for Outgoing and Incoming Data," on page 4-19.
<code>{-a --appNumber} appnumber</code>	Specifies the application number.
<code>{-D --deviceAddrString} address</code>	Specifies the network device address if your computer has more than one network card.
<code>--disVersion num</code>	Specifies the DIS protocol version for PDUs sent onto the network. <i>num</i> can be 4 (DIS 2.0.4), 5 (IEEE 1278.1), or 6 (DIS 2.1.4, IEEE 1278.1a):
<code>{-e --exerciseId} ID</code>	Specifies the exercise ID.
<code>--mcastTtl ttl</code>	Specifies the multicast time-to-live.
<code>{-P --disPort} port</code>	Specifies which DIS port (UDP port if -A not specified, otherwise, TCP) the Logger uses to send or receive data. Default: 3000.
<code>--recvBufferSize size</code>	Specifies the receive buffer size.

Table 5-4: Command-line options (Sheet 3 of 3)

Option	Description
<code>--remoteControlId ID</code>	Specifies the Logger's remote control ID. By default the value is assigned the Logger's site ID. This value is used if <code>SetRemoteControlSettings allowAllRemoteControl</code> is disabled.
<code>{-S --mcastAddresses} addresses</code>	Subscribes the Logger to the specified multicast address. To subscribe to several multicast addresses, use this option more than once on the command line.
<code>--sendBufferSize size</code>	Specifies the send buffer size.
<code>{-s --siteId} ID</code>	Specifies the site ID.



Recording and Playing Logger Files

This chapter explains how to use the Logger's core features from the GUI.

Starting the Logger.....	6-3
Starting the Logger on Linux.....	6-3
Starting the Logger in Windows.....	6-3
The Logger Window.....	6-4
Hiding and Displaying Plug-in Views	6-5
Docking and Undocking GUI Plug-in Views and Toolbars	6-5
Connecting to the Simulation Network	6-6
Disconnecting from the Simulation Network	6-6
Recording Exercises.....	6-7
Recording Immediately (to a Temporary File)	6-7
Recording to a Specified File	6-7
Recording to Multiple Files	6-8
Pausing the Recording Process	6-10
Stopping the Recording Process.....	6-10
Opening Logger Files	6-11
Opening Incomplete Files	6-12
Closing a Logger File.....	6-12
Playing Logger Files	6-13
Playing a Logger File	6-13
Pausing Playback of a Logger File	6-13
Playing a Segment of a File.....	6-13
Looping Playback.....	6-15
Stopping Playback of a Logger File.....	6-15
Setting the Playback Speed	6-15
Using High Speed Playback.....	6-16

Saving Files	6-17
File Backups	6-17
Saving a File to a New Name.....	6-17
Saving a Portion of a File to a New File	6-17
Upgrading a Logger File's Format	6-18
Zooming the Timeline.....	6-18
Using the Zoom Sliders.....	6-18
Setting the Zoom from the Set Zoom Dialog Box	6-19
Resetting the Zoom Level.....	6-19
Moving the Timepoint to a Specific Time.....	6-20
Jumping to the Beginning or End of the Playback Range	6-20
Moving to a Specific Time in a File	6-20
Setting the Time by Dragging the Timeline Cursor	6-21
Viewing a File's Packet Count Histogram.....	6-21
Viewing Object Statistics	6-22

6.1. Starting the Logger

The Logger has separate executables for each protocol supported (please see [Table 2-1](#) for a description of the directory structure). You can use command-line options to override default configuration parameters and otherwise specify the Logger's initial settings. “[Summary of Command-line Options for the Logger](#),” on page 5-23, lists command-line options for the Logger.

6.1.1. Starting the Logger on Linux

To start the Logger:

1. Open a Linux shell.
2. Change to the appropriate *bin* directory.
3. Enter:

```
logger [filename options]
```

6.1.2. Starting the Logger in Windows

Start the Logger in any of the following ways:

- ♦ On the Start Menu, choose **Programs** → **MÄK Technologies** → **Logger 5.1** → **Logger *protocol***, where *protocol* is DIS, HLA 1.3, HLA 1516, or HLA Evolved.
- ♦ In a command window, change to the appropriate *bin* directory, and type:

```
logger [filename options]
```
- ♦ Create a shortcut on the Windows desktop and specify the appropriate command line in the Properties dialog box.
- ♦ In the Windows Explorer, double-click a Logger file.

6.2. The Logger Window

The Logger window has a set of control buttons, a timeline, and a status bar (Figure 6-1). Data supplied by plug-ins, such as the data view and annotations, is displayed between the toolbar controls and the timeline. You can enable and disable the display of plug-in data.

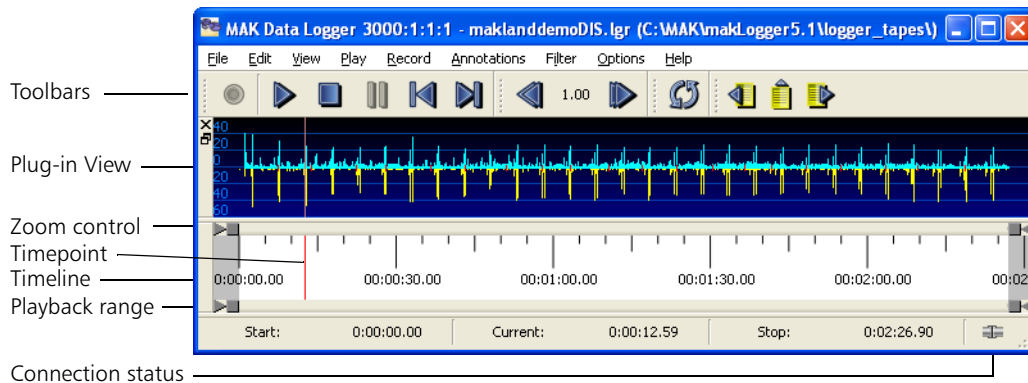


Figure 6-1. Logger window with a Logger file loaded

The Logger window controls, as identified in Figure 6-2, are described in the list that follows.



Figure 6-2. Logger controls

- Record (1)
- Play (2)
- Stop (3)
- Pause (4)
- Move the timepoint (playback cursor) to the start of the playback range (5)
- Move the timepoint to the end of the playback range (6)
- Decrease the playback speed (by 50%) (7)
- Displays current speed; open Set Playback Speed dialog box (8)
- Increase the playback speed (double the speed) (9)
- Set the end of playback action (stop or wrap) (10)
- Jump to the previous annotation (11)
- Insert a point annotation (12)
- Jump to the next annotation (13).

6.2.1. Hiding and Displaying Plug-in Views

Plug-ins can display data in the area above the zoom control. Multiple plug-in views can be displayed at one time.

- To display plug-in data, select the appropriate option on the View menu (the menu options toggle the display on and off).

To hide a plug-in view, use one of the following methods:

- ♦ Select the appropriate option on the View menu.
- ♦ Click the hide button for the plug-in view that you want to hide (Figure 6-1).

6.2.2. Docking and Undocking GUI Plug-in Views and Toolbars

You can undock and dock plug-in views and toolbars from the Logger window and place them anywhere on the desktop. You can dock the toolbars to the window at the top and sides and can reorganize the order in which they are placed.



Floating windows are supported only in the Windows version of the Logger.

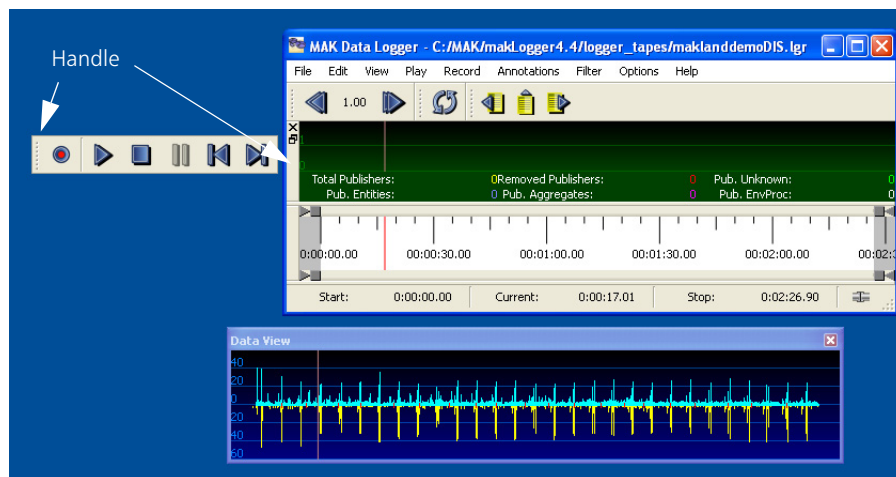


Figure 6-3. Undocked GUI components

- To undock a plug-in view or toolbar, click and hold its handle and drag it off the window.
- To dock a toolbar, drag it to the Logger window where you want to dock it. You can dock toolbars to the top and sides of the window.
- To dock a plug-in view, drag it to the window or double-click its title bar. You can dock plug-in views in the space above the timeline.



If the Logger window is resized such that there is not enough room above the timeline for a plug-in view, you might not be able to dock a previously undocked view. If this happens, double-click the title bar of the undocked view.

6.3. Connecting to the Simulation Network

When you start the Logger, it automatically connects to the simulation network using the stored connection parameters. If you disconnect from the network, you can connect again as described in this section and as described in “Editing a Connection,” on page 4-4.

- To connect to the network, choose **File** → **Connect**, or double-click the connection status icon on the status bar ().

6.3.1. Disconnecting from the Simulation Network

When you disconnect from an exercise, you can no longer send data to it or record data from it.



You cannot disconnect from an exercise while you are recording or playing a file.

- To disconnect from an exercise from the GUI, choose **File** → **Disconnect**, or double-click the connection status icon on the status bar ().

6.4. Recording Exercises

When you record an exercise, you can specify a name for the new recording and then start the recording process, or you can start recording immediately and save the recording to a file when you are done (the default behavior).

The Status Bar displays the starting time, the current time, and the total time of the recording.

As recording progresses, the Time field is updated to display elapsed time. In record mode, the timescale is fixed at 1.0 (normal speed).



- ♦ If you are recording an exercise that is running faster than real-time, the Logger records the simulation at that rate and will play it back faster than real-time when the playback timescale is set to 1.0.
 - ♦ You cannot record while you are playing a file.
-

6.4.1. Recording Immediately (to a Temporary File)

- To begin recording, choose **Record** → **Record**, or click the Record button (●).

6.4.2. Recording to a Specified File

Unless you enable Pause on Record As, the Logger starts recording as soon as you specify a filename.

To record an exercise to a named file:

1. Choose **Record** → **Record As**. The New Logger File dialog box opens.
2. Type a new filename or select an existing file.



If you select an existing file, its contents will be overwritten by the new file.

3. Click Save. If Pause on Record As is disabled (the default), the Logger starts recording immediately.
- If Pause on Record As is enabled, to begin recording, choose **Record** → **Record**, or click the Record button (●).

Enabling Pause on Record As

By default, when you choose Record As, the Logger starts recording as soon as you close the New Logger File dialog box. If you do not want to start recording immediately, you must enable Pause on Record As. When Pause on Record As is enabled, the Logger does not start recording until you click the Record button.

- To enable or disable immediate recording for the Record As command, choose **Options** → **Pause on Record As**.

6.4.3. Recording to Multiple Files

The Logger can record an exercise to multiple files as it is recording the primary Logger file. When you record to multiple files, the Logger saves data to file at specified time intervals. You can play or otherwise process these files in a second Logger while the first Logger continues to record the exercise. This option also ensures that if logging of the exercise is interrupted due to system or application instability, you will not lose the data saved prior to the application problems.

To record to multiple files:

1. Choose **Record** → **Record to Multiple Files**. The Record to Multiple Files dialog box opens.
2. Optionally, change the location to which the complete file will be saved in the Complete File Directory text box.
3. In the Complete File Name box, specify the name of the complete file.

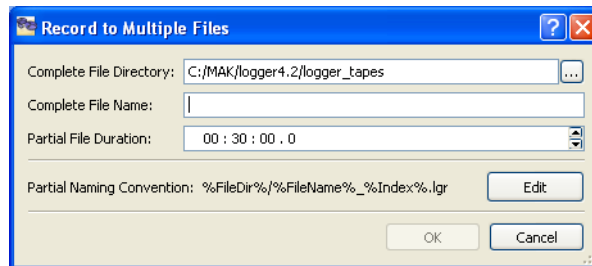


Figure 6-4. Record to Multiple Files dialog box

4. Specify the length of each partial file in the Partial File Duration box.
5. Optionally, change the convention used to name the new files. For details about changing the partial file naming convention, please see [“Specifying the Partial File Naming Convention,”](#) on page 6-9.
6. Click OK.

Specifying the Partial File Naming Convention

The default convention for naming partial files is *complete_filename_n.lgr*. Files are written to the complete file directory. However, you can change the naming convention using a set of macros provided with the Logger.

To change the partial file naming convention:

1. In the Record to Multiple Files dialog box, click Edit. The dialog box expands to display a list of macros.

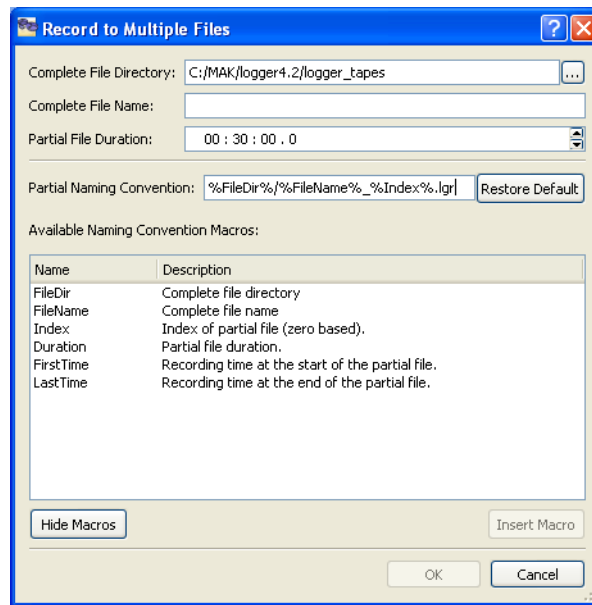


Figure 6-5. Record to Multiple Files dialog box in Edit mode

2. Edit the text in the Partial Naming Convention box manually as desired or add macros, or both.
3. To add a macro to the naming convention, select it in the list and click Insert Macro (or double-click the list entry). It gets added at the location of the insertion point in the Partial Naming Convention box. To ensure that each partial file has a unique name, the naming convention must include at least one of Index, First-Time, or LastTime. For explanations of the macros, please see [Table 6-1](#).
4. Click Hide Macros to close the macro list.
5. Click Restore Default to revert to the default naming convention.

Table 6-1: Partial file naming convention macros

Macro	Definition
FileDir	The path of the directory in which the complete file will be saved. The macro does not include the final path separator.
FileName	The name of the complete file, without the file extension.
Index	An integer index, starting at 0 and increasing by one for each partial file created.
Duration	The duration of the file in seconds. Each partial file, except the final file will be this long.
FirstTime	The time in the complete file that maps to the beginning of the new file, in seconds.
LastTime	The time in the complete file that maps to the end of the new file, in seconds.

6.4.4. Pausing the Recording Process

You can pause the recording process and then resume recording to the same file.



When you pause during the recording process, the file is not saved. You cannot play back the file from the paused recording state. If for some reason you exit the Logger while in the paused state, the file will not be saved.

- To pause the recording process, click the Pause button (⏸).
- To resume a paused recording, choose **Record** → **Record**, or click the Record button (⏵).

6.4.5. Stopping the Recording Process

When you stop recording, the Logger enters playback state and you cannot resume recording. If you want to resume a recording, pause it.



If the Logger has not recorded any data, it cannot create a file.

- To stop recording, choose **Record** → **Stop**, or click the Stop button (⏹).

6.5. Opening Logger Files

If you want to play a file or edit it, you must open the file. When you open a file, its filename is displayed in the Title Bar. If the Data View is enabled, a histogram is calculated and displayed. (For details about the Data View, please see “[Viewing a File’s Packet Count Histogram](#),” on page 6-21.)

To open a Logger file:

1. Choose **File** → **Open**. The Open File dialog box opens.
2. Select the file.
3. Click Open. If there is an annotations file with the same name as the Logger file, you are prompted to open it. If Play On Load is enabled, the Logger starts playing the file immediately.

i

- ♦ You can also load a file by double-clicking its icon in a folder or by dragging it onto the Logger window.
 - ♦ If you load a file that was created using a previous Logger format, you are prompted to upgrade to the current format. If you decline to do so, you can upgrade manually at a later point. For details, please see “[Upgrading a Logger File’s Format](#),” on page 6-18.
 - ♦ If you load a file that was made with a previous version of the Logger and which is not indexed, the Logger does not index the file. Therefore, time jumps may take longer than they would for indexed files.
-

6.5.1. Opening Incomplete Files

When the Logger finishes recording a file, it writes a table of all entities and a checkpoint index table to the end of the file. It uses this data to load and play back the file. If recording terminates abnormally, for example due to a crash or running out of disk space, this data is not written. The Logger can repair and open such files.

When the Logger loads a file that is missing the end-of-file data, it prompts you to repair the file. Your options for repairing files are as follows:

- ♦ Repair the file. If you choose this option, the Logger uses the data in the incomplete file and the current checkpointing policy to create a new, complete file, which it then loads. It saves the repaired file to the same location as the incomplete file. The new file is named *original_filenameRepaired.lgr* (for example, *../logger_tapes/demo.lgr* would become *../logger_tapes/demoRepaired.lgr*). The original file is unchanged.
- ♦ Do not repair the file. If you choose this option, then the partial file is loaded. It has the original checkpoints if they are in the file; however the checkpoints may be unreliable. Also, when you load an incomplete file, it does not load the entity tracking information. The entity tracking table provides the entity information for filtering by entity name (entity ID in DIS). Therefore, filter by entity name will have an empty list.

The partial file cannot be saved or saved to another filename. However, it can be exported, which will create a repaired file. Also, editing the file will repair it and allow it to be saved using the original name. The entity tracking information will be restored to the loaded, edited copy.

6.5.2. Closing a Logger File

- To close a Logger file, choose **File** → **Close**.

6.6. Playing Logger Files

The Status Bar in the Logger window displays the starting time for playback, the current time in the file, and the ending time for playback.



- ♦ Playback options are restricted in time-managed mode. For details, please see [“Some Logger Features are Restricted When You Use Time Management,”](#) on page 3-5.
- ♦ For information about how to configure the Logger for faster playback, please see [“Configuring Faster Playback,”](#) on page 5-4

6.6.1. Playing a Logger File

By default, the Logger enters a stopped state when a file is opened.

To play a Logger file:

1. Open the file.
2. Choose **Play** → **Play/Pause**, or click the Play button (

Enabling Immediate Playback

When Play On Load is enabled, the Logger starts playing as soon as you open a file. If the file has an annotation file, it is loaded automatically. When Play On Load is disabled, you must click the Play button to start playing.

- To enable or disable immediate playback, choose **Options** → **Play On Load**.

6.6.2. Pausing Playback of a Logger File

- To pause playback of a Logger file, choose **Play** → **Play/Pause**, or click the Pause button (

6.6.3. Playing a Segment of a File

You can configure the Logger to play any arbitrary segment of a file. If you have the Logger configured to loop, it loops within the segment you have specified. You can specify exact start and stop times in a dialog box or you can specify the start and end times with the time sliders.

Specifying a Playback Segment Using the Time Sliders

To play a segment of a file:

1. On the Logger timeline, drag the start time slider (Figure 6-6) to the time at which you want to start playback. The Start time display in the Status Bar shows the exact time that you select.

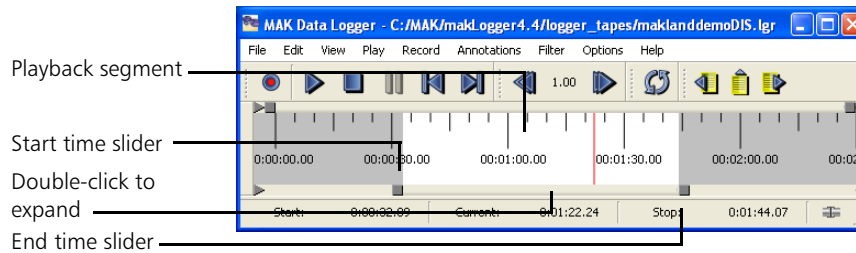


Figure 6-6. Timeline sliders

2. Drag the end time slider to the point on the timeline at which you want to end playback. The Stop time display in the Status Bar shows the exact time that you select.
3. Play the file.

Specifying a Playback Segment in a Dialog Box

To specify a playback segment using a dialog box:

1. Choose **Play** → **Set Start and End Time**. The Set Start and End Time dialog box opens (Figure 6-7).

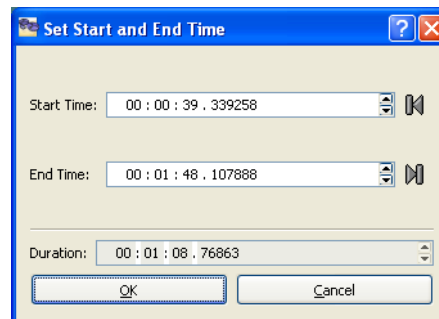


Figure 6-7. Set Start and End Time dialog box

2. Type start and end times or use the spinner control to select them. The duration of the segment is displayed in the Duration box.
3. To reset the start time to 0 or the end time to the end of the file, click the reset buttons to the right of the time value.
4. Click OK.

Expanding the Playback Segment to Full Length

- To expand the playback segment to the full length of the file, drag the sliders back to the edge of the window or double-click the timeline between the start and end time sliders (Figure 6-6).

6.6.4. Looping Playback

When the Logger plays a file to the end or the end of a playback range, it can stop or it can loop back to the beginning of the file or range and start playing it again.

- To set the looping behavior, choose **Play** → **Toggle Loop at End**, or click the Toggle Loop At End icon (🔄) in the Logger window (Figure 6-2).

6.6.5. Stopping Playback of a Logger File

- To stop playback of a Logger file, choose **Play** → **Stop**, or click the Stop button (■).

6.6.6. Setting the Playback Speed

You can vary the playback speed. The icon bar lets you increase speed by a factor of 2 and decrease by 0.5. The Set Playback Speed dialog box lets you increase and decrease the speed by any value. If you set the speed using the Set Playback Speed dialog box, the icon bar displays the new speed and increases or decreases the speed relative to the new value, not the default set of values. The actual speed of the action in the file depends on the speed at which you recorded it.

To set the playback speed:

1. Choose **Play** → **Playback Speed** → **Set Playback Speed**, or click the current speed setting on the Logger toolbar (Figure 6-2). The Set Playback Speed dialog box opens (Figure 6-8).

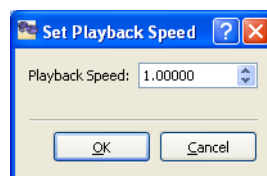


Figure 6-8. Set Playback Speed dialog box

2. Type a speed or set it using the controls.
3. Click OK.

You can quickly increase or decrease the playback speed by a factor of two.

- To quickly increase or decrease playback speed, choose **Play** → **Playback Speed** → **Increase Playback Speed 2x** (or **Decrease Playback Speed 1/2x**), or click the Increase Speed button () or Decrease Speed button () on the icon bar.

6.6.7. Using High Speed Playback

The high speed playback feature lets you play files at higher speeds than would normally be possible. (The Logger has a practical limit on the rate at which it can send updates and interactions based on processor and network speed.) Use it when the combination of playback speed and file density (number of recorded packets per second) results in a required packet output rate that exceeds the rate at which the Logger can transmit the packets on the network. For example, high speed playback would be useful for simulations with many entities, or simulations that are quite lengthy (for which you might want to speed up playback significantly), or both.

In high speed playback, instead of playing every packet in the file, the Logger skips from checkpoint to checkpoint and sends the entity updates for the entities that exist at the time of each checkpoint. It does not send interactions. Where necessary, the Logger creates or removes the entities that were created or removed in between the jumps. The effect is very similar to that of a digital video player 20-60x playback, in which snapshots of scenes are shown, but not every frame.

When high speed playback is enabled, the Logger sends updates only for checkpoints. It does not send any messages between checkpoints. The frequency of updates is determined by the frequency of checkpoints in the file and the playback speed. For example, suppose that you have a file that has checkpoints every 60 seconds. If you play the file at 60x speed, playback would advance to a new checkpoint every second.

i

- During high speed playback, the Logger does not send any interactions.
 - To use high speed playback, a file must contain checkpoint data.
-

By default, high speed playback is disabled. To enable it, you must edit the *lgrConfig.xml* file for the protocol that you are using. In *lgrConfig.xml*, set the value for the `skippingPlaybackThreshold` variable to a value greater than 0, for example:

```
<var name="skippingPlaybackThreshold" type="double" value="16.0"/>
```

The value you enter is the playback speed at or above which the Logger skips from checkpoint to checkpoint rather than playing every packet. Therefore, do not set the threshold too low unless you have a very high entity count.

Once you configure high speed playback, set the playback speed, as described in [“Setting the Playback Speed,”](#) on page 6-15, and if the speed meets or exceeds the threshold, high speed playback is used.

6.7. Saving Files

When you record a simulation or edit a Logger file, the file is in an unsaved state, which is indicated by an asterisk next to the filename in the Title bar. If you specify a name for a file before you start the recording process, it is automatically saved when you stop recording. If you have an unsaved file loaded, the Logger prompts you to save it before exiting, opening a file, closing a file, or recording a new file.

- To save a file, choose **File** → **Save**. If the file already has a filename, it is saved immediately. If it does not yet have a filename, you are prompted to supply one (Save As).

6.7.1. File Backups

When you save a previously saved file, the Logger creates a backup file by renaming the previous version of the file to *filename.bak.lgr*. For example, if you edit *berkeley.lgr* and save the changes, the original version of *berkeley.lgr* is saved to *berkeley.bak.lgr*.

The Logger maintains only one backup for a particular Logger file. If you edit and save a file that was previously backed up, the original backup file gets replaced with a newer backup file. For example, if after saving *berkeley.lgr* in the previous example, you were to edit it again and save it, a new version of *berkeley.bak.lgr* would replace the original backup file.

You can disable automatic backup with the `SetBackupOnSave` command.

6.7.2. Saving a File to a New Name

- To save a file to a new name, choose **File** → **Save As** and type a new name in the Save As dialog box.

6.7.3. Saving a Portion of a File to a New File

You can save a segment of a file to a new file. The timestamps of the packets in the new file are offset so the first packet is at time zero. If the segment has annotations, they get saved with the segment.

To save a segment of a file to a new file:

1. Open the file.
2. Use the timeline sliders to select the segment you want to save.
3. Choose **File** → **Export Selection To** → **Logger File**. A standard Save file dialog box opens.
4. Enter a name for the new file.
5. Click OK.

6.7.4. Upgrading a Logger File's Format

When you load a Logger file that was created with a previous format, the Logger can automatically upgrade to the latest file format. You can also upgrade manually.

To upgrade a file to the latest format:

1. Open the file.
2. Choose **Edit** → **Upgrade File Format**.

6.8. Zooming the Timeline

You can zoom in and out on the timeline (adjust the granularity) and change the time displayed at either end of it. You can zoom manually, using the zoom sliders, or you can specify zoom boundaries, center point, and duration in the Set Zoom dialog box.

6.8.1. Using the Zoom Sliders

- To zoom the timeline in or out using a zoom slider, drag a zoom slider (Figure 6-9) to the left or right.
- To return a zoom slider to its default location, click the appropriate zoom out arrow.

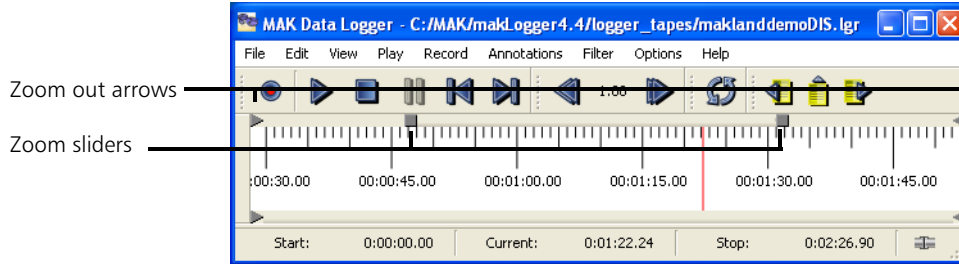


Figure 6-9. Zoom sliders

Dragging a zoom slider affects the timeline as follows:

- ♦ Dragging a zoom slider toward the middle of the Logger window zooms in.
- ♦ Dragging a zoom slider toward the edge of the window zooms out.
- ♦ Dragging the left zoom slider changes the leftmost time visible in the timeline.
- ♦ Dragging the right zoom slider changes the rightmost time visible in the timeline.



Adjusting the timeline zoom and the visible time segment does not affect the start and end times for playback. These are determined by the timeline controls. Please see “[Playing a Segment of a File](#),” on page 6-13 for details about the playback time.

6.8.2. Setting the Zoom from the Set Zoom Dialog Box

The Set Zoom dialog box allows you to precisely specify the start and end point of the zoomed view.

To zoom the timeline using the Set Zoom dialog box:

1. Choose **View** → **Set Zoom**. The Set Zoom dialog box opens (Figure 6-10).

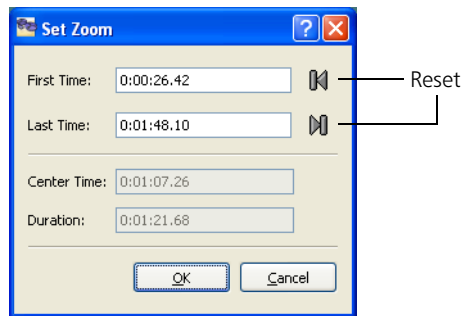


Figure 6-10. Set Zoom dialog box

2. Enter the beginning time in the First Time box.
3. Enter the end time in the Last Time box.
4. If you want to reset the start time value to 0:00:00.0 or the end time to the end of the file, click the appropriate Reset button.
5. Click OK.

6.8.3. Resetting the Zoom Level

- To quickly zoom the timeline to its full length, double-click on the zoom bar between the zoom sliders (Figure 6-11).

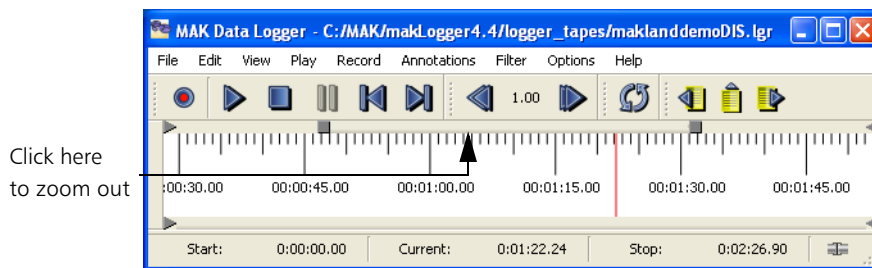


Figure 6-11. Zooming out

6.9. Moving the Timepoint to a Specific Time

The Logger provides the following ways to move the timepoint to a specific time:

- ♦ Move to the beginning or end of the playback range
- ♦ Move to a specific time
- ♦ Move to a time defined by an annotation
- ♦ Drag the timeline cursor.

i

- ♦ If you execute a time jump while playback is paused or stopped, the Logger remains in that state. The Logger also retains a play state after a time jump unless the end of the file is reached and wrapping is disabled.
 - ♦ During slow-speed playback, when you use the Set Time option, the Logger sends interim messages to ensure that entities do not time out. For more information, please see [“Updating Entity Information Quickly After a Time Jump,”](#) on page 3-7.
-

6.9.1. Jumping to the Beginning or End of the Playback Range

- To jump to the beginning of a playback range, choose **Play → Jump To Start**, or click the Jump to Start button (.
- To jump to the end of a playback range, choose **Play → Jump To End**, or click the Jump to End button (.

6.9.2. Moving to a Specific Time in a File

The Set Time feature lets you move to a specific point in time measured from the start of the file. The speed of time jumps is affected by the checkpoint policy. For details about setting the checkpoint policy, please see [“Configuring Indexing,”](#) on page 5-2.

To move to a specific time:

1. Choose **Play → Jump to Time**. The Jump to Time dialog box opens.
2. Enter the time you want to jump to.
3. Click OK.

6.9.3. Setting the Time by Dragging the Timeline Cursor

The current time point during playback is indicated by a red bar (cursor) that moves along the timeline (Figure 6-1).

- To set the playback time point using the timeline cursor, drag the cursor to any location along the time line. The Current time in the Status bar changes accordingly.

6.10. Viewing a File's Packet Count Histogram

You can display a histogram of the number of packets being sent or recorded by the Logger. When you load a file with the Data View enabled, the Logger calculates the histogram values for the entire file. Then it displays them (Figure 6-1). When you record a simulation with the Data View enabled, the Logger updates the Data View as data is recorded (Figure 6-12).

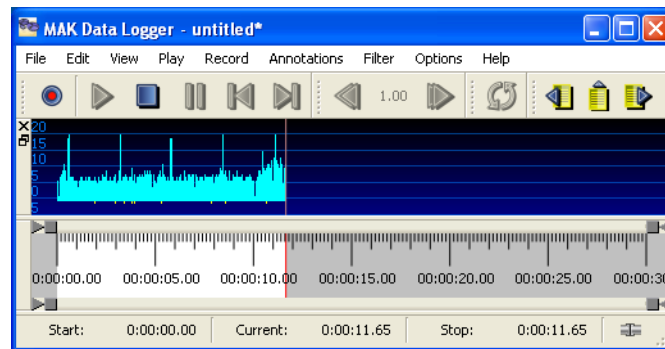


Figure 6-12. Data View in record mode

In the Data View, blue lines (above the centerline) represent the number of object packets; yellow lines (below the centerline) represent the number of interaction packets; red lines (small tick marks along the centerline) represent checkpoints.

i

- ◆ Since the Logger updates the Data View when it loads a file, loading large files with the Data View visible will increase load time considerably.
- ◆ The Data View is completely updated each time you edit a file.

- To show or hide the Data View histogram, choose **View** → **Data View**.

6.11. Viewing Object Statistics

The Logger can display statistics about the objects being published (during playback) or reflected (during recording) by the Logger application. Statistics are displayed in the display area above the Logger timeline (Figure 6-13).

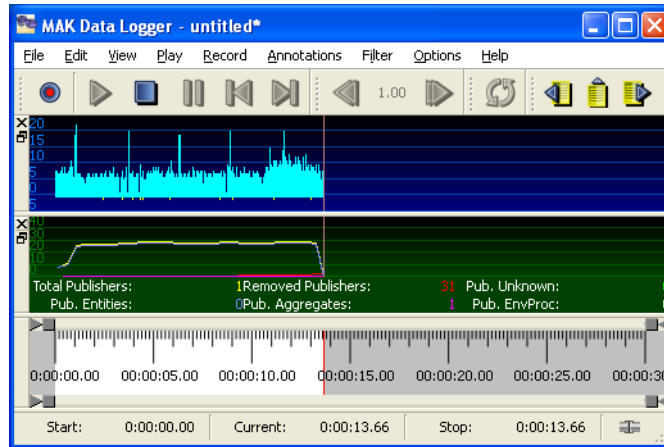


Figure 6-13. Object statistics

- To view or hide object statistics, choose **View** → **Object Statistics**.



Annotating Files

This chapter explains how to add annotations to Logger files.

Annotating a Logger File.....	7-2
Adding an Annotation.....	7-2
Saving an Annotation File	7-4
Opening an Annotation File.....	7-5
Importing an Annotation File.....	7-5
Selecting an Annotation	7-5
Editing an Annotation.....	7-5
Deleting an Annotation.....	7-6
Moving the Timeline Cursor to the Start of an Annotation	7-6
Jumping to the Next or Previous Annotation.....	7-7
Zooming the Timeline to an Annotation.....	7-8
Bounding Playback to an Annotation.....	7-9

7.1. Annotating a Logger File

An annotation is associated with a particular time or time range in a Logger file. Each annotation has a name, and you can associate text with it. Besides being a way to store information about the events in a Logger file, annotations allow you to quickly zoom in on a file or bound playback to a segment of a file. The Annotations View has a context sensitive menu from which you can execute the various annotation commands.

7.1.1. Adding an Annotation

To annotate a Logger file:

1. Choose **View** → **Annotations**. Annotations, if any, are displayed in the Logger data display area (Figure 7-1).

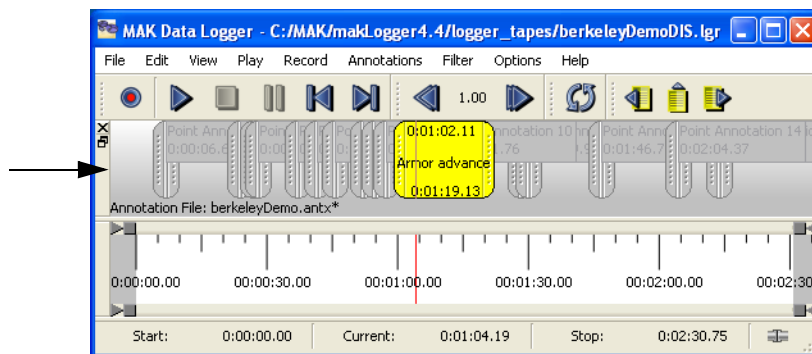


Figure 7-1. Annotations display

2. Choose **Annotations** → **Create** → *annotation_type*, where *annotation_type* type is Point Annotation, Range Annotation, or an instant annotation. (For details about inserting instant annotations, please see “[Adding an Instant Annotation](#),” on page 7-3.) The Point Annotation (or Range Annotation) dialog box opens.

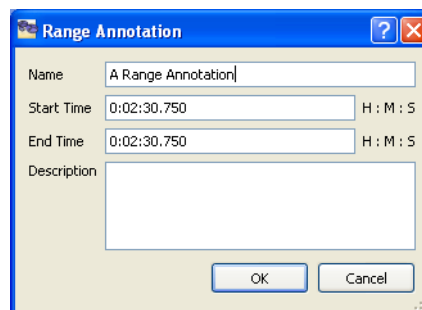


Figure 7-2. Range Annotation dialog box

3. In the Name text box, type a name for the annotation. The name is displayed in the annotation view.

4. Optionally, change the timepoint. For a point annotation, the default time for the annotation is the current timepoint. For a range annotation, the default start time and end time for the annotation's time range is the current timepoint. The time is specified in hour:minute:second.tenth_of_second format.
5. Optionally, in the Description box, enter any text that you want to accompany the annotation.
6. Click OK. The annotation is added to the display area ([Figure 7-1](#)).



If you create an annotation while the Annotations View is hidden, the Annotations View is automatically displayed.

Adding an Instant Annotation

Instant annotations add an annotation at the current time point without requiring you to fill out a dialog box. The annotation time, for point annotations, or start time, for range annotations, is set to the time point. This allows you to quickly add annotations as you play or record a file without needing to stop the file. You would typically go back and edit the annotation at a later time.

Instant range annotations are inserted with identical start and end times.

- To add an instant annotation, choose **Annotations** → **Create** → **Instant Point Annotation** (or **Instant Range Annotation**), or press \ (backslash) or click the Add Point Annotation button () to insert a point annotation, or [(left bracket) to insert a range annotation.
- To set the end time for an instant range annotation, use the Update Annotation feature described in the next section or press the] key at the point you want to end the annotation's range.

Dynamically Resetting an Annotation's Start or End Time (Update)

When you insert annotations while you are recording or playing a file, you may want to adjust their time point. If you insert a range annotation, you will want to set the end point. The Update Annotation command lets you change the start and end times of the currently selected annotation while the Logger is playing or recording.

When you update a range annotation, if the current time is earlier than the start time, the start time is reset. If the current time is later than the start time, the end time is reset.

To update an annotation:

1. Create an annotation or select an annotation while the Logger is recording or playing a file.
2. At the time point to which you want to update the annotation, choose **Annotations** → **Update** or press **]** (right bracket). The appropriate timepoint of the annotation is reset to the time at which you issued the update command. The time-point can be updated to an earlier or later time.

7.1.2. Saving an Annotation File

You can save annotations to a file. Annotation files have the extension *.antx*.

To save annotations to a file:

1. Create annotations as described in [“Adding an Annotation,”](#) on page 7-2.
2. Choose **Annotations** → **Save**. The Save Annotation File dialog box opens.
3. Type a name for the annotation file.
4. Click Save.

7.1.3. Opening an Annotation File

When you open an annotation file, any annotations currently displayed in the Logger window are removed and replaced with the annotations in the file you opened. If you load a file with Play on Load enabled and the file has an annotation file, the annotation file is loading automatically.



You can open annotation files that have been saved in legacy format (*.ant*) and the current format (*.antx*).

To open an annotation file:

1. Choose **Annotations** → **Open**. The Open Annotation File dialog box opens.
2. Select an annotation file.
3. Click Open. The annotations in the file are displayed.

7.1.4. Importing an Annotation File

When you import an annotations file, the annotations are added to any annotations that are already displayed in the Logger window.

To import an annotation file:

1. Choose **Annotations** → **Import**. The Open Annotation File dialog box opens.
2. Select an annotation file.
3. Click Open. The annotations in the file are imported.

7.1.5. Selecting an Annotation

- To select an annotation, choose **Annotations** → **Select** → *annotation*, where *annotation* is one of the annotations in the list, or in the Annotations View, click the annotation.

7.1.6. Editing an Annotation

You can edit the properties of an annotation (name, text, and start and end times).

To edit an annotation:

1. Select the annotation and choose **Annotations** → **Properties**, or double-click the annotation. The Point Annotation or Range Annotation dialog box opens.
2. Edit the properties as desired.
3. Click OK.

Editing a Point Annotation Directly

- To edit the time of a point annotation, drag it to a new time.

Editing a Range Annotation Directly

You can edit the start and end times of a range annotation by dragging its endpoints or dragging the entire annotation.

To edit a range annotation use either or both of the following methods:

- ♦ In the Annotations bar, click and hold the left mouse button on an end point of an annotation and drag it to a new time.
- ♦ In the Annotations bar, click and hold the left mouse button anywhere inside an annotation's range and drag the entire annotation to a new location.

Converting an Annotation to the Other Annotation Type

You can convert point annotations to range annotations and range annotations to point annotations. The time point of a point annotation becomes the start time of the new range annotation. You can then change the end time as described in previous sections. The start point of a range annotation becomes the time point of a point annotation.

To convert an annotation to the other annotation type:

1. Select the point annotation that you want to convert.
2. Choose **Annotations** → **Convert To** → **Range Annotation** (or **Point Annotation**).
3. If you converted a point annotation to a range annotation, specify the end point of the annotation, as described in [“Editing an Annotation,”](#) on page 7-5.

7.1.7. Deleting an Annotation

To delete an annotation:

1. Select the annotation you want to delete.
2. Choose **Annotations** → **Delete**, or press the **Delete** key.

If you remove an annotation from a set of annotations that you have saved to a file, save the edited list with the **File** → **Save** command.

7.1.8. Moving the Timeline Cursor to the Start of an Annotation

You can move the timeline cursor to an annotation or the start time of a range annotation.

To move the timeline cursor to an annotation:

1. Select the annotation.
2. Choose **Annotations** → **Jump to Start**.

Moving the Timeline Cursor to the End of an Annotation

You can move the timeline cursor to the end time of a range annotation.

To move the cursor to the end of an annotation:

1. Select the annotation.
2. Choose **Annotations** → **Jump to End**.

7.1.9. Jumping to the Next or Previous Annotation

You can quickly jump to the start of the next or previous annotation, if there is one.

- To jump to the previous annotation, choose **Annotations** → **Jump to Next**, or click the Jump to Next Annotation button (.
- To jump to the previous annotation, choose **Annotations** → **Jump to Previous**, or click the Jump to Previous Annotation button (.

7.1.10. Zooming the Timeline to an Annotation

You can zoom the timeline view to an annotation. For example, in [Figure 7-3](#), in the first view, the time line shows the entire length of the file and the zoom sliders are fully expanded. After zooming to the Armor Advance annotation, the zoom sliders show the location of the start and end points of the annotation along the zoom bar, the annotation takes up the entire data display, and the timeline is zoomed to the start and end points of the annotation.



Zooming the timeline to an annotation does not change the location of the timeline cursor.

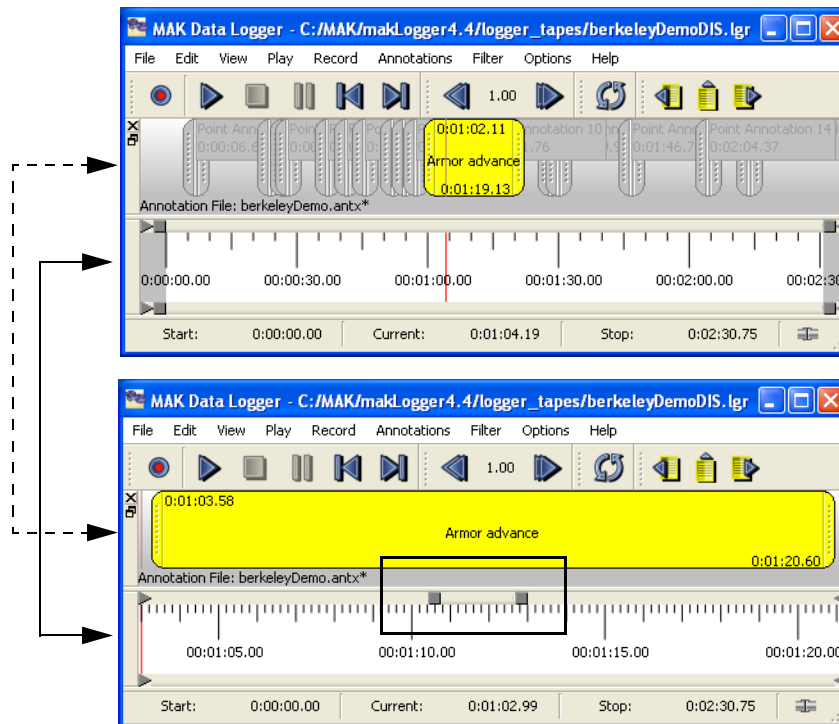


Figure 7-3. Zooming an annotation

To zoom the timeline to an annotation:

1. Select the annotation.
2. Choose **Annotations** → **Zoom to Annotation**. The Annotations bar and the time line zoom to fit the range of the annotation ([Figure 7-3](#)).

7.1.11. Bounding Playback to an Annotation

You can set the playback segment to the range of an annotation. For example, in [Figure 7-4](#), in the first view, the playback handles are at the start and end of the file. After bounding to the annotation, the playback handles are at the start and end points of the annotation. For further comparison, notice that in [Figure 7-3](#), the playback handles are not affected by zooming to the annotation. (They are not visible in the illustration because they are out of the range of the zoomed area.)

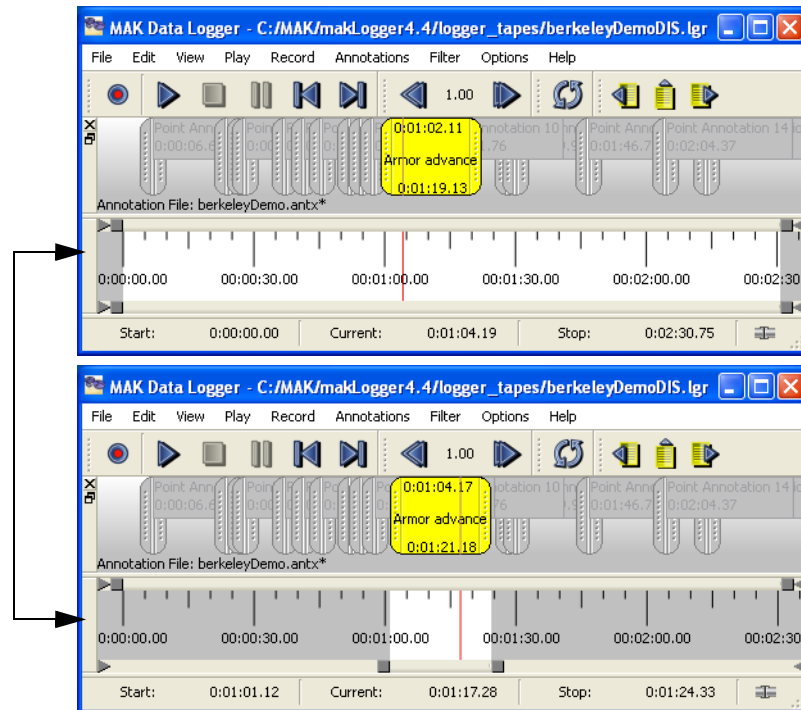
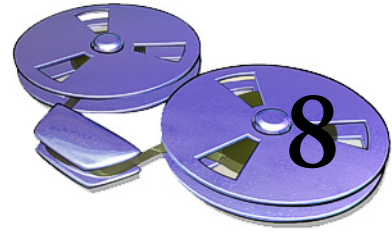


Figure 7-4. Bounding playback to an annotation

To set the playback segment to the range of an annotation:

1. Select the annotation.
2. Choose **Annotations** → **Bound to Annotation**. The playback range changes to match the start and end points of the annotation ([Figure 7-4](#)).



Filtering Input and Output

This chapter explains how to filter input to the Logger during recording and output during playback.

Filtering Logger Input and Output	8-2
Filtering DIS PDUs by Entity Type	8-2
Removing Entity Types from the Entity Types Lists	8-2
Configuring the List of Common Entity Types	8-3
Filtering DIS PDUs by PDU Kind	8-3
Including Custom PDUs in Filter Lists	8-4
Filtering by Entity ID or Object Name	8-5
Filtering HLA Exercises by Class	8-6
Filtering VR-Forces Messages	8-7

8.1. Filtering Logger Input and Output

When you record an exercise or play a file, you can filter the input or output. In DIS, you can filter by entity type, entity ID, or PDU. In HLA, you can filter by class or object name. During a Logger session, filtering settings apply to both file and playback. Settings do not persist between sessions.

8.2. Filtering DIS PDUs by Entity Type

To filter Logger input or output by entity type:

1. Choose **Filter** → **By DIS Entity Type**. The Filter by Entity Type dialog box opens (Figure 8-1). It contains the list of common entity types that is configured in the Logger configuration file or which was specified on the command line when the Logger was started.

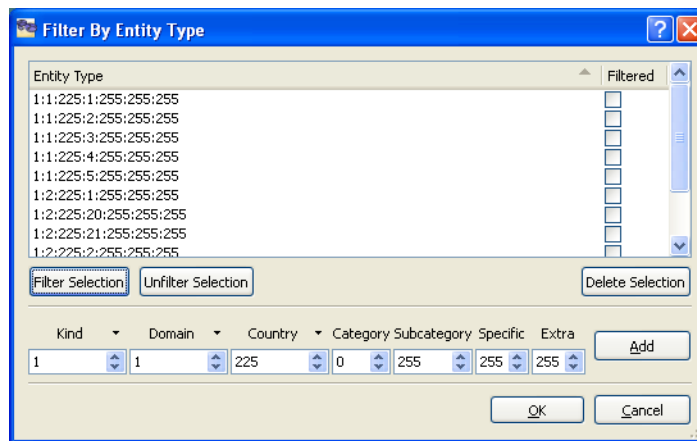


Figure 8-1. Filter By Entity Type dialog box

If the entity type that you want to filter is not listed in the list:

- a. Enter the 7-digit enumeration for the entity type you want to filter in the boxes at the bottom of the dialog box.
 - b. Click Add. The entity type is added to the list.
2. To filter out an entity type, check the box in the Filtered column for that entity.
 3. To process a filtered entity, clear its check box in the Filtered column.
 4. To filter or unfilter multiple entities at the same time, select the entities in the list and click Filter Selection or Unfilter Selection.

8.2.1. Removing Entity Types from the Entity Types Lists

- To remove an entity type from the Entity Types lists, select the entity type and click the Delete Selection button.

8.2.2. Configuring the List of Common Entity Types

You can configure a list of common entity types to populate the Common Entity Types list in the Filter by Entity Type dialog box. To configure a list of entity types, use the `commonEntityTypes` command on the command-line or text interface, or add entries to the `commonEntityTypes` variable in *lgrConfig.xml*, as in the following example:

```
<var name="commonEntityTypes" type="list" listtype="string">
  <!-- US tank -->
  <listitem>1:1:225:1:-1:-1:-1</listitem>
  <!-- US armored vehicle -->
  <listitem>1:1:225:2:-1:-1:-1</listitem>
  <!-- US armored utility vehicle -->
  <listitem>1:1:225:3:-1:-1:-1</listitem>
  <!-- US self-propelled artillery -->
  <listitem>1:1:225:4:-1:-1:-1</listitem>
</var>
```

8.3. Filtering DIS PDUs by PDU Kind

To filter Logger input and output by PDU kind:

1. Choose **Filter** → **By DIS PDU Kind**. The Filter by PDU Kind dialog box opens (Figure 8-2).

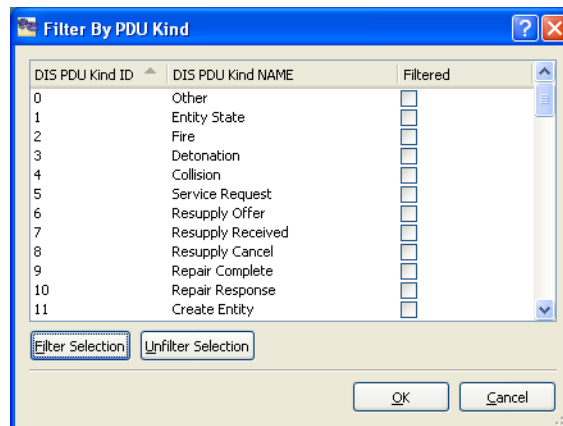


Figure 8-2. Filter By PDU Kind dialog box

2. To change the filtering status of an object, select or clear the check box in the Filtered column for that object.
3. To filter or unfilter multiple objects, select them and click the Filter Selection or Unfilter Selection button.

8.4. Including Custom PDUs in Filter Lists

By default, the Logger lists the PDUs known to VR-Link in the filter lists. If your application uses custom PDUs, you can add them to these lists.

To add custom PDUs to the filter list:

1. Specify a `customPdu` variable in *lgrConfig.xml* for each custom PDU. Each custom PDU must have a unique variable name, as in the following example:

```
<var name="customPDU1" type="list" listtype="string">
  <listitem>MyCustomPdu1</listitem>
  <listitem>186</listitem>
  <listitem>12</listitem>
  <listitem>24</listitem>
</var>

<var name="customPDU2" type="list" listtype="string">
  <listitem>MyCustomPdu2</listitem>
  <listitem>186</listitem>
  <listitem>12</listitem>
  <listitem>24</listitem>
</var>
```

The four fields for each PDU listing are the name of the PDU, the PDU kind, the offset in bytes for the entity ID, and the offset in bytes for the entity type. If there is no entity ID or type, use -1.

2. In the commands section of *lgrConfig.xml*, add each custom PDU to the filter list with an `AddCustomPDU` command, as in the following example:

```
<command domain="Simulation" protocol="DIS" command="AddCustomPDU">
  <param name="pdu" var="customPDU1"/>
</command>

<command domain="Simulation" protocol="DIS" command="AddCustomPDU">
  <param name="pdu" var="customPDU2"/>
</command>
```

8.5. Filtering by Entity ID or Object Name

In DIS, the Logger filters by the site:app:entity triplet. In HLA, it filters by the object name.



To filter by entity ID or object name, Logger files must be in the current Logger format. When you open a Logger file, if you are prompted to upgrade the format, you must do so to use this feature.

To filter messages by entity ID or object name:

1. Open a Logger file.
2. For DIS, choose **Filter** → **By Entity ID**. (For HLA, choose **Filter** → **By Object Name**.) The Filter By Entity ID (or Filter by Object Name) dialog box opens (Figure 8-3). (The dialog boxes for the two protocols are almost identical.)

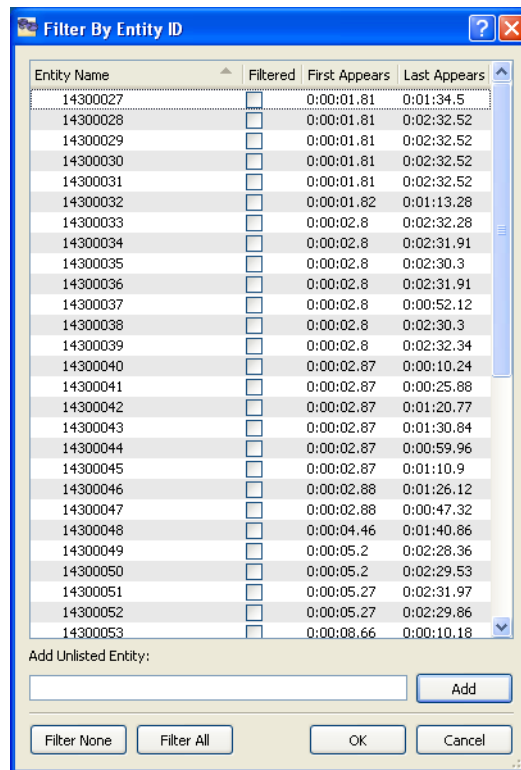


Figure 8-3. Filter by Entity ID dialog box

3. To filter individual entities, select the check box in the Filtered column.
4. To filter all entities, click Filter All.
5. To filter no entities, click Filter None.

6. To add an entity to the list:
 - a. In the Add Unlisted Entity box, type an entity ID that you want to filter.
 - b. Click Add. The entity ID is added to the list.
7. Click OK.

8.6. Filtering HLA Exercises by Class

You can filter HLA exercises by class.

To filter Logger input and output by class:

1. Choose **Filter** → **By Class**. The Filter By Class dialog box opens (Figure 8-4).

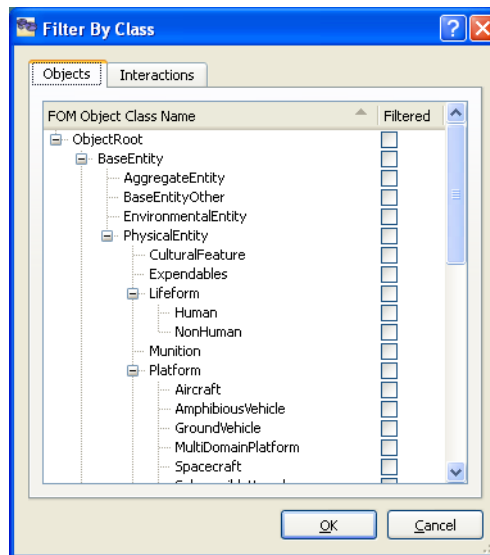


Figure 8-4. Filter By Class dialog box

2. Select the Objects tab to filter object classes. Select the Interactions tab to filter Interaction classes.
3. To change the filtering status of an object, select or clear the check box in the Filtered column for that object.



Select or clearing a parent object selects or clears all of its child objects.

8.7. Filtering VR-Forces Messages

If you are recording a simulation that includes messages sent by VR-Forces, MÄK's computer generated forces product, you should filter out the internal messages that VR-Forces sends between the VR-Forces GUI and simulation engine. These messages are meaningless for playback by the Logger. Filtering is enabled by default for both recording and playback.



Filtering of VR-Forces messages is enabled by default. Changing the default message ID range is not recommended unless you have specific instructions from MÄK or a local developer who has changed the range of message IDs used by a customized version of VR-Forces.

To filter out VR-Forces messages:

1. Choose **Filter** → **VR-Forces Messages**. The Filter VR-Forces Messages dialog box opens (Figure 8-5).



Figure 8-5. Filter VR-Forces Messages dialog box

2. Select the filter VR-Forces Messages check box.
3. Specify the range of message IDs to filter out.
4. Click OK.



Editing Logger Files

This chapter explains how to merge, concatenate, and edit Logger files.

Editing Files.....	9-2
Reverting to the Saved Version of a File.....	9-2
Clearing and Deleting Segments of a File.....	9-2
Merging Files	9-4
Concatenating Files.....	9-4
Trimming Files.....	9-5
Offsetting Files.....	9-5

9.1. Editing Files

The Logger has the following options for editing files:

- ♦ Merge two files
- ♦ Add a file to the beginning or end of another file
- ♦ Clear a portion of a file of data and leave empty space
- ♦ Delete a portion of a file
- ♦ Trim dead space at the beginning of a file
- ♦ Offset the timestamps of packets.



When you edit a file, the changes are not saved automatically. The title bar adds an asterisk to the file name to indicate that it has changed. To retain your edits, you must save the edited file when you are finishing editing it.

9.1.1. Reverting to the Saved Version of a File

If you edit a file and decide that you want to remove all edits, you can revert to the previously saved version of the file. This option is available only for files that have a saved version. You cannot apply it to unsaved, new recordings.

- To revert to the previously saved version of a file, choose **File** → **Revert**.

9.2. Clearing and Deleting Segments of a File

You can select a portion of a file and clear it or delete it. Clearing a portion of a file removes all packets within the selection, but does not change the timestamps of the packets that follow the cleared section. The result is empty space in the file.

If you delete a section of a file, all packets and any associated annotations are removed. The packets that come after the deleted section are offset to fill in the empty space. If a range annotation spans the deleted section its start point or end point is offset to the appropriate amount of time.

[Figure 9-1](#) shows a logger file with the segment from 0:30 to 1:30 selected for editing. If the segment is cleared, the file stays the same length, but there are no messages in the cleared segment. If the segment is deleted, the file is shortened by the amount of time deleted. There is no dead space in the file.

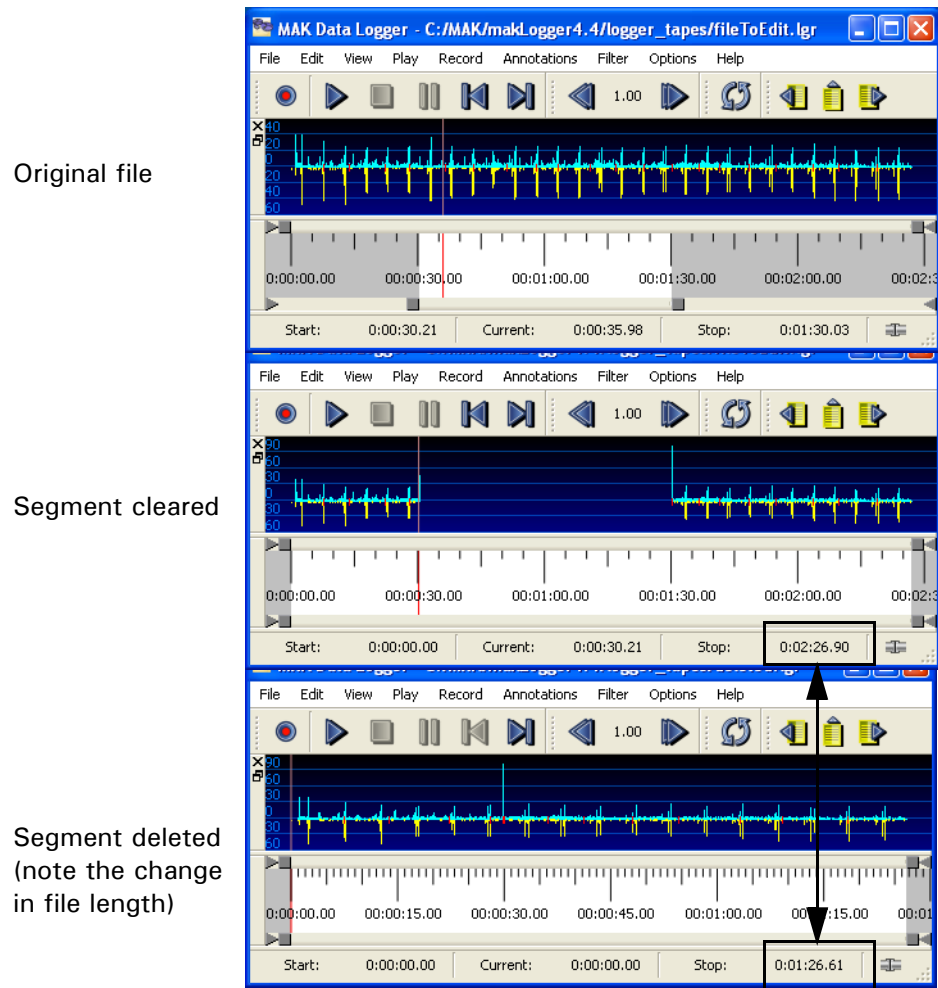


Figure 9-1. Clearing and deleting file segments

The Logger adds data, if necessary, to properly remove or add objects that are affected by a clear or delete operation.



When you delete a portion of a file, it does not get placed on the clipboard. You cannot paste it back into the file.

To clear a portion of a file:

1. Open the file.
2. Use the timeline sliders to select the segment you want to clear.
3. Choose **Edit** → **Clear Section**.

To delete a portion of a file:

1. Open the file.
2. Use the timeline sliders to select the segment you want to delete.
3. Choose **Edit** → **Delete Section**.
4. Save the edited file.

9.3. Merging Files

When you merge files the packets for the file that you are merging in are added to the original file in order of the timestamps. The Logger does not do any checking for duplicate names. It adds new checkpoints using the current checkpoint settings.



To successfully merge HLA files, they must have been created with the same FOM.

To merge two files:

1. Open a file.
2. Choose **Edit** → **Merge In File**. An Open dialog box opens.
3. Select the file that you want to merge in.
4. Click Open. The selected file is merged into the first file.
5. Save the edited file.

9.4. Concatenating Files

You can concatenate (append) a file to the beginning or end of another file.



To successfully concatenate HLA files, they must have been created with the same FOM.

To concatenate files:

1. Open a file.
2. Choose **Edit** → **Append File to End** or **Prefix File to Start**. An Open dialog box opens.
3. Select the file that you want to concatenate.
4. Click Open. The selected file is concatenated to the open file.
5. Save the edited file.

9.5. Trimming Files

When you trim a file, empty space at the beginning is removed and packets are offset so that the first packet is sent at the beginning of the file.

To trim a file:

1. Open the file.
2. Choose **Edit** → **Trim Start**.
3. Save the edited file.

9.6. Offsetting Files

When you offset a file, a specified number of seconds is added to the time stamp of each packet. (This results in dead space at the start of the file.)

To offset a file:

1. Open the file.
2. Choose **Edit** → **Offset Time**. The Set Offset dialog box opens.
3. Type the time, in seconds, that you want to offset the file.
4. Click OK.
5. Save the edited file.



Exporting Logger Data

This chapter explains how to export Logger files to various data formats.

Exporting Data	10-2
Exporting Data to a Database	10-3
Configuring an ODBC Data Source	10-5
Exporting DIS Data to a Database	10-6
Exporting HLA Data to a Database.....	10-7
The Export Log.....	10-7
Configuring a Data Source for Microsoft Access	10-8
Configuring a Data Source for MySQL.....	10-11
Importing Logger Data into a Spreadsheet	10-14
Importing Data from MySQL to Microsoft Excel.....	10-14
Importing Data from Access to Excel.....	10-18

10.1. Exporting Data

You can export some or all of the data in a Logger file to another supported format.



- ♦ You cannot export data while you are in Record mode.
- ♦ Before you can export data to a database, you must set up a data source. For details, please see “[Configuring an ODBC Data Source](#),” on page 10-5, “[Configuring a Data Source for Microsoft Access](#),” on page 10-8, and “[Configuring a Data Source for MySQL](#),” on page 10-11.

To export data:

1. Open a Logger file.
2. Choose **File** → **Export to** → *format*, where *format* is a supported format, such as Plain Text. A dialog box appropriate to the export type opens.
3. To export to another Logger file or to text, type a name for the new file and click OK. For other export types, please see the additional procedures in this chapter.

To export part of a Logger file:

1. Select the portion of a Logger file that you want to export.
2. Choose **File** → **Export Selection To** → *format*.
3. Continue as in step 3 of the previous procedure.

10.2. Exporting Data to a Database

The Logger can export data to a database using ODBC. Please see *MÄK Data Logger Release Notes* for a list of ODBC drivers supported. The Logger creates tables that contain the messages that it can decode. Optionally, you can create an object instance table and a checkpoint information table. You can apply filters to the data when it is exported.

The Logger uses underscores to separate words used as table and row names. Each PDU or HLA class has its own table. Variable length data is stored as indexed data in a subtable that contains the parent table's name followed by the variable's name. For example, if the parent table is aggregateentity, the subtable might be aggregateentity_silentaggregates.

Notes

- ♦ Before you can export data to a database, you must set up a data source. For details, please see [“Configuring an ODBC Data Source,”](#) on page 10-5, [“Configuring a Data Source for Microsoft Access,”](#) on page 10-8, and [“Configuring a Data Source for MySQL,”](#) on page 10-11.
- ♦ You do not have to purchase Microsoft Access to set up a data source for this format. For details, please see [“Configuring a Data Source for Microsoft Access,”](#) on page 10-8.
- ♦ When the Logger exports data to an existing database, it only overwrites tables for which it is exporting data. If the database has existing tables for which there is no equivalent in the simulation being exported, the resulting database will contain data that does not represent any actual simulation. For example if you export a simulation that contains detonation interactions to database myLogger and then you export a simulation that does not have detonation interactions to myLogger, the detonation interaction data from the first simulation will still be in the database.

It is your responsibility to ensure that data is exported to a clean database.

- ♦ The Logger does not have to be connected to an exercise to export data. However if you use a custom filter, be sure that the filter does not require the Logger to be connected.

To export data to a database:

1. Open the file you want to export or stop a current recording and enter playback mode.
2. Optionally, select a segment of the file using the timeline sliders.
3. Choose **File** → **Export to** → **Database** or **File** → **Export Selection To** → **Database**. The ODBC Export Settings dialog box opens ([Figure 10-1](#) and [Figure 10-2](#)).

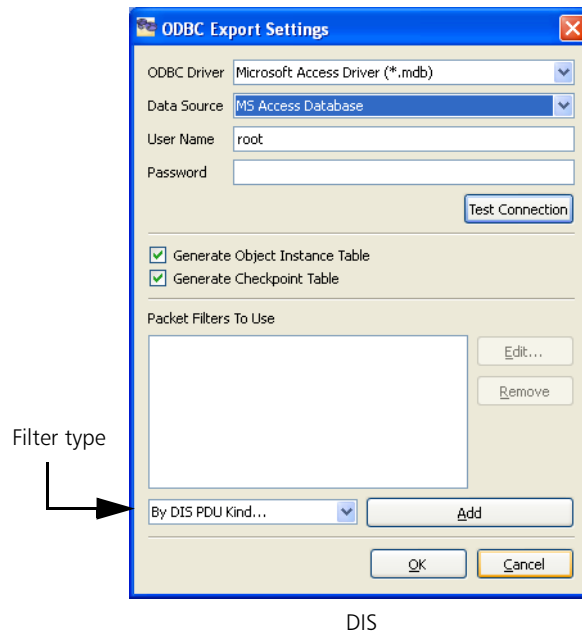


Figure 10-1. ODBC Export Settings dialog box (DIS)

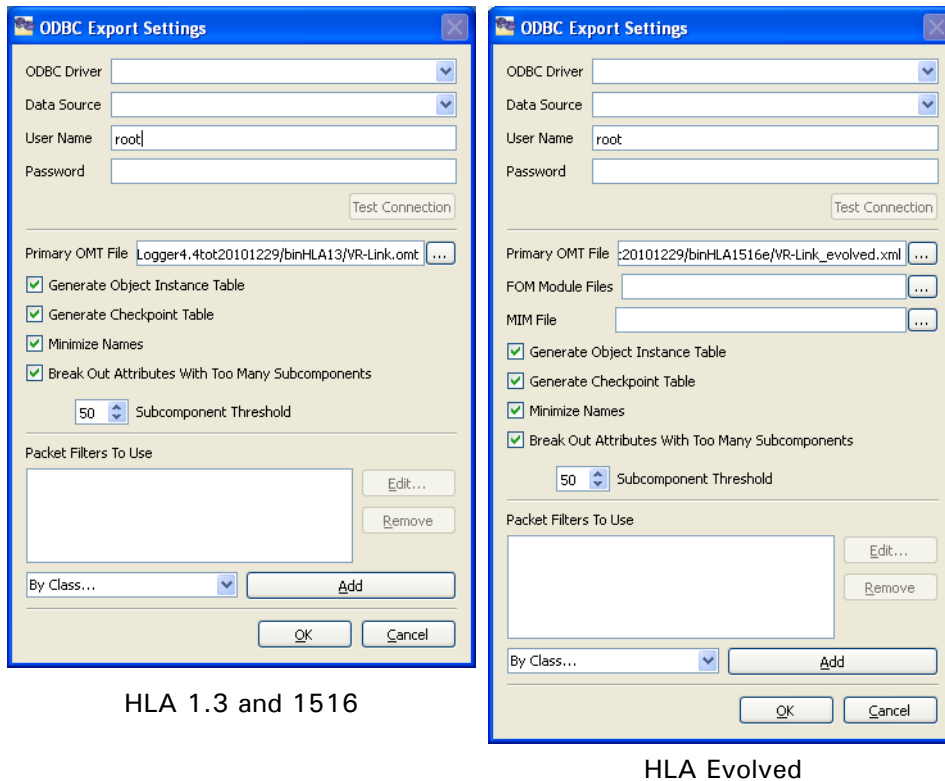


Figure 10-2. ODBC Export Settings dialog box (HLA)

4. Select an ODBC driver in the ODBC Driver list.
5. Select a data source from the Data Source list.
6. For export to MySQL, enter the user name and password for the data source.
7. Optionally, click Test Connection to test the connection to the data source.
8. For HLA Evolved only, optionally add FOM Modules and a MIM file.

The FOM Module Files entry is a list of files separated by semi-colons. The list can be modified directly or populated using the the FOM Module browse button. It selects a single file but it is additive. Each time it is used, the selected file is added to the contents of the FOM Modules entry. If existing values need to be removed, the entry must be edited directly.

9. Optionally, check or clear the check boxes for the optional tables.
10. Optionally, for HLA export, change the threshold for breaking up attributes into subtables or disable this feature. (RPR FOM 1.0 files should not need to do this, but RPR FOM 2.0, draft 17 files do. For more information, please see [“Exporting HLA Data to a Database,”](#) on page 10-7.)
11. Optionally, specify a filter to use while exporting data, as follows:
 - a. Select a filter type from the drop-down list.
 - b. Click Add.
 - c. Select the filter in the Packet Filters to Use list.
 - d. Click Edit.
 - e. Edit the list of classes to filter. For details about editing filter lists, please see [“Filtering Logger Input and Output,”](#) on page 8-2.
12. Click OK.

10.2.1. Configuring an ODBC Data Source

You must set up a data source before you open the Export To Database dialog box.

On Unix systems you must edit configuration files. GUIs vary depending on the operating system. The Logger uses the iODBC library to provide ODBC capability. Please see the following web site for details about configuring ODBC:

<http://www.iodbc.org/index.php?page=docs/odbcstory>

In Windows you create and configure data sources using the ODBC Data Source Administrator, as described later in this chapter.

10.2.2. Exporting DIS Data to a Database

The Logger supports export to database of the following DIS PDUs:

- ♦ Aggregate State
- ♦ Collision
- ♦ Designator
- ♦ Detonation
- ♦ Electronic Emission
- ♦ Entity State
- ♦ Fire
- ♦ Receiver
- ♦ Signal
- ♦ Start
- ♦ Stop
- ♦ Transmitter
- ♦ Unknown.

You can use the plug-in API to write additional PDU encoders, if needed.

10.2.3. Exporting HLA Data to a Database

The Logger can export data from any FOM. If you are using HLA Evolved, it can export data from any FOM module. This is done by processing the FOM's OMT or XML file. One of these files is required to export any HLA data. If the OMT or XML file is poorly formed, incomplete, or ambiguous, or if a data format does not map conceptually to a database table/row format, the Logger might not be able to decode all data. However, the generic implementation handles the majority of data.

The Logger supports export of the following HLA data:

- ♦ Simple Data Types
- ♦ Complex Data Types
- ♦ Array Data (XML FOMs only)
- ♦ Fixed Cardinality Data

There is no defined HLA database schema. The schema is dynamically generated using the OMT file. HLA class names can be quite long. The Logger has an option to use the shortest unique name possible. For example, instead of using `HLAObject_PhysicalObject_Platform_Aircraft` it can use `Aircraft`. If there is name collision, each immediate parent class is prefixed to the name until a unique name is found.

When exporting to an Access database, there is a limit of 255 fields for any given table. It is possible that the number of fields generated for a class will exceed this limit. As an example, the RPR FOM 2.0017 `BaseEntity.PhysicalEntity.Platform` subclasses generate over 400 fields, not including the variable length attributes that are already broken out into subtables. The Logger can export these classes by breaking out additional attributes and parameters into subtables. It uses the number of fields an attribute will generate (based on the data type subcomponents) as a trigger. When the number of subcomponents exceeds a specified threshold, the logger can put that attribute into a separate subtable similar to the variable length data attributes. You can configure whether this threshold is considered and the number of subcomponents it takes in the ODBC Export Settings dialog box.

10.2.4. The Export Log

When you export Logger data, the Logger writes information about the export process to `sql_export.txt` in the `./bin` directory from which you are running the Logger. The log file is important because it identifies the data that cannot be exported.

10.3. Configuring a Data Source for Microsoft Access

If you want to export data to Microsoft Access, you must configure a data source. You do not have to have Access installed to complete this procedure. (Of course, you will not be able to manipulate the database without Access installed. However, you can open the database in Microsoft Excel, as described in “Importing Data from Access to Excel,” on page 10-18.)

To configure an ODBC Data Source for Microsoft Access in Windows XP:

1. Open the Control Panel.
2. Double-click Administrative Tools. (It is part of the Performance and Maintenance group if your Control Panel groups applets.) The Administrative Tools folder opens (Figure 10-3).

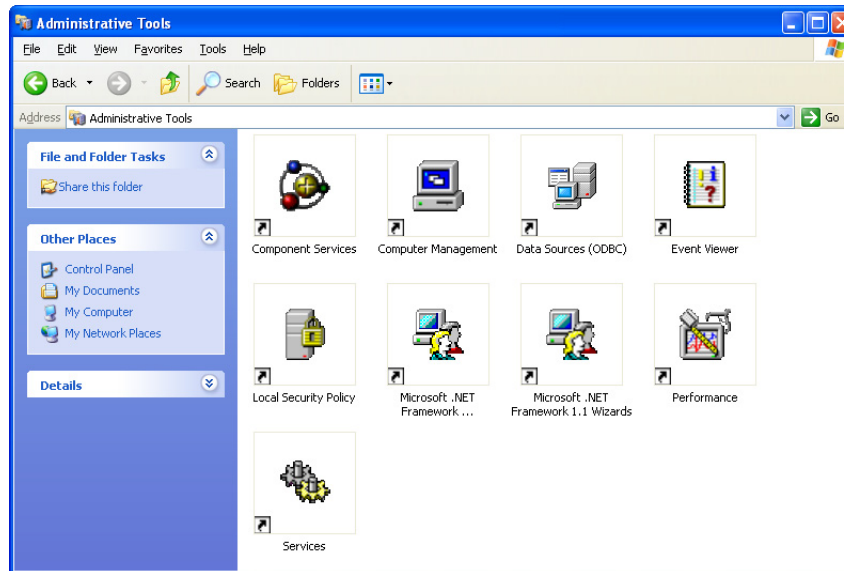


Figure 10-3. Administrative Tools applet window

3. In the Administrative Tools folder, double-click Data Sources (ODBC). The ODBC Data Source Administrator dialog box opens (Figure 10-4).

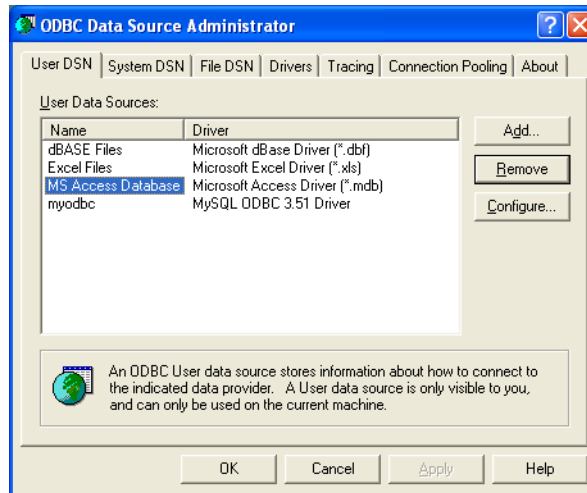


Figure 10-4. ODBC Data Sources Administrator

4. Select the System DSN tab.
5. Click Add. The Create New Data Source window opens (Figure 10.2.1).

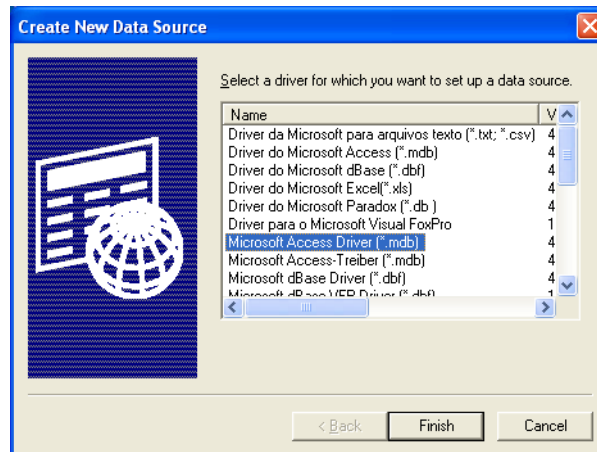


Figure 10-5. Create New Data Source window

6. Select Microsoft Access Driver (*.mdb).

- Click Finish. The ODBC Microsoft Access Setup dialog box opens (Figure 10-6).

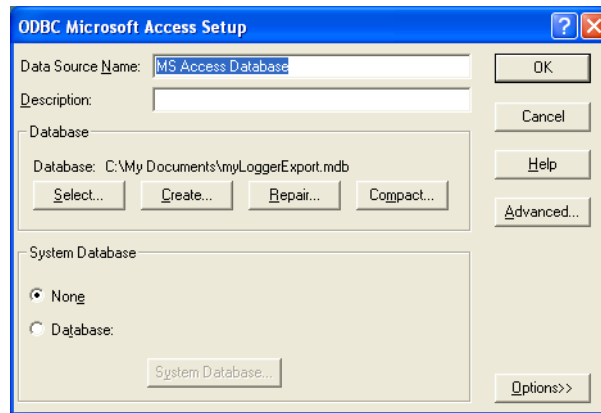


Figure 10-6. ODBC Microsoft Access Setup dialog box

- In the Data Source Name box, type a name for the data source.
- In the Database group box, click Create. The New Database dialog box opens (Figure 10-7).

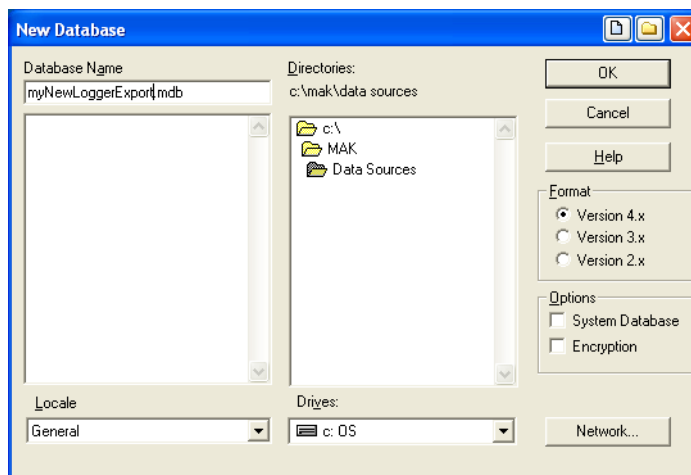


Figure 10-7. New Database dialog box

- In the Database Name box, type a name for the database, for example, *myNewLoggerExport.mdb*.
- In the Directories list, select the directory where you want to create the database.

12. Click OK. An informational prompt advises you that the database has been created (Figure 10-8).

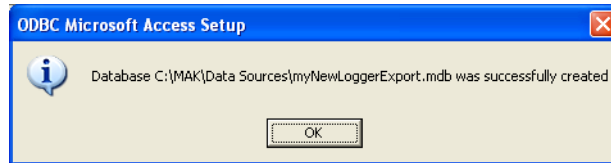


Figure 10-8. Information prompt

13. Click OK.
14. Continue clicking OK to back out of the dialog box sequence.

The next time you export a Logger file to database using the Microsoft Access driver and data source, it is exported to the database file you created in this procedure.

10.4. Configuring a Data Source for MySQL

This section explains how to configure a data source for MySQL on Windows XP. Before you can configure a data source, you must install MySQL and the MySQL ODBC driver and you must create a MySQL database to configure as the data source. Please consult your MySQL documentation for an explanation of how to do this. The procedure in this section was developed using MySQL 5.0.

To configure an ODBC data source for MySQL:



If you are running the Logger on 64-bit Windows, run *C:/Windows/SysWOW64/odbcad32.exe*. Then go to step 4.

1. Open the Control Panel.
2. Double-click Administrative Tools. The Administrative Tools folder opens (Figure 10-3).

3. In the Administrative Tools folder, double-click Data Sources (ODBC). The ODBC Data Source Administrator dialog box opens (Figure 10-9).

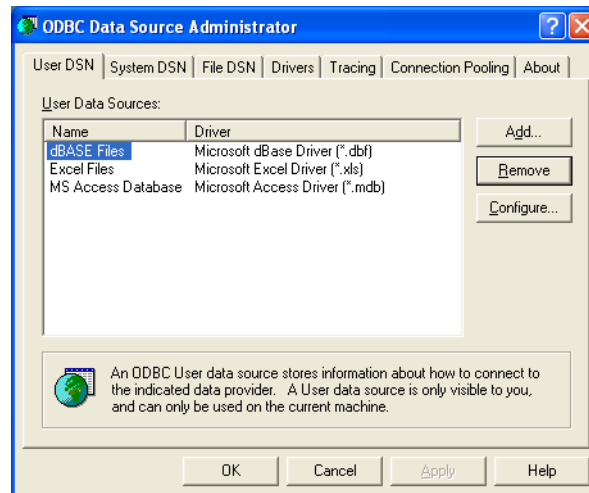


Figure 10-9. ODBC Data Source Administrator dialog box

4. Click Add. The Create New Data Source dialog box opens (Figure 10-10).

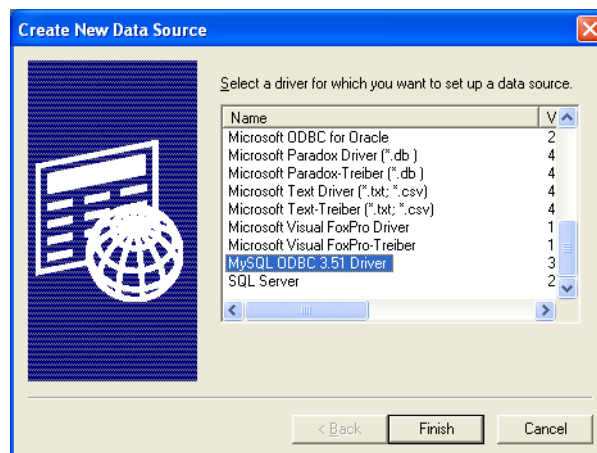


Figure 10-10. Create New Data Source dialog box

5. Select the MySQL ODBC driver.
6. Click Finish. The Connector/ODBC *version* - Add Data Source Name dialog box opens.

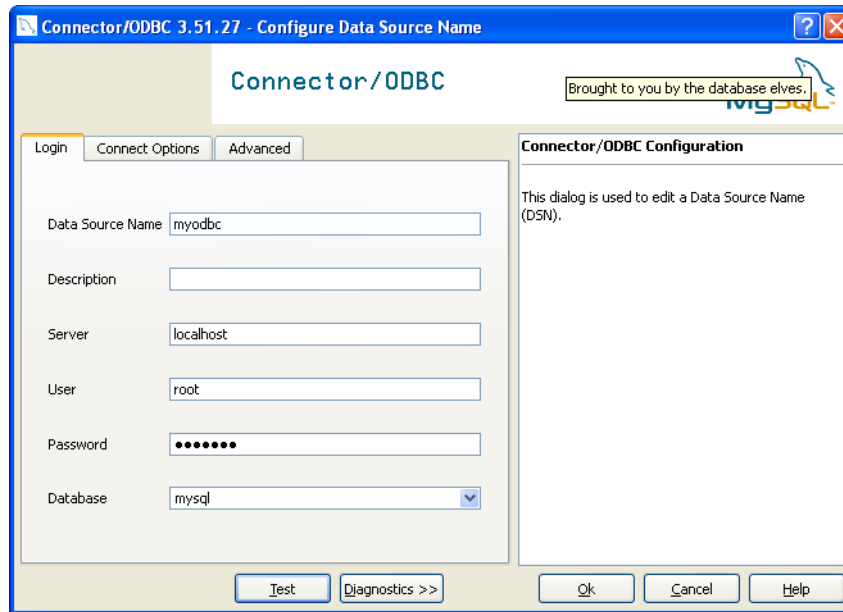


Figure 10-11. Connector ODBC *version* - Add Data Source Name dialog box

7. Type a name for your data source.
8. Complete the Server, User, and Password fields.
9. Enter the name of the database that you created for Logger data.
10. Click OK. The new data source is displayed in the ODBC Data Source Administrator dialog box (Figure 10-12).

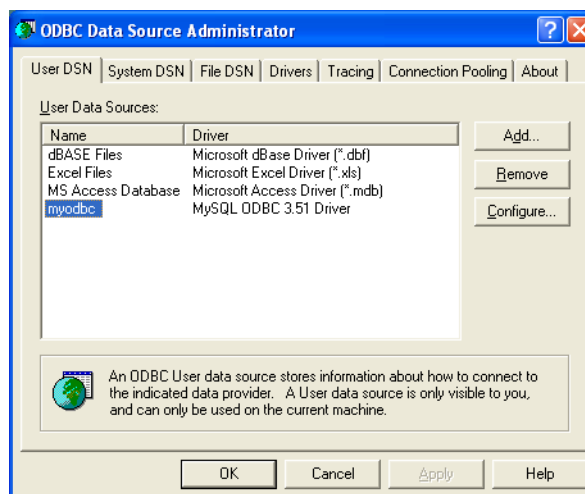


Figure 10-12. ODBC Data Source Administrator dialog box

10.5. Importing Logger Data into a Spreadsheet

After you export Logger data to a database, you can examine and manipulate the data just as you would any database. One common way to view the data is to import it into a spreadsheet and use the spreadsheet's tools to analyze it, graph it, and so on.

10.5.1. Importing Data from MySQL to Microsoft Excel

This section explains how to import data stored in a MySQL database into Microsoft Excel. (This procedure uses Microsoft Excel 2010. Earlier versions may have slightly different menu options and dialog box names.)

To import data from a MySQL database to Microsoft Excel:

1. Start Excel.
2. Choose **Data** → **From Other Sources** → **From Microsoft Query**. The Choose Data Source dialog box opens ([Figure 10-13](#)).

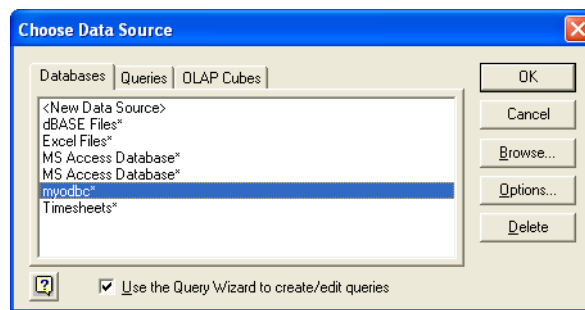


Figure 10-13. Choose Data Source dialog box

3. In the window, select the data source from which you want to import data.
4. Click OK. The Query Wizard - Choose Columns dialog box opens ([Figure 10-14](#)). In the Available Tables and Columns window is a list of tables in the database. You can expand each table for a list of the columns in the table.

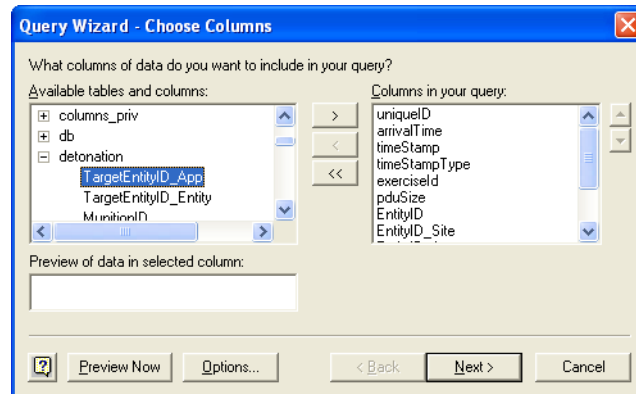


Figure 10-14. Query Wizard - Choose Columns

5. Select the tables and columns that you want to import to Excel.
6. Click the right-facing arrow to move the selected tables and columns to the Columns In Your Query window.
7. Click Next. The Query Wizard - Filter Data dialog box opens ([Figure 10-15](#)).

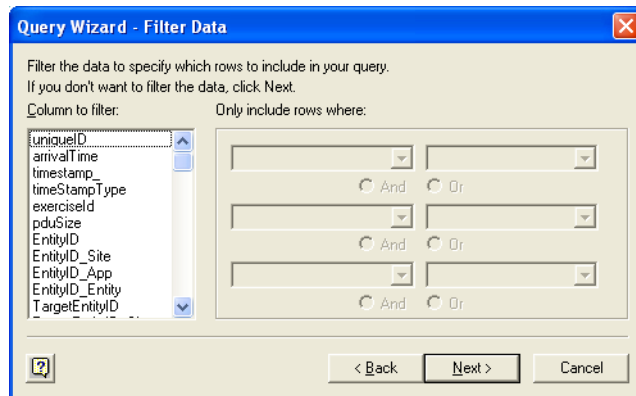


Figure 10-15. Query Wizard - Filter Data

8. Set up any filtering that you want, then click Next. The Query Wizard - Sort Order dialog box opens (Figure 10-16).

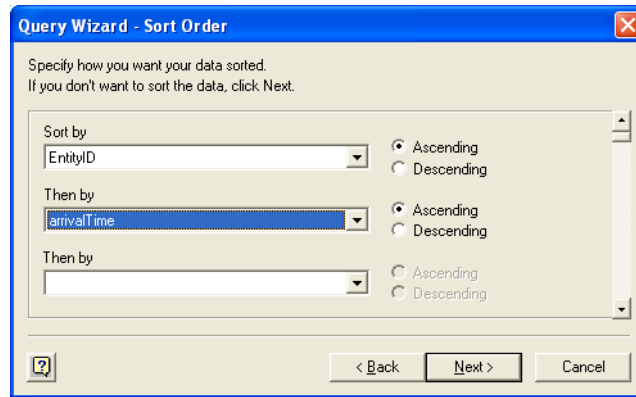


Figure 10-16. Query Wizard - Sort Order

9. Set up any sorting that you want, then click Next. The Query Wizard - Finish dialog box opens (Figure 10-17).

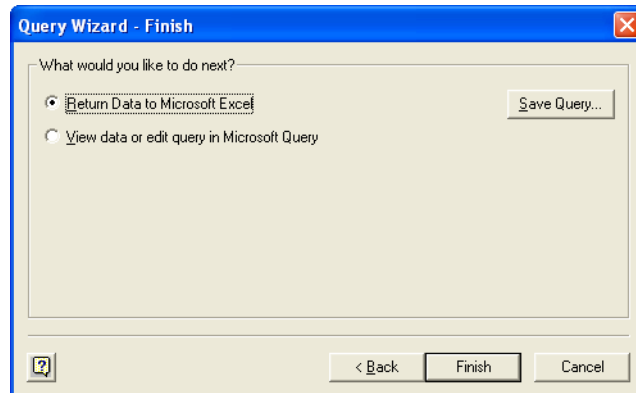


Figure 10-17. Query Wizard - Finish

10. Select the Return Data to Microsoft Excel option and click Finish. The Import Data dialog box opens and a cell in the spreadsheet is highlighted (Figure 10-18).

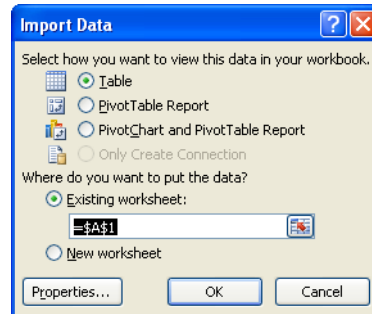


Figure 10-18. Import Data dialog box

11. Specify where you want the data to be imported to and click OK. The data is imported into the spreadsheet (Figure 10-19).

	A	B	C	D	E	F	G	H	I	J	K	L
	uniqueID	arrivalTime	timeStamp	timeStampType	exerciseId	pduSize	EntityID	EntityID_Site	EntityID_App	EntityID_Entity	TargetEntityID	TargetEntityID_Site
2	69	124.9313801	1516.823423	Relative		1	104 1:3006:10		1	3006	10 1:3006:8	
3	1	9.036422491	484.1252424	Relative		1	104 1:3009:236		1	3009	236 0:0:0	
4	2	9.572120786	484.6636422	Relative		1	104 1:3009:237		1	3009	237 0:0:0	
5	3	10.26580024	485.3323383	Relative		1	104 1:3009:238		1	3009	238 0:0:0	
6	5	27.98223853	503.0621443	Relative		1	104 1:3009:240		1	3009	240 1:3009:244	
7	6	28.32178533	503.34989	Relative		1	104 1:3009:244		1	3009	244 1:3009:240	
8	7	30.36473632	505.4434078	Relative		1	104 1:3009:244		1	3009	244 1:3009:240	
9	8	30.49501395	505.5854509	Relative		1	104 1:3009:244		1	3009	244 1:3009:240	
10	11	48.36644602	523.4575326	Relative		1	104 1:3009:259		1	3009	259 1:3009:263	
11	16	54.01944625	529.0493382	Relative		1	104 1:3009:259		1	3009	259 1:3009:243	
12	17	57.6967212	532.7884549	Relative		1	104 1:3009:260		1	3009	260 0:0:0	
13	18	58.05586541	533.096401	Relative		1	104 1:3009:260		1	3009	260 1:3009:292	
14	19	63.01631296	538.1087054	Relative		1	104 1:3009:260		1	3009	260 0:0:0	
15	21	67.8188505	542.8861152	Relative		1	104 1:3009:260		1	3009	260 1:3009:285	
16	24	72.61590338	547.6666298	Relative		1	104 1:3009:260		1	3009	260 1:3009:321	
17	61	103.1997027	578.246644	Relative		1	104 1:3009:260		1	3009	260 0:0:0	
18	15	52.33607841	527.398162	Relative		1	104 1:3009:261		1	3009	261 1:3009:263	
19	12	48.47135711	523.5633356	Relative		1	104 1:3009:262		1	3009	262 0:0:0	
20	20	63.72245657	538.7721229	Relative		1	104 1:3009:262		1	3009	262 0:0:0	
21	28	76.04163206	551.0707939	Relative		1	104 1:3009:262		1	3009	262 0:0:0	
22	29	77.38084757	552.4209945	Relative		1	104 1:3009:284		1	3009	284 1:3009:296	
23	33	88.47812307	563.5172797	Relative		1	104 1:3009:286		1	3009	286 1:3009:296	
24	34	89.05890357	564.1078267	Relative		1	104 1:3009:286		1	3009	286 1:3009:296	
25	38	93.86607313	568.9350838	Relative		1	104 1:3009:286		1	3009	286 1:3009:296	
26	40	94.44691432	569.5267808	Relative		1	104 1:3009:286		1	3009	286 1:3009:296	

Figure 10-19. Spreadsheet with data imported

10.5.2. Importing Data from Access to Excel

This section explains how to import data from a Microsoft Access database to Microsoft Excel.

To import data from Access to Excel:

1. Create an MS Access data source as described in “[Configuring a Data Source for Microsoft Access](#),” on page 10-8.
2. Export a Logger file to Access, as described in “[Exporting Data to a Database](#),” on page 10-3.
3. Start Excel.
4. Choose **File** → **Open**.
5. Set Files of Type to Access Databases.
6. Select the database that you want to open.
7. Click Open. You may receive a warning prompt. If so, click Open. The Select Table dialog box opens ([Figure 10-20](#)).

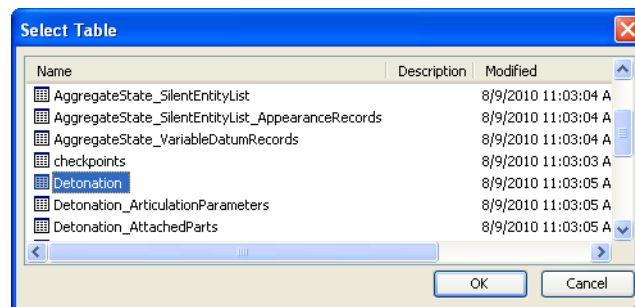
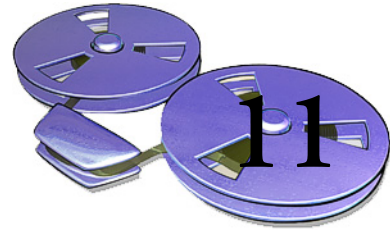


Figure 10-20. Select Table dialog box

8. Select the table that you want to import.
9. Click OK. The data is imported ([Figure 10-21](#)).

	A	B	C	D	E	F	G	H	I	J	K	L
	uniqueID	arrivalTime	timestamp	timestampType	exerciseld	pduSize	EntityID	EntityID_Site	EntityID_App	EntityID_Entity	TargetEntityID	TargetEntityID
1	1	8.380643491	996.1273419	Relative	1	104	1:100:1	1	100	1	1:100:18	
2	2	11.60724855	999.3540158	Relative	1	104	1:100:1	1	100	1	0:0:0	
3	3	15.56175173	1003.308482	Relative	1	104	1:100:1	1	100	1	0:0:0	
4	4	20.67284286	1008.419674	Relative	1	104	1:100:1	1	100	1	0:0:0	
5	5	24.01634789	1011.762409	Relative	1	104	1:100:1	1	100	1	1:100:19	
6	6	27.1227876	1014.862741	Relative	1	104	1:100:1	1	100	1	0:0:0	
7	7	30.86653424	1018.613292	Relative	1	104	1:100:1	1	100	1	0:0:0	
8	8	31.82572151	1019.572516	Relative	1	104	1:100:5	1	100	5	0:0:0	
9	9	42.4714067	1030.218185	Relative	1	104	1:100:5	1	100	5	0:0:0	
10	10	43.31518078	1031.040503	Relative	1	104	1:100:25	1	100	25	1:100:10	
11	11	43.8990171	1031.624819	Relative	1	104	1:100:25	1	100	25	1:100:10	
12	12	44.4888475	1032.218209	Relative	1	104	1:100:25	1	100	25	1:100:10	
13	13	45.06844807	1032.806646	Relative	1	104	1:100:25	1	100	25	1:100:10	
14	14	45.44996822	1033.196238	Relative	1	104	1:100:5	1	100	5	1:100:28	
15	15	45.64600456	1033.377359	Relative	1	104	1:100:25	1	100	25	1:100:10	
16	16	45.90158332	1033.627712	Relative	1	104	1:100:26	1	100	26	1:100:10	
17	17	46.22398591	1033.955342	Relative	1	104	1:100:25	1	100	25	1:100:10	
18	18	46.48613954	1034.215493	Relative	1	104	1:100:26	1	100	26	1:100:10	
19	19	46.80799914	1034.539046	Relative	1	104	1:100:25	1	100	25	1:100:10	
20	20	47.06128657	1034.799485	Relative	1	104	1:100:26	1	100	26	1:100:10	
21	21	47.40133441	1035.13281	Relative	1	104	1:100:25	1	100	25	1:100:10	
22	22	47.6613375	1035.392934	Relative	1	104	1:100:26	1	100	26	1:100:10	
23	23	47.91994513	1035.647663	Relative	1	104	1:100:25	1	100	25	1:100:10	
24	24	48.2404126	1035.971101	Relative	1	104	1:100:26	1	100	26	1:100:10	
25	25	48.51227808	1036.241489	Relative	1	104	1:100:25	1	100	25	1:100:10	
26	26	48.84774912	1036.574972	Relative	1	104	1:100:26	1	100	26	1:100:10	
27	27	49.08392513	1036.829674	Relative	1	104	1:100:25	1	100	25	1:100:10	
28	28	49.43187046	1037.163068	Relative	1	104	1:100:26	1	100	26	1:100:10	
29	29	49.68186498	1037.413177	Relative	1	104	1:100:25	1	100	25	1:100:10	
30	30	50.00922072	1037.740589	Relative	1	104	1:100:26	1	100	26	1:100:10	
31	31	50.27111676	1038.001028	Relative	1	104	1:100:25	1	100	25	1:100:10	

Figure 10-21. Logger file data imported from Access to Excel



Using Logger Commands

The Logger has a set of commands that cover most Logger operations. They are very versatile in that you can use them on the command line, in the text interface, or in the Logger configuration file. This chapter explains how to use the commands in each of these contexts. It also describes each command.

Using Logger Commands on the Command Line.....	11-2
Using Logger Commands in the Configuration File.....	11-2
Using Logger Commands in the Text Interface	11-3
Entering Commands	11-3
Displaying a List of Commands	11-4
Command Descriptions	11-4
Argument Types	11-5
System Domain Commands.....	11-6
Player Domain Commands	11-7
Recorder Domain Commands.....	11-8
Simulation Domain Commands.....	11-10
Editor Domain Commands.....	11-17
Command List.....	11-18

11.1. Using Logger Commands on the Command Line

When you start the Logger, you can use the `-r` command-line option to specify Logger commands. Commands must be surrounded by double quotes. To use double quotes within a command, protect them with a backslash escape character (`\`). Strings in string lists are separated by spaces. Strings cannot contain spaces.



Commands that are specified using `-r` are executed after the Logger connects to the exercise. Therefore, commands that need to be executed at connection time, such as `SetConnectionParameters`, do not work from the command line.

The following example starts the Logger and opens the file `../logger_tapes/berkeleyDemoHLA13.lgr`. Note the use of the backslash to protect the quoted string within the command. (Quotes are not required unless the path has spaces.)

```
logger -r "OpenFile  
  \"../logger_tapes/berkeleyDemoHLA13.lgr\" 0"
```



The `OpenFile` command is unique. If you simply specify a Logger file on the command line, the Logger assumes `-r OpenFile` and opens the file, for example:

```
logger "../logger_tapes/berkeleyDemoHLA13.lgr"
```

The following example enables time management and specifies a subscription list:

```
logger -n 3 --federateName logger  
  -r "SetTimeManagement true true true true 0.1"  
  -r "SetSubscriptionLists \"ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft  
    ObjectRoot.BaseEntity.PhysicalEntity.Radio InteractionRoot.WeaponFire\""
```

11.2. Using Logger Commands in the Configuration File

You can put Logger commands in the Logger configuration file. For complete details, please see [“Editing lgrConfig.xml,”](#) on page 5-18.

11.3. Using Logger Commands in the Text Interface

The Logger text interface lets you enter commands in a console window instead of, or in addition to, using the graphical user interface.

- To start the Logger with the console enabled, in the *lgrConfig.xml* file, set the value of the `showConsole` variable to true.

```
<var name="showConsole" type="bool" value="true" />
```

or start the Logger with the `-c` command-line option:

```
logger -c 1
```

11.3.1. Entering Commands

The text interface has commands for most Logger functions. Most commands consist of a command and one or more arguments, for example:

```
OpenFile ../logger_tapes/test.lgr 0
```

Commands must include all arguments and the arguments must be typed in the correct order. (For command descriptions, please see [“Command Descriptions,”](#) on page 11-4.) For example, in the `SetTimeManagement` command, you must include values for each time management mode, even if you only want to use one of them:

```
SetTimeManagement true true true true 0.1
```

String list parameters are encoded as quoted strings separated by spaces. Strings in a string list may not have spaces, for example:

```
SetSubscriptionLists "ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft" "ObjectRoot.BaseEntity.PhysicalEntity.Radio" "InteractionRoot.WeaponFire"
```

You can manipulate the command line as follows:

- To execute a command, press **Enter**.
- To reset a command entry, press **Escape**.
- To execute a previously executed command, press the **Up Arrow** key until the command is displayed. Then, press **Enter**.
- If you have displayed a previous command by pressing **Up Arrow**, press **Down Arrow** to move back down the command list.

11.3.2. Displaying a List of Commands

You can display a list of commands. They are displayed in the format:

```
command_name argument_1_name(argument_type)
            argument_2_name(argument_type) ...
```

- To display the list of available commands, type `help` in the console window and press **Enter**.

11.4. Command Descriptions

Commands are organized into domains. If you add a command to *lgrConfig.xml*, you must specify the domain. The domains are:

- ♦ System
- ♦ Player
- ♦ Recorder
- ♦ Editor
- ♦ Simulation
- ♦ Invalid.

When used in the configuration file, simulation domain commands require a protocol string. The valid protocol strings are:

- ♦ None
- ♦ DIS
- ♦ HLA 13
- ♦ HLA 1516
- ♦ HLA 1516e.

11.4.1. Argument Types

Many commands have arguments. The data you enter for arguments must be of the correct type. For example, if an argument takes an int (integer), the value 5 is acceptable, but 5.0 is not. In the command descriptions, command argument types are shown in parentheses following each argument. [Table 11-1](#) describes the argument types used for commands.

Table 11-1: Argument types

Argument Type	Description
int	32 bit signed integer. For example: <code>SetNotifyLevel 2</code>
int64	64 bit signed integer. For example: <code>SetStepSize 1000000</code>
double	64 bit floating point number. For example: <code>SetTimeoutRate 5.0</code>
stringlist	A series of strings, separated by spaces. The individual strings cannot have spaces. The entire series is surrounded by double quotes. For example: <code>"string1 string2 string3"</code>
string	A string of alphanumeric characters.
time	Time values can be entered as a double (in seconds) or in Hour:Minute:Second:Fractions of a second format.
base64	Binary data encoded as a base 64 string.
bool	Boolean value. Can be true, false, 1 (true), or 0 (false).

11.4.2. System Domain Commands

Table 11-2 describes the commands in the System domain.

Table 11-2: System domain commands

Command	Description
DeleteFile <i>filename</i> (string)	Deletes the specified file. This command is considered dangerous and should only be used with caution.
Exit	Shuts down the Logger.
Help	Displays a list of available commands.
LoadPlugin <i>pluginName</i> (string)	Loads the specified plug-in.
SetEntityTracking	Allows entity tracking to be processed. Enabled by default. When disabled, entity tracking is not allowed to be processed and the entity filter is empty.
SetFastJumpsWithoutDiff	When jumping, checkpoints are used to establish the relative state of objects, but the packets between the checkpoint and jump time are not processed.
SetNotifyLevel <i>vrLevel</i> (int) <i>lgrLevel</i> (int)	Sets the notification level of VR-Link and the Logger.
SetPlaybackReadTimeLimit	When greater than 0.0, this command sets a limit on how long the Logger can spend reading packets from the Logger file during playback, after which the remaining packets up to the current time will be dropped.
SetTempDir <i>directory</i> (string)	Sets the directory in which the Logger stores temporary Logger files when recording and editing. By default the Logger uses the current user's <i>temp</i> directory (OS specific). If you specify a temporary directory, it is best to use a local directory for performance reasons. To reset the temp directory to the system directory use an empty string. The Logger deletes the temporary files when they are no longer needed.

11.4.3. Player Domain Commands

Table 11-3 describes the commands in the Player domain.

Table 11-3: Player domain commands (Sheet 1 of 2)

Command	Description
<code>ClosePlaybackFile</code>	Closes the current playback file.
<code>OpenFile filename(string) constructFilename (boolean)</code>	Opens the specified file for playback or constructs a file name using the the specified file name and remote control ID. If the filename has spaces, you must enclose it in quotes, which need to be protected with backslashes on the command line. For more information, please see “File Naming Conventions,” on page 13-8.
<code>Pause</code>	Pauses playback. This command has no effect if the Logger does not have a file open or is already paused.
<code>Play</code>	Starts playback. If the Logger is already playing this command has no effect.
<code>PlayPause</code>	Toggles between the paused and play state. If the Logger is stopped, it starts playback.
<code>Restart</code>	Starts playing from the beginning of the file.
<code>RevertFile</code>	Reverts a previously saved, but currently unsaved file to the previously saved state. All changes since the last save are lost. This command applies only to files that you open for playback or editing.
<code>SaveFile</code>	Saves a previously saved, but currently unsaved file to its current file name. The previous file is backed up to a file named <i>filename.bak.lgr</i> , for example: <i>test.lgr</i> would get saved to <i>test.bak.lgr</i> . The current file contains the latest changes.
<code>SaveFileAs filename(string)</code>	Saves the current file to a new file.
<code>SetBackupOnSave backup(bool)</code>	Enables or disables automatic backup when you save a previously saved file.
<code>SetCurrentTime time(time)</code>	Moves the timeline cursor to the specified time.
<code>SetLoopAtEnd value(bool)</code>	Enables or disables looping to start at the end of playback.
<code>SetPlaybackSpeed speed(double) skippingPlayback (bool)</code>	Sets the playback speed of the Logger. If <i>skippingPlayback</i> is True, skip from checkpoint to checkpoint without sending the interim state updates.

Table 11-3: Player domain commands (Sheet 2 of 2)

Command	Description
<code>SetPlayOnLoad value(bool)</code>	Enables starting playback immediately after opening a file.
<code>SetStartStopTimes startTime(time) stopTime(time)</code>	Sets the start and stop time of the Logger. The two times are bound to the first and last times in the tape. The stop time must be after the start time.
<code>StopPlayback</code>	Stops playback. This command has no effect if the Logger is not in the playback state.
<code>Unpause</code>	Unpauses the Logger. This command has no effect if the Logger is not paused.

11.4.4. Recorder Domain Commands

[Table 11-4](#) describes the commands in the Recorder domain.

Table 11-4: Recorder domain commands (Sheet 1 of 2)

Command	Description
<code>CancelRecording</code>	Cancels the current recording. The Logger must be currently recording. After the command is executed, the Logger is in an empty state, no file is loaded.
<code>CloseRecordFile</code>	Closes the file currently recording.
<code>NewFile filename(string) overwrite(bool)</code>	Opens a new file for recording.
<code>PauseRecording</code>	Pauses the recording process. Recording stops, but the data is not saved and the Logger remains in record mode. No data is recorded while the Logger is paused. You can resume recording to the same file.
<code>Record</code>	Starts recording to a new file.
<code>SetCheckpointParameters policy(int) size(int) count(int) time(time)</code>	Sets the checkpoint policy parameters on when checkpoints are generated. The Policy is one of the following: ♦ 0 — count, in number of packets ♦ 1 — size of packets, in bytes ♦ 2 — time, in seconds between checkpoints.  You must specify a value for each argument on the command line. The value does not need to be meaningful for the policies that are not being used.

Table 11-4: Recorder domain commands (Sheet 2 of 2)

Command	Description
<code>SetPauseOnRecordAs <i>record</i>(bool)</code>	Determines how the Logger behaves when given the Record As command. If true, the Logger pauses and does not begin recording until it is given the Record command. If false, the Logger begins recording as soon as a file name is specified.
<code>SetStopCondition <i>condition</i>(int) <i>close</i> (bool)</code>	Specifies the stop condition for recording as follows: ♦ 0 — no condition ♦ 1 — duration (enables <code>SetStopDuration</code>) ♦ 2 — filesize (enables <code>SetStopSize</code>). If the <i>close</i> parameter is True, the file is closed when recording is stopped.
<code>SetStopDuration <i>duration</i>(time)</code>	Stops recording at the specified time if <code>SetStopCondition</code> is set to 1.
<code>SetStopSize <i>filesize</i>(int64)</code>	Stops recording when the file reaches <i>filesize</i> size if <code>SetStopCondition</code> is set to 2.
<code>StopRecording <i>closeAndOpenForPlayback</i>(bool)</code>	Stops recording and ends. By default <i>closeAndOpenForPlayback</i> is true (1). If you set <i>closeAndOpenForPlayback</i> to false (0), then after you issue <code>StopRecording</code> , you are still in record mode. You would need to issue the <code>CloseRecordFile</code> command to close the file and exit record mode.

11.4.5. Simulation Domain Commands

Table 11-5 describes the commands in the Simulation domain. Examples of using the `SetConnectionParameters` command are in the protocol-specific configuration files. For a full definition of the arguments, please see Chapter 5, [Configuring the Logger](#).

Table 11-5: Simulation domain commands (Sheet 1 of 7)

Command	Description
Protocol-independent Commands	
Connect	Connects to the network simulation using the previously set parameters.
Disconnect	Disconnects from the network simulation.
SetRemoteControlId remoteControlId (integer)	Specifies the Logger's remote control ID. By default the value is assigned the Logger's site ID. This value is used if <code>SetRemoteControlSettings allowAllRemoteControl</code> is disabled.
SetRemoteControlSettings process(bool) record(bool) allowDelete(bool) allowAll(bool) sendResponses(bool) crippleGui(bool) connectSendInterval(double) heartbeatSendInterval(double)	Specifies how the Logger responds to Logger Control messages. ♦ <i>process</i> determines whether the Logger responds to Logger control messages. ♦ <i>record</i> determines whether or not Logger control messages are included in the file. ♦ <i>allowDelete</i> determines whether or not to respond to the delete file Logger control message. ♦ <i>allowAll</i> determines whether remote control PDUs are filtered based on the Logger's site-application-entityNum triplet, where the Logger's remote control ID is used for the entityNum. By default this parameter is enabled and all remote control PDUs are accepted. When disabled, the Logger accepts remote control PDUs when the receiver ID matches the Logger. The receiver ID can have wildcard values (0xFFFF or unsigned int = 65535) in any of the triplet values (for example, 1:1:65535 matches any Logger with site = 1 and application ID = 1). Remote control PDUs are filtered when they do not match. By default, the Logger's remote control ID equals its site ID. The remote control ID can be changed using the <code>SetRemoteControlId</code> command. ♦ <i>sendResponses</i> specifies whether the Logger sends the internal responses over the network for the remote control interface. The responses are generated by the Logger to provide feedback on the changes in state of the Logger based on the execution of commands (for example, fileOpened, playback paused, and so on).

Table 11-5: Simulation domain commands (Sheet 2 of 7)

Command	Description
	♦ <i>crippleGui</i> disables some features in the GUI and causes it to start up minimized. It forces the data view to not be visible at startup and it disables mouse clicks in the data view (that is, you cannot jump via mouse clicks). It also disables pop up error messages. ♦ <i>connectSendInterval</i> specifies how frequently, in seconds, to send connect messages. The connect message is used by a remote control interface to discover Loggers. Once the Logger is connected, the Logger will stop sending the connect message and start sending a heartbeat message. If the <i>connectSendInterval</i> is less equal 0, then the connect message will not be sent. ♦ <i>heartbeatSendInterval</i> specifies how frequently, in seconds, to send a heartbeat message after the Logger has connected. The heartbeat message is used by a remote control interface to determine if the Logger is no longer responsive (for example, crashed or broken network connection). If the <i>heartbeatSendInterval</i> is less than 0, then the heartbeat message is not sent.
<code>SetScaleAttributes <i>scale</i>(bool)</code>	Enables or disables attribute scaling due to changes in playback speed. When enabled, attributes (such as velocity and acceleration) are scaled to synchronize with playback speed changes. When disabled no scaling occurs.
<code>SetVRFFiltering <i>filter</i>(bool) <i>loId</i>(int) <i>hiId</i>(int)</code>	Configures VR-Forces message filtering. The filter argument enables or disables filtering of VR-Forces messages. The <i>loId</i> argument specifies the lowest message ID to filter out. The <i>hiId</i> argument specifies the highest message ID to filter out.
HLA-specific Commands	
<code>LoadSubscriptionListFile <i>filename</i>(string)</code>	Loads a subscription list file. The subscription list file is an XML file that you create using the Logger GUI. The Logger must be connected. The subscription settings are loaded from the file.
<code>SaveSubscriptionListFile <i>filename</i>(string)</code>	Saves the subscription settings to an XML file with the specified name. In the GUI, the extension <i>.subx</i> is added to filenames, but it is not added to the filename when saved from the command line.

Table 11-5: Simulation domain commands (Sheet 3 of 7)

Command	Description
<code>SetConnectionParameters</code>	<code>federateName(string) federateType(string) federationName(string) fedFile(string) fomModules(string) MIM(string) ignoreAdvisories(bool) fomVersion(double) fomRevision(int) fomMapper(string) fomMapperData(base64) useEmptyFomMapper(bool)</code>



The *federateType*, *fomModules*, and *MIM* parameters apply only to HLA Evolved. Do not include them for HLA 1.3 or 1516-2000.

Specifies HLA connection parameters:

- ♦ *federateName*. The name of the Logger federate.
- ♦ *federateType*. The type of the Logger federate. (HLA Evolved only)
- ♦ *federationName*. The name of the federation execution.
- ♦ *fedFile*. The name of the FED file (or FDD or XML file).
- ♦ *fomModules*. One or more FOM module files. (HLA Evolved only)
- ♦ *MIM*. A MIM file. (HLA Evolved only)
- ♦ *ignoreAdvisories*. Configures the Logger to ignore RTI advisory queues and always send data. Default: false.
- ♦ *fomVersion*. The RPR FOM version being used.
- ♦ *fomRevision*. The RPR FOM revision being used for version 2.0017. (1, 2, or 3)
- ♦ *fomMapper*. The name of the FOM Mapper being used, if any.
- ♦ *fomMapperData*. Initialization data for the FOM Mapper, if required.
- ♦ *useEmptyFomMapper*. The Logger will use an empty FOM Mapper, allowing it to record and playback using any FOM. The empty FOM Mapper does not support some playback features such as pause or scaling, which require the Logger to use dead reckoning to generate intermediate updates.

Table 11-5: Simulation domain commands (Sheet 4 of 7)

Command	Description
<code>SetDdmParameters</code>	<code>enableDdm(bool)</code> <code>playbackMode(string)</code> <code>subscribeMode(string)</code> <code>playbackDynamicAssociation(bool)</code> <code>playbackDynamicExtents(bool)</code> <code>allowPlaybackWithoutRegions(bool)</code> <code>recordConveyedRegions(bool)</code> <code>requestUpdates(bool)</code> <code>subscribeUnspecifiedAttributes(bool)</code> Specifies DDM parameters, as follows: ♦ <code>enableDdm</code>. Enable record and playback using DDM. Default: false. ♦ <code>playbackMode</code>. Supported modes are: – <code>PlaybackRegionsUseConveyedExtents</code>. During playback, regions are created and updated using recorded region data. – <code>PlaybackRegionsUseData</code>. Regions are created and updated by inspecting simulation data (. Requires plug-in extension. – <code>PlaybackRegionNotEnabled</code> (default). Updates and interactions are sent without regions. ♦ <code>subscribeMode</code>. Supported modes: – <code>SubscribeRegionsAreGlobal</code>. DDM subscriptions are made with global regions with extents of global min and max. – <code>SubscribeRegionsAreSpecified</code>. The regions are specified through subscription configuration files. (For details, please see the <code>LoadSubscriptionListFile</code> command.) The subscription region configuration can also be specified programmatically through the Logger plug-in API. – <code>SubscribeRegionNotEnabled</code> (default). Subscriptions are performed without regions. ♦ <code>playbackDynamicAssociation</code>. During playback, the association of regions with object instance attributes can be fixed or dynamic. If it is known that region associations with attribute instances never change, the overhead of checking for changes can be eliminated. This parameter should be disabled if possible. Default: false. ♦ <code>playbackDynamicExtents</code>. During playback, the region extents can be fixed or dynamic. The conventional notion of DDM regions regards them as dynamically changing extents to reflect the state of simulation data (for example, surrounding an entity's position). An alternative approach is for the regions to be fixed and the object attribute instances move from region to region. If it is known that region extents never change, the overhead of checking for changes can be eliminated. This parameter should be disabled if possible. Default: false. ♦ <code>allowPlaybackWithoutRegions</code>. During playback, it may happen that a region association cannot be established. In these cases, the Logger sends the data without region association. Disable this parameter if the federation agreement requires that all data must be transmitted with associated regions. Default: true.

Table 11-5: Simulation domain commands (Sheet 5 of 7)

Command	Description
	♦ <i>recordConveyedRegions</i>. Enable this parameter to record conveyed regions. Default: true. ♦ <i>requestUpdates</i>. Requesting updates can result in spikes in network traffic, especially since it is often performed by late joining federates when the exercise is already populated with many objects. In some DDM federations, object heartbeats are used instead of update requests. Disable this parameter to prevent the Logger from making requests for updates. ♦ <i>subscribeUnspecifiedAttributes</i>. The Logger may not be able to determine a region association to apply to an interaction class or object class attribute subscription. This situation may occur if the interaction or class does not specify a routing space or dimension in the FOM. It can also happen if the <i>subscribeMode</i> is configured as <i>SubscribeRegionsAreSpecified</i>, but some classes or attributes are left unspecified. By default, the Logger subscribes to these interaction classes and object class attributes without using a region association (in effect a global subscription). Disable this parameter if the Logger should only subscribe when a region association can be used. Default: true.
<code>SetDisableTimeManagementTest <i>disableTesting</i>(bool)</code>	Disables testing for time management if that test causes the Logger to behave badly. Ordinarily, when time management is enabled, when the Logger connects, it tries to become a time constrained federate to see if time management is enabled in the RTI. If it succeeds, the Logger sets time management settings as configured. In some cases, this test causes the Logger to hang. If this problem occurs, setting <i>disableTesting</i> to true turns off the test. However, time management is disabled.
<code>SetDrainInput(double <i>maxTime</i>, double <i>minTime</i>)</code>	Specifies the minimum and maximum amount of time, in seconds, to spend reading messages from the exercise connection (that is, the network) each tick.
<code>SetDynamicNamePrefix <i>prefix</i>(string)</code>	Specifies the prefix that the Logger attaches to the beginning of object names that begin with HLA_ when it is recording exercises.
<code>SetHeartbeatIntervals "<i>className</i>(string):<i>interval</i>(float):<i>inheritable</i>(bool) ..."</code>	Specifies a heartbeat interval for playback. You can specify multiple classes. For details, please see "Configuring HLA Heartbeating," on page 5-7.
<code>SetIgnoreAdvisories <i>mode</i>(bool)</code>	Specifies the Logger's response to HLA advisory messages. You must specify this command before the Connect command. Default: false.

Table 11-5: Simulation domain commands (Sheet 6 of 7)

Command	Description
<code>SetPassiveSubscription mode(bool)</code>	Configures passive subscription for HLA. You must specify this command before the Connect command. Default: false.
<code>SetSubscriptionLists objectClasses(stringList) interactionClasses(stringList)</code>	Sets the class subscription list. To subscribe to a class the class must be in the list. Each class must be specified using the full HLA class name (including parents). Some operating systems will not allow command line buffers that are long enough to allow use of this command on the command line. The Logger must be connected. The updated subscription list takes effect the next time you record.
<code>SetTimeManagement timeConstrainedRecording(bool) timeConstrainedPlayback(bool) timeRegulatingRecording(bool) timeRegulatingPlayback(bool) lookahead(double)</code>	Configures time management options. The <i>timeConstrainedPlayback</i> and <i>timeRegulatingRecording</i> arguments must be included on the command line, but are not supported and are ignored.
<code>SetTimeoutIntervals "className(string):interval(float):inheritable(bool) ..."</code>	Specifies an object timeout for recording. You can specify multiple classes. For details, please see “Configuring HLA Timeouts,” on page 5-7.
DIS-specific Commands	
<code>AddCustomPDU pduName(string) pduKind(string) entityIdOffset(string) entityTypeOffset(string)</code>	Specifies a PDU to add to the Filter by PDU Kind dialog box.
<code>SetCommonEntityTypes entityType1 ... entityTypen(string)</code>	Specifies a list of entity types to list in the Filter by Entity Type dialog box.

Table 11-5: Simulation domain commands (Sheet 7 of 7)

Command	Description
SetConnectionParameters <i>port</i> (int) <i>exerciseId</i> (int) <i>siteId</i> (int) <i>appNumber</i> (int) <i>useAsynchIO</i> (bool) <i>sendBufferSize</i> (int) <i>recvBufferSize</i> (int) <i>multicastTtl</i> (int) <i>destAddr</i> (string) <i>subnetMask</i> (string) <i>deviceAddr</i> (string) <i>mcastAddresses</i> (stringList)	Specifies parameters for the DIS exercise connection, as follows: ♦ <i>port</i>. The port to listen to. ♦ <i>exerciseId</i>. The Logger's exercise ID. ♦ <i>siteId</i>. The Logger's site ID. ♦ <i>appNumber</i>. The Logger's application number. ♦ <i>useAsynchIO</i>. Enable (1) or disable (0) asynchronous I/O. ♦ <i>sendBufferSize</i>. The size of the send buffer. ♦ <i>recvBufferSize</i>. The size of the receive buffer. ♦ <i>multicastTtl</i>. The multicast time-to-live, in seconds. ♦ <i>destAddr</i>. The destination address to send to. ♦ <i>subnetMask</i>. The subnet mask used for determining the destination when <i>destAddr</i> is not unspecified. ♦ <i>deviceAddr</i>. The network device to use. ♦ <i>mcastAddresses</i>. A list of multicast addresses.
SetDrainInput(double <i>maxTime</i> , int <i>numPdusToRead</i>)	Specifies the maximum amount of time, in seconds, to spend reading messages from the exercise connection (that is, the network) and the number of PDUs to read between each check of the maximum time.
SetHeartbeatRate <i>rate</i> (double)	Specifies the DIS heartbeat rate.
SetTimeoutRate <i>rate</i> (double)	Specifies the DIS timeout rate.

11.4.6. Editor Domain Commands

Table 11-6 describes the commands in the Editor domain.



Editing operations can be performed only on files that have been opened for playback. After you perform an edit, the edited file is unsaved. You must save it to preserve the change.

Table 11-6: Editor domain commands (Sheet 1 of 2)

Command	Description
<code>AppendFileToEnd filename(string)</code>	Appends the file <i>filename</i> to the end of the open file.
<code>ClearSection start(time) end(time)</code>	Removes all packets between the <i>start</i> and <i>end</i> times in a file.
<code>ConvertFile</code>	Converts the current file to the latest file format. This command is not valid for files that are already in the latest file format.
<code>CutSection start(time) end(time)</code>	Removes all packets between the <i>start</i> and <i>end</i> times in a file. All packets after the <i>end</i> time are offset so that the first packet after the <i>end</i> time is now at the <i>start</i> time.
<code>ExportFile exportFormat(string) destination(string) checkpoint(bool) openAfterExport(bool)</code>	Exports a file using a given <i>exportFormat</i> to a <i>destination</i>. The last two parameters are only used when exporting a Logger file. The Logger tries to export using the <i>exportFormat</i> and the <i>destination</i>. If <i>checkpoint</i> is true and the file is a Logger file, new checkpoints are generated for the output. If <i>openAfterExport</i> is true and the file is a Logger file, the Logger closes the current file and tries to open the new file. The valid <i>exportFormat</i> strings are based upon the registered export objects. The valid strings are identical to the Export and Export Selection menu commands in the GUI. The formats supplied with the Logger are Plain Text (if the lgrDump plug-in is loaded) and Logger File (export to a Logger file). The <i>destination</i> typically is a filename (but does not need to be) and is specific to the <i>exportFormat</i>. The Plain Text exporter exports to any filename, but is typically exported to a <i>.txt</i> file. The Logger File exporter exports to a <i>.lgr</i> file and adds that extension if it is not in the <i>destination</i> string. Other exporters may encode the <i>destination</i> differently.

Table 11-6: Editor domain commands (Sheet 2 of 2)

Command	Description
<code>ExportSection</code> <i>exportFormat</i> (string) <i>destination</i> (string) <i>start</i> (time) <i>end</i> (time) <i>offset</i> (double) <i>checkpoint</i> (bool) <i>openAfterExport</i> (bool)	Exports a section of a file using a given <i>exportFormat</i> to a <i>destination</i> . This command works just like the <code>ExportFile</code> command except that it exports a segment of a file rather than the entire file.
<code>MergeInFile</code> <i>filename</i> (string) <i>offset</i> (double)	Merges the Logger file <i>filename</i> into the open file. An optional offset is applied to all packets from the file before being combined with the current data.
<code>OffsetTime</code> <i>offset</i> (double)	Offsets the time of the packets in the open file by the <i>offset</i> value. The <i>offset</i> value can be positive or negative. However, if you specify a negative offset, any data that results with a time less than zero is lost.
<code>PrefixFileToStart</code> <i>filename</i> (string)	Inserts the file named <i>filename</i> at the start of the open file.
<code>TrimStart</code>	Removes any dead time from the start of the file and offsets all packets so that the first packet time stamp is 0.0.

11.5. Command List

Table 11-7 section lists all Logger commands and their domain:

Table 11-7: Logger commands

Command	Domain
<code>AddCustomPDU</code>	Simulation
<code>AppendFileToEnd</code>	Editor
<code>CancelRecording</code>	Recorder
<code>ClearSection</code>	Editor
<code>ClosePlaybackFile</code>	Player
<code>CloseRecordFile</code>	Recorder
<code>Connect</code>	Simulation
<code>ConvertFile</code>	Editor
<code>DeleteFile</code>	System
<code>Disconnect</code>	Simulation
<code>ExportFile</code>	Editor

Table 11-7: Logger commands

Command	Domain
ExportSection	Editor
Exit	System
Help	System
LoadPlugin	System
LoadSubscriptionListFile	Simulation
MergeInFile	Editor
NewFile	Recorder
OffsetTime	Editor
OpenFile	Player
Pause	Player
PauseRecording	Recorder
Play	Player
PlayPause	Player
PrefixFileToStart	Editor
Record	Recorder
Restart	Player
RevertFile	Player
SaveFile	Player
SaveFileAs	Player
SaveSubscriptionListFile	Simulation
SetCheckpointParameters	Recorder
SetBackupOnSave	Player
SetCommonEntityTypes	Simulation
SetConnectionParameters	Simulation
SetCurrentTime	Player
SetDdmParameters	Simulation
SetDisableTimeManagementTest	Simulation
SetDrainInput	Simulation
SetDynamicNamePrefix	Simulation
SetEntityTracking	System
SetFastJumpsWithoutDiff	System
SetHeartbeatRate	Simulation
SetIgnoreAdvisories	Simulation

Table 11-7: Logger commands

Command	Domain
SetLoopAtEnd	Player
SetNotifyLevel	System
SetPassiveSubscription	Simulation
SetPauseOnRecordAs	Recorder
SetPlaybackReadTimeLimit	System
SetPlaybackSpeed	Player
SetPlayOnLoad	Player
SetRemoteControlId	Simulation
SetRemoteControlSettings	Simulation
SetScaleAttributes	Simulation
SetStartStopTimes	Player
SetStopCondition	Recorder
SetStopDuration	Recorder
SetStopSize	Recorder
SetSubscriptionLists	Simulation
SetTempDir	System
SetTimeoutRate	Simulation
SetVRFFiltering	Simulation
StopPlayback	Player
StopRecording	Recorder
TrimStart	Editor
Unpause	Player



Using The Logger API

The Logger API allows you to add features to the Logger, modify its behavior, and embed Logger functionality into your own code.

Introduction	12-3
The Logger Architecture	12-4
Drivers	12-4
Factories	12-5
Interfaces	12-5
The Logger Message Pipeline.....	12-6
The Simulation Manager.....	12-6
Embedding the Logger in an Application	12-7
Embedding Plug-ins in an Application	12-7
The playFile Example	12-8
Creating a Stand-Alone Logger Application	12-10
The Logger Plug-in API	12-10
Writing a Plug-In	12-11
Loading Plug-Ins	12-12
Logger API Coding Conventions	12-13
Namespaces	12-13
Templates	12-13
Shared Pointers.....	12-14
Virtual Constructors.....	12-14
Runtime Values	12-15
Thread Safety	12-16
Interface Signal Slots	12-16
Observer Pattern.....	12-17
Logger Libraries	12-18

Other Libraries and Plug-ins	12-18
Compiling and Linking with the Logger API	12-19
Compiling and Linking for Linux	12-19
Compiling and Linking for Microsoft Windows	12-21
Building Examples	12-21

12.1. Introduction

The Logger application programming interface (API), allows you to extend and customize the functionality available in the Logger. For example, you can embed a Logger in your own executable to play and record Logger files, or you can build your own version of a Logger executable with your own customizations. You can also use the Logger API to build dynamically linked modules, or plug-ins, that the standard Logger executable can load at runtime.



You must have a VR-Link developer's license to build applications or plug-ins that use the Logger API.

The Logger API is logically separated into logging functionality and application functionality. Logging functionality involves the low level classes responsible for recording simulation data and playing it back. The application functionality provides a GUI, configuration file loader, and other high-level end-user functionality. The API is designed to be extensible, to make it easier to replace or extend both low-level and high-level functionality.

If your goal in using the Logger API is to add standard Logger features to your own application, the *DtLogger* and *DtLgrInterface* classes may be the only portion of the API that you need to understand. (It is possible to use the *DtLogger* class alone, without any corresponding Logger interface components. However, you would need to implement a way to control the logging process.)

You can use the Logger API as a starting point for writing your own data logging application that uses whichever components of the Logger application that you want. However, you will be limited by project dependencies. For example, if you do not want to use the Qt Toolkit for your GUI library, you cannot use the Logger GUI libraries.

You can extend the Logger application's functionality. Possible extensions could include new file formats, data inputs and outputs, or additional processing of data within the Logger. You can extend the application by completely rebuilding it, or by writing plug-ins.

12.2. The Logger Architecture

The Logger architecture is highly modular and designed to be independent of the simulation protocol, user interface, and execution type (it supports both iterative or multi-threaded).

Every functional module in the Logger has a driver, a factory and an interface. The factory defines the commands that can be sent to the driver and the responses the driver sends back to the interface. The interface provides a way to send the commands and receive the responses. The driver receives the commands, acts upon them, and sends responses if necessary. The stateObserver plug-in example shows how to implement a driver, factory, and interface.

The Logger also has a message pipeline that is controlled by various drivers (only one at a time). The pipeline takes messages from an input stage, moves them through the state stage, and sends them out of the output stage. For example, the playback driver reads a file during the file input stage and sends the data to a simulation over a network during the simulation output stage.

12.2.1. Drivers

The base class of drivers that act on commands is *DtLgrCommandHandler* (*lgrDriver.h*). This class provides the ability to receive and respond to commands from the *DtLgrSystemDriver* (*lgrSystemDriver.h*). To connect a command to a function in the driver use the `addCommandHandler()` function. Member functions are usually connected using a functor object in the driver's constructor. Depending on the operation, a driver may request access to the pipeline (if another driver currently owns the pipeline it is notified and can perform any necessary operations to release the pipeline or reject the request). The driver's `tick()` member function is called regularly by *DtLogger*, which executes any submitted commands. An example of a driver class is *DtLgrPlaybackDriver* (*lgrPlaybackDriver.h*). For an example of how to write a driver see *stateObModel.h* and *stateObModel.cxx* in the stateObserver example.

12.2.2. Factories

The factory class provides static functions for defining, creating, and decoding commands. Each command has three functions – a define function, a create function, and a decode function. The command definition function returns a *DtCommandDefinition* (*lgrRuntimeCommandFactory.h*) object, which defines the name of the command, its type, and its parameters. This definition is used by the drivers to register command handlers and by the runtime command factory, which parses commands from strings (typically originating from an input console). The `create()` function takes the necessary parameters and returns a *DtCommand* (*lgrCommand.h*) object. The `decode()` function takes a *DtCommand* object and decodes its parameters. If the parameters or type of the *DtCommand* object are not correct for the specific command, an exception is thrown. This indicates an error in the create function or in the command handlers. Similarly, there are create functions and decode functions for responses (there is no definition function). An example of a factory class is *DtLgrPlaybackCommand Factory*, in *lgrPlaybackCommandFactory.h*.

A factory class is not strictly necessary. However, it provides a clear distinction between the definition of a command and the implementation of the command handler. Any class, including the driver or the interface, can contain the functions for defining, creating, and decoding a command object. Utility C++ macros for declaring the necessary static command functions are provided in *lgrCommand.h*.

12.2.3. Interfaces

The interface classes provide a simplified method for sending commands and receiving responses. Each interface maintains a local state, which is updated via the responses. Interfaces are designed to be thread-safe and are used to communicate across thread boundaries. They are also safe to construct in separate threads, because the Logger is locked when it creates interfaces. Interfaces must be updated regularly to process queued responses. This is very important, because only one *DtResponse* is created and shared across all interfaces. If an interface does not call `update()` the response will remain in its queue and never be deallocated. Please see *playTape.cxx* and *writeTape.cxx* for examples of how to use interface classes. If you need to use several interfaces of different types, use an instance of *DtLgrFullInterface*, which contains several interfaces.

An example of an interface class is *DtLgrPlaybackInterface*, in *lgrPlaybackInterface.h*.

The *DtStateObserverController* in *stateObController.h* and *stateObController.cxx* uses a base *DtCommandInterface* object, rather than subclassing it specifically for the state observer commands. Commands are submitted manually and callbacks are added for specific responses. This is easier for very small interfaces. However, once an application has several commands and responses, it is preferable to organize it as a full sub-class, which hides these details when using the interface.

12.2.4. The Logger Message Pipeline

The Logger pipeline has three major stages – the input stage, the state stage, and the output stage. Each stage is highly configurable and largely independent of each other. Depending on the mode of operation, different objects are added to and removed from the pipeline. For example during recording a *DtSimInput* acquired by the simulation manager does the following:

1. Reads data from the network.
2. Sends the data to the state stage, which maintains the state of the simulation internal to the Logger.
3. Sends the data to the output stage, which in this case is a file write object, which writes the data to a file.

The Input Stage

The input interface is defined by *DtMessageInput* (*messageInput.h*). For simulation input, the *DtLgrSimInput* class (*lgrSimInput.h*) provides a common interface for input from a live simulation. The primary purpose of the input interface is to retrieve messages over time.

The State Stage

The state interface is defined by *DtMessageState* (*messageState.h*). It has derived classes specific to each supported protocol. The state stage stores the state of persistent objects in a network-independent format. It can generate the messages needed to store the entire state of the simulation as a checkpoint.

The Output Stage

The output interface is defined by *DtMessageOutput* (*messageOutput.h*). The derived class *DtLgrSimOutput* (*lgrSimOutput.h*) provides a common interface for output to a live simulation.

12.2.5. The Simulation Manager

The simulation manager (*DtLgrBaseSimManager*) (*lgrBaseSimManager.h*) and its sub classes (*DtLgrDisSimulationManager* (*lgrDisSimManager.h*), *DtLgrHlaSimulationManager* (*lgrHlaSimManager.h*), and so on) provide simulation management when connected to a distributed simulation. The simulation manager constructor must be defined by the *DtLgrProtocolRegistration* (*lgrProtocolRegistration.h*) implementation, which is required for the *DtLogger* to function. The simulation manager is responsible for maintaining all simulation specific objects such as publishers and the necessary callbacks. It also provides access to creating simulation-specific objects, such as the simulation input and output stages, and opens files for a specific format.

12.3. Embedding the Logger in an Application

You can use the Logger API to embed logging capability into your own applications.

The core functionality of the Logger is provided in a class called *DtLogger*. A *DtLogger* object has a system driver and a protocol registration object. The system driver provides all low level functionality, such as command passing, driver registration, managing of the pipeline, and the base simulation manager. A protocol registration object virtual constructor must be set before creating a *DtLogger* object. Utility classes *DtDisLogger* and *DtHlaLogger* provide automatic registration of their respective protocols and are designed for use when embedding the Logger in other applications.

If you are using Logger functionality in your own application, you must instantiate a *DtLogger* class. If you are using it in conjunction with one (or more) Logger interface components, then you must also instantiate these components. If you are creating a Logger plug-in, then the standard Logger application creates the instance of the *DtLogger* class for you, and allows you to access it from your plug-in code.

The Logger includes the following examples of using *DtLogger* in a custom application: *playFile.cxx*, *readFile.cxx*, *recordFile.cxx*, and *writeFile.cxx*. They are in *./examples/simpleFileActions*.

12.3.1. Embedding Plug-ins in an Application

Important parts of Logger functionality, such as annotations, are implemented as plug-ins, rather than as core libraries. If you are embedding the Logger in an application, you can also embed plug-ins. To embed plug-in functionality, link to the plug-in libraries and use the plug-in's classes in your application as if they were part of the Logger API.

12.3.2. The playFile Example

This section describes the playFile example.

First construct a Logger object. We use the helper classes to facilitate protocol registration.

```
#if DtDIS
    DtDisLogger* aLogger = DtDisLogger::create(true);
#endif
#if DtHLA
    DtHlaLogger* aLogger = DtHlaLogger::create(true);
#endif
```

Next, connect the Logger object to an exercise. (This does not necessarily need to come first; however, it makes for clearer logic.)



The Logger is not actually connected after the connect call since only a connect command is submitted. Call tick() to execute the command.

```
DtExerciseConnInitializer exConnInit;
aLogger->connect(exConnInit);
```

While the Logger object provides some basic functions, to control and receive responses from the Logger, you must construct an interface to the Logger. This portion of the example constructs a simulation interface and attaches it to the Logger object.

```
DtLgrSimInterface* simInterface = DtLgrSimInterface::create(*aLogger);
```

The Logger is ticked to process the submitted commands, which up to this point will be setConnectionParameters and connect. The results of these commands will be sent to the interface. The interface, like the Logger, must be updated to process the responses, which is done via the update() member function.

```
aLogger->tick();
simInterface->update();
```

With the interface up-to-date, you can check to see if the Logger was able to connect.

```
if(!simInterface->isConnected())
{
    return 1;
}
```

When an interface is no longer needed, delete it:

```
delete simInterface;
```

It is better to maintain a single interface rather than repeatedly creating and destroying them. A single interface can be shared across all objects in the same execution thread. When you write multi-threaded code, you must have one interface per thread. Interfaces are designed with thread safety in mind, which makes them an optimal solution for multi-threaded programs.

To control Logger playback, you need a playback interface. Any number of interfaces may be constructed and attached to the Logger. The performance overhead for each interface is not large; however, it is often best to minimize the number of interfaces used.

```
DtLgrPlaybackInterface* pbInterface =  
    DtLgrPlaybackInterface::create(*aLogger);
```

Each interface has two types of functions – those that control the Logger by sending commands, and those that read the state of the interface. For example, to open a file and determine if the operation was successful requires the command to be submitted, the Logger to be ticked, the interface to be updated, and the state to be read back.

```
pbInterface->openFile(fileName);  
aLogger->tick();  
pbInterface->update();  
if(!pbInterface->isFileLoaded())  
{  
    return 1;  
}
```

This pattern is repeated by all operations, for example:

```
pbInterface->play();  
aLogger->tick();  
pbInterface->update();  
if(!pbInterface->isPlaying())  
{  
    return 1;  
}
```

When performing a continuous action, for example playing, you must construct an execution loop. (This is not necessary if you use the *DtThreadedLogger*, which runs in a separate thread.)

```
double loopDelay = .1;  
while(pbInterface->isPlaying())  
{  
    aLogger->tick();  
    pbInterface->update();  
    DtSleep(loopDelay);  
    if(keyCheck("q\r")) break;  
}
```

It is not strictly necessary to stop the Logger before deleting it, however it is included here for completeness.

```
if(pbInterface->isPlaying())  
{  
    pbInterface->stop();  
    aLogger->tick();  
}
```

When you are finished with the Logger, delete all interfaces, then delete the Logger object.

```
delete pbInterface;  
delete aLogger;  
return 0;
```

12.4. Creating a Stand-Alone Logger Application

Creating a stand-alone Logger application can be as simple as the embedded Logger examples, or can include more advanced front-end capabilities. The MÄK Data Logger application is an instance of a complex Logger application.

You can embed the entire Logger application using a *DtLoggerApplication* object, as follows:

```
DtLoggerApplication app(argc,argv);
...
app.run();
```

Typically, you would put additional configuration commands after constructing the application, but before running it. For example, the following simple program opens a specific file and plays it:

```
DtLoggerApplication app(argc,argv);
DtCommand aCommand =
    DtLgrPlaybackCommandFactory::createOpenFileCommand("test.lgr");
DtCommand bCommand = DtLgrPlaybackCommandFactory::createPlayCommand();
app.logger().submitCommand(aCommand);
app.logger().submitCommand(bCommand);
app.run();
```

While creating a new application based on *DtLoggerApplication* gives you a large amount of control over the low level implementation of the Logger, in many cases this level of control is not needed or exceeds what is necessary for the project. It is usually preferable to use the plug-in capability to extend the Logger application.

12.5. The Logger Plug-in API

A plug-in is a dynamically linked module that is loaded by the Logger at runtime. You can use plug-ins to modify the Logger executable. For example, you can change how the Logger processes packets, or you can override the Logger's *DtLogger* object with your own. While *DtLogger* is able to use plug-ins to extend functionality, the value of plug-ins is greater when using the existing Logger application, which has a plug-in interface for extending the application.

When you use a plug-in, all the basic functionality of the Logger application remains intact – command-line arguments are read and interpreted, user interfaces are created, and Logger control messages are intercepted. After all the normal initialization work, the application loads your plug-in and your changes take effect.

The Logger State Observer and lgrDump are examples of how to write a plug-in for the Data Logger application. They show the use of many techniques employed by the Logger API. The source code for the examples is in *./examples/stateObserver* and *./examples/lgrDump*.

12.5.1. Writing a Plug-In

All plug-in libraries must export a “C” style function, which will be recognized by the Logger executable when it loads the plug-in. A plug-in uses the following functions:

- ♦ void LoadPlugin()
- ♦ void UnloadPlugin()
- ♦ void EnableLoggerPlugin(DtLogger& logger)
- ♦ void EnableLoggerAppPlugin(DtLoggerApplication& app).

LoadPlugin() and UnloadPlugin() are required. They are called when a plug-in is loaded and unloaded. They can be used for allocation and deallocation of resources. Either EnableLoggerPlugin() or EnableLoggerAppPlugin() or both are required and are called after the *DtLogger* or *DtLoggerApplication* object is instantiated. Typically *DtLogger* plug-ins override existing classes and provide core functionality. *DtLoggerApplication* plug-ins extend the Logger application and can add additional functionality to the user-interface and extend the functionality of the application.

The following code example is from the State Observer plug-in (*stateObserver.cxx*):

```
extern "C" {
void LoadPlugin()
{
}
void EnableLoggerPlugin(MAKLogger::DtLogger& logger)
{
    // Create controller
    controller = new DtStateObserverController();
    // Create model
    model = new DtStateObserverModel(controller, logger.driver());
    // Add drive to Logger to handle commands.
    logger.driver().addDriver(DtStateObserverController::theCommandType,
DtLgrCommandHandlerSP(model));
}
void EnableLoggerAppPlugin(MAKLogger::DtLoggerApplication& application)
{
    DtLgrAppGui* gui = DtLgrAppGui::getQtGui(application);
    if(!gui)
    {
        DtLgrPrinter::errorMessage("Unable to get GUI.");
        return;
    }
    // Create view, must be done here after the gui is created.
    view = new DtStateObserverView(controller);
    controller->init(model,view);
    // Connect model to the sim state.
    model->connect(application.logger().simManager()->state());
    // Add component to GUI.
    gui->addGuiComponent(view);
}
void UnloadPlugin()
{
    // No need to delete view, that is handled by the GUI
    // No need to delete model, that is handled by the system driver.
    delete controller;
} }
```

12.5.2. Loading Plug-Ins

The Logger looks for plug-ins in the location specified in *lgrConfig.xml*. A file is recognized as a plug-in library if it has the appropriate extension (*.dll* or *.so*) and can be loaded as a DLL or shared library.

The following is an example of configuring plug-ins in *lgrConfig.xml*:

```
<defaults>
...
<!-- -->
<!-- Library Modules -->
<!-- Do not add extension or 'd' if it is debug, these will be added -->
<!-- automatically. -->
<...
<var name="stateObserverPlugin" type="string"
    value="./stateObserver"></var>
<var name="annotationPlugin" type="string"
    value="./lgrAnnotations"></var>
<!-- -->
...
</defaults>
<commands name="init">
    <!-- Initialize commands executed after the command line is parsed -->
    ...
    <command domain="System" protocol="None" command="LoadPlugin">
        <param name="pluginName" var="annotationPlugin"></param>
    </command>
    <command domain="System" protocol="None" command="LoadPlugin">
        <param name="pluginName" var="stateObserverPlugin"></param>
    </command>
</commands>
```

You can also load plug-ins programmatically by sending the `loadplugin` command to the *DtLogger*.

The *DtLogger* can load a plug-in when it is sent the correct system command.

```
//Assume DtLogger* aLogger
DtCommand cmd =
    DtLgrSystemCommandFactory::createLoadPluginCommand("stateObserver");
aLogger->submitCommand(cmd);
aLogger->tick();
```

12.6. Logger API Coding Conventions

This section describes some of the coding mechanisms used in the Logger API.

12.6.1. Namespaces

The Logger uses API namespaces where possible to limit symbol conflict with other APIs. The Logger API uses the MAKLogger namespace. When you use Logger API classes, you have the following options for specifying the required namespace:

- ♦ In a header file, either explicitly specify the namespace scope or surround declarations with a namespace block
- ♦ In the source file use the `using` statement with the class name or use the `using namespace` statement.

Examples

```
// foo.h
namespace MAKLogger
{
    class DtLogger;
}

class Foo
{
    ...
    MAKLogger::DtLogger* myLogger;
};

//foo.cxx
#include "foo.h"

// Import a single class
using MAKLogger::DtLogger;
//or import them all
using namespace MAKLogger;

Foo() : myLogger( DtLogger::create( ) )
{
}
```

12.6.2. Templates

The Logger API uses C++ templates and template specializations. Therefore, to use the API, you should be familiar with their use.

12.6.3. Shared Pointers

The Logger uses shared pointers (also known as smart pointers). These are template classes that act like normal pointers, but are reference counted and only deallocated when the last reference is lost. All shared pointer specializations are appended with the characters SP. The exception to this rule is *DtResponse* which is an alias to *DtCommandSP*. Shared pointers are not explicitly deleted and should always be passed by value (not by reference). They are safe to pass across thread boundaries; however the objects themselves are not.

```
// test.cxx
#include <vLSharedPtr.h>

typedef DtBoost::shared_ptr<Foo> FooSP;

void test()
{
    FooSP aFoo = FooSP(new Foo( ));
    aFoo->doSomething();
    Foo* anUnsafeFoo = aFoo.get();
    // No need to delete since there is only one reference to aFoo.
    // aFoo will be deallocated on leaving the function
}
```

12.6.4. Virtual Constructors

The Logger implements a pattern for creating classes with virtual constructors. Virtual constructors allow objects to be replaced at runtime by enforcing a creating pattern via a static function pointer, which can be reassigned. Several macros are provided which make creating the necessary functions and members easier and smaller. Each type of virtual constructor has two macros, one for the declaration in the header, and one for the definition in the source.



Argument lists must be surrounded by parentheses, including when there are no arguments.

The general format for a header file is:

```
DT_VIRTUAL_CTR( CLASS_NAME , ( TYPE_LIST ) )
```

The corresponding format for a source file is:

```
DT_VIRTUAL_CTR_DEFINE( CLASS_NAME , ( TYPE_NAME_LIST ) , (NAME_LIST) )
```

An example of a virtual constructor in a header file with no arguments (from *logger.h*):

```
DT_VIRTUAL_CTR(DtLogger,())
```

The corresponding virtual constructor in the source file:

```
DT_VIRTUAL_CTR_DEFINE(DtLogger,(),())
```

An example of a virtual constructor in a header file, with arguments:

```
DT_VIRTUAL_CTR(DtLoggerFileWriter, (DtMessageIOInfo*,
DtLoggerRecordFile*))
```

The corresponding virtual constructor in the source file:

```
DT_VIRTUAL_CTR_DEFINE(DtLoggerFileWriter, ( DtMessageIOInfo* info,
DtLoggerRecordFile* file ), ( info , file ) )
```

Classes that have a virtual constructor can only be created using a static create() function. This function, by default, calls a constructor with the supplied parameters. It can be overwritten via the static setCreator(functionPtr) function.

```
DtLogger* logger = DtLogger::create();
...
DtLogger* overrideConstructor() { return new DtOverrideLogger();}
DtLogger::setCreator(&overrideConstructor);
DtLogger* newLogger = DtLogger::create();
```

In addition to virtual constructors the API has pure virtual constructors, which must be defined before they can be called, DT_PURE_VIRTUAL_CTR. It also has virtual copy constructors and assignment operators using DT_VIRTUAL_COPY_ASSIGN. For documentation on all the macros and their implementations, please see *virtualConstructor.h*.

12.6.5. Runtime Values

The Logger uses a class of objects which wrap multiple C++ types in the same object hierarchy. This allows the Logger to provide a generic interface for commands and to convert strings to native types and vice-versa. The abstract base class is *DtAnyValue*, defined in *runtimeValue.h*. It has a templated sub-class, *DtValue*, which is templated upon a single value semantics type (for example, int, float *DtString* and so on). To construct a *DtAnyValue* for a specific type use the utility function `DtAnyValue* DtWrapValue(const T& input)`, which constructs a new `DtValue<T>` object.

12.6.6. Thread Safety

You can ensure thread safety by using mutexes. VR-Link provides a mutex implementation via the *DtMutex* class. The Logger also provides a utility class call *DtThreadSafe*, which wraps a class's interface with the required Mutex calls to ensure the object is always thread safe.

```
DtThreadSafe<std::vector<int> > safe_vector;
safe_vector->push_back(4);
safe_vector->push_back(2);

//Create a local scope to lock the vector only during iteration.
{
    DtScopedSequenceLock lock = safe_vector.sequenceLock();
    std::vector<int>::iterator iter = safe_vector->begin();
    std::vector<int>::iterator end = safe_vector->end();
    for(;iter != end;++iter)
    {
        std::cout << *iter << _g,_h;
    }
}
```

12.6.7. Interface Signal Slots

It is desirable to allow interfaces to call a function when a value changes. This is implemented by the Logger API as *DtValueSignals*. A *DtValueSignal* is templated on the type of value it signals. The signals are public objects that can be connected to functions and member functions. When a signal is emitted, all connected functions are called. The call signature of the function or member function must only take a constant reference to the type of the value signal. The signal slot implementation provides a capability similar to that of traditional callbacks.

```
class FooSig
{
    int i;
    DtValueSignal<int> i_signal;
};

class FooListen
{
    void print(const int& i) { std::cout << i; }
};

int sum(const int& i)
{
    static int sum
}

FooSig s;
FooListen l;
Connect(s,&l,FooListen::print);
```

12.6.8. Observer Pattern

The observer pattern is similar to signals and slots, but works on the basis of events and an observable object. The observer pattern allows events to trigger callbacks and pass a single object to be observed temporarily. The observer pattern implementation is primarily used through three templated classes, *DtObserver* (*observe.h*), *DtObserveEvent* (*observeEvent.h*), and *DtObserveConnection* (*observeConnection.h*). The *DtObserver* is an object that can be constructed using a function or a member function and acts as a functor when an event is fired. *DtObserveEvent* is an object that can be fired at any time by a host object. *DtObserveConnection* constructs the connection between the observer and an event of the same observed object type. The connection can track the lifetime of both the event and the observer and automatically breaks the connection if one object is deallocated. You can add multiple observers to the same event. For an example of how to use the observer pattern classes, please see the `stateObserver` example.

12.7. Logger Libraries

The minimum libraries necessary to build a functional Logger are the *lgrUtil*, *lgrCore*, and *lgrSim* libraries.

Table 12-1: Logger API libraries

Library	Description
<i>lgrUtil</i>	The Logger utility library contains classes that help implementation and integration of all the Logger libraries. The utility library depends on VR-Link's utility library.
<i>lgrCore</i>	The Logger core library contains the basic required classes for the Logger, except for recording and playback, which are in <i>lgrSim</i> .
<i>lgrSim</i>	The <i>lgrSim</i> library provides the classes for recording a specific simulation protocol. <i>lgrSim</i> has three versions, one for each supported protocol. Each variation is linked against the corresponding VR-Link library. The <i>lgrSim</i> libraries also function as a <i>lgrApp</i> plugin.
<i>lgrApp</i>	The Logger application library builds upon the <i>lgrCore</i> library to create a complete application. It can load plug-ins, which provide additional functionality. It must have a <i>DtProtocolRegistration</i> registered that implements the abstract protocol-independent classes. <i>DtLgrAppGuiInterface</i> must be defined (typically by a plug-in loaded by the configuration file) to display a GUI. <i>lgrApp</i> runs multiple threads.
<i>lgrGui</i>	The Logger application GUI provides the base implementation of the GUI for the Logger application. This library is not loaded by the Logger directly, but rather is used by the <i>lgrSimGui</i> libraries.
<i>lgrSimGui</i>	The Logger simulation protocol specific GUI builds on the <i>lgrGui</i> to add protocol specific interfaces. This library is loaded by the <i>lgrApp</i> as a plug-in and must define the <i>DtLgrAppGuiInterface</i> creator function.

12.7.1. Other Libraries and Plug-ins

appEnv – The application environment library provides a generic environment for loading complex values and maintaining a hierarchical relationship of the data. The data is independent of any specific file format and the XML and MTL file formats are supported.

lgrAnnotations – The Logger annotations library provides the ability to annotate a Logger file. This library functions as a plug-in into the *lgrGui*.

12.8. Compiling and Linking with the Logger API

This section lists the settings you need to use to compile and link with the Logger API under Linux and Windows. The Logger includes dynamic libraries. On Windows, it includes release and debug (d) libraries. The set of libraries that you link against depends on how you plan to use the Logger. In all cases, link against *lgrUtil* and *lgrCore*. If you are rebuilding the Logger application or embedding it in another application, link against *lgrApp*, *lgrSim*, and *appEnv*. If you are using the Qt GUI for the Logger, link against *lgrGui* and *lgrSimGui*. You can link to the *lgrAnnotations* and *stateObserver* plug-ins or load them through the plug-in interface.



The example link lines in this section assume you are using the core libraries and application libraries.

12.8.1. Compiling and Linking for Linux

To compile against the Logger API in Linux, you must add the following include directive to your compile line:

```
-I loggerDir/include -I $(MAK_VRLDIR)/include
-I $(MAK_VRLDIR)/include/compatibility
```

where *loggerDir* is the path to the root of your Logger installation and the environment variable MAK_VRLDIR is set to the path to the root of your VR-Link installation. When building for HLA 1.3, HLA 1516, or HLA Evolved, also use:

```
-I $(MAK_RTIDIR)/include
```

where the environment variable MAK_RTIDIR points to the root of your RTI installation.

Libraries that are not prefaced with “lib” (*lgrSim*, *lgrSimGui*, *lgrAnnotations*, and *stateObserver*) must use their absolute pathname in the link line.

Linking for HLA Exercises

For HLA 1.3, link against

```
-LloggerDir/lib -lliblgrUtil -lliblgrCore -lliblgrApp
-llibappEnv -lliblgrGui loggerDir/lib/lgrSimGuiHLA13.so
-L$(MAK_VRLDIR)/lib -lvlHLA13 -lvl -lmatrix -lmtl
-lvlutil -L$(MAK_RTIDIR)/lib -lRTI-NG -lfedtime
```

For HLA 1516, link against:

```
-LloggerDir/lib -lliblgrUtil -lliblgrCore -lliblgrApp
-llibappEnv -lliblgrGui
loggerDir/lib/lgrSimGuiHLA1516.so -L$(MAK_VRLDIR)/lib
-lvlHLA1516 -lvl -lmatrix -lmtl -lvlutil
-L$(MAK_RTIDIR)/lib -lrtl1516 -lfedtime1516
```

For HLA Evolved, link against:

```
-LloggerDir/lib -lliblgrUtil -lliblgrCore -lliblgrApp  
-llibappEnv -lliblgrGui  
loggerDir/lib/lgrSimGuiHLA1516e.so -L$(MAK_VRLDIR)/lib  
-lvlHLA1516e -lvl -lmatrix -lmtl -lvlutil  
-L$(MAK_RTIDIR)/lib -lrtil516e -lfedtime1516e
```

Linking for DIS Exercises

For DIS, link against:

```
-LloggerDir/lib -lliblgrUtil -lliblgrCore -lliblgrApp  
-llibappEnv -lliblgrGui loggerDir/lib/lgrSimGuiDIS.so  
-L$(MAK_VRLDIR)/lib -lvlDIS -lvl -lmatrix -lmtl -lvlutil
```

Each example in *./examples* has an example makefile.

Follow the directions in the VR-Link documentation for compiling and linking with VR-Link. The Logger library link flag (and the enumeration of the Logger libraries) must come before the VR-Link linking flag.

12.8.2. Compiling and Linking for Microsoft Windows

On Windows you must add the paths to the Logger's *include* and *lib* directories to the *include* and *lib* paths in your project file. We distribute both debug and release versions of the Logger DLLs. The example link lines in this section are for the release libraries. To build for debug, use the *d* versions of the libraries.

Linking for HLA

For HLA 13, release configuration, link against:

```
lgrUtil.lib lgrCore.lib lgrApp.lib appEnv.lib lgrGui.lib  
lgrSimGuiHLA13.lib (Logger libraries)  
v1HLA13.lib vl.lib matrix.lib mtl.lib vlutil.lib (VR-Link  
libraries)  
libRTI-NG.lib libFedtime.lib (RTI libraries)
```

For HLA 1516, release configuration, link against:

```
lgrUtil.lib lgrCore.lib lgrApp.lib appEnv.lib lgrGui.lib  
lgrSimGuiHLA1516.lib (Logger libraries)  
v1HLA1516.lib vl.lib matrix.lib mtl.lib vlutil.lib (VR-Link  
libraries)  
librti1516.lib libfedtime1516.lib (RTI libraries)
```

For HLA Evolved, release configuration, link against:

```
lgrUtil.lib lgrCore.lib lgrApp.lib appEnv.lib lgrGui.lib  
lgrSimGuiHLA1516e.lib (Logger libraries)  
v1HLA1516e.lib vl.lib matrix.lib mtl.lib vlutil.lib (VR-  
Link libraries)  
librti1516e.lib libfedtime1516e.lib (RTI libraries)
```

Linking for DIS

For DIS, release configuration, link against:

```
lgrUtil.lib lgrCore.lib lgrApp.lib appEnv.lib lgrGui.lib  
lgrSimGuiDIS.lib (Logger libraries)  
v1DIS.lib vl.lib matrix.lib mtl.lib vlutil.lib (VR-Link librar-  
ies)
```

12.8.3. Building Examples

The Logger includes several examples. Each example has project files that you can use to build it.



To build the Logger application and state observer examples that use the Logger GUI libraries, you must use the Qt Toolkit.



Remote Control API

The Remote Control API lets you can control the Logger from a remote application.

Overview	13-2
Using Multiple Loggers	13-3
Connecting to Loggers	13-3
Inspecting Logger State	13-4
Sending Commands Using the Remote Control Interface	13-5
Invoking Actions Using Domain Interfaces	13-6
Processing Responses	13-7
File Naming Conventions	13-8
Using the Remote Control Interface	13-8
Configuring the Logger to use the Remote Control API	13-9
Configuring the Remote Control Interface	13-10

13.1. Overview

The Logger remote control interface is a set of classes within the Logger API that can be used to remotely control one or more Loggers from a remote application. The interface automatically connects to a specified number of remote Loggers. It provides methods to invoke basic commands such as play, record, pause, stop, rewind, and so on, that are sent to connected Loggers. It receives and collects acknowledgements (indication of command reception) and responses (indication of command execution) from all the Loggers and presents them in unified callbacks.

The interface is broken down into a primary remote control interface that handles sending and receiving messages, and domain-specific interfaces for the various Logger controls. The remote control interface is implemented by the class *DtRemoteControlInterface* (in *remoteControlInterface.h*). The domain interfaces are derived from *DtRemoteControlDomainInterface* (*remoteControlDomainInterface.h*), for example, *DtRemoteControlRecordInterface* (*remoteControlRecordInterface.h*) or *DtRemoteControlPlaybackInterface* (*remoteControlPlaybackInterface.h*).

The remote control interface creates the domain interfaces using virtual creators and provides accessors to them. The domain interfaces provide methods to invoke basic Logger actions (for example, openFile, play, pause, and so on). Calls to these methods are converted into Logger commands, which are then transmitted over the network using the remote control interface. The domain specific interfaces also have methods that are called by the remote control interface in response to messages received from the Logger indicating changes in the Logger state. Derivation of the domain interfaces allows these methods to be overridden with customized behavior in order for the controlling application to react to the responses.

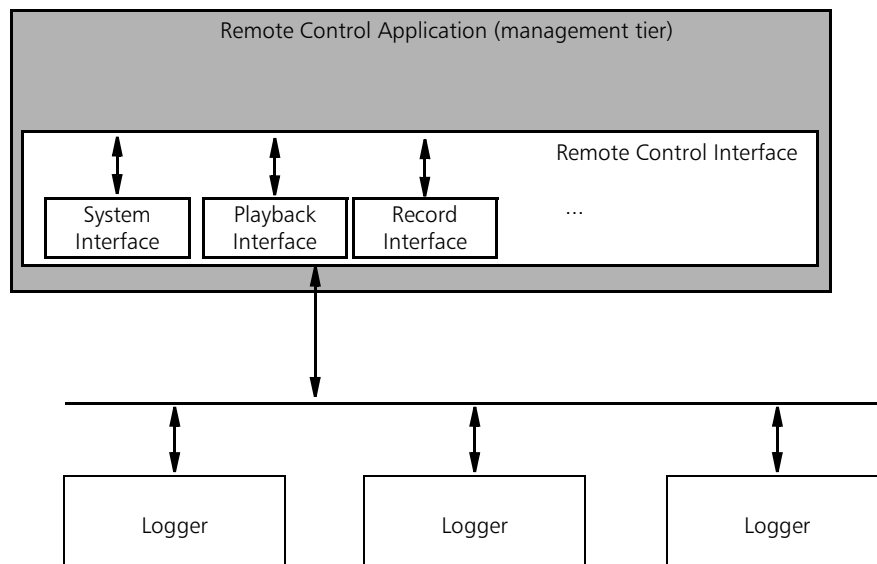


Figure 13-1.

The example Logger control application demonstrates the use of the interface.

13.1.1. Using Multiple Loggers

The primary purpose of using multiple Loggers is to distribute the load among multiple Logger applications when the amount of simulation data would overwhelm a single Logger. The best approach for segregating simulation data is the use of DDM. However, the configuration of DDM can be complicated for both the Logger and the participating federates.

Another purpose for using multiple Loggers is for fault tolerance.

To configure multiple Loggers, in *lgrConfig.xml*, do the following:

- ♦ Enable `processRemoteControl`.
- ♦ Enable `sendResponses`.
- ♦ Specify a unique remote control ID.

13.2. Connecting to Loggers

When you construct a remote control interface, the number of Loggers expected to connect is specified in the constructor. The value can be modified later. If the number of Loggers specified is greater than 0, then only that many Loggers will be allowed to connect. The default value is 0, which indicates there are no restrictions on Logger connections.

```
DtRemoteControlInterface* remoteControl = new
DtRemoteControlInterface(appInit.numberOfLoggers());
```

The remote control interface (*DtRemoteControlInterface*) uses a VR-Link exercise connection to send and receive Logger control interactions. The remote control interface:

- ♦ Automatically establishes connections with remote Loggers by listening for connection messages and responding with connection established messages.
- ♦ Automatically disconnects Loggers when receiving a disconnect message.
- ♦ Maintains a list of the connected Loggers.
- ♦ Processes Logger heartbeat messages to determine if any connections become stale, in which case they are timed out and removed.

The connect and heartbeat transmission interval is configurable in the Logger using the `SetRemoteControlSettings` command. For details, please see “[Simulation Domain Commands](#),” on page 11-10. The interface provides a method to configure the timeout interval (`DtRemoteControlInterface::setConnectionTimeout()`).

```
<command domain="Simulation"
  protocol="None" command="SetRemoteControlSettings">
  <param name="process" var="processRemoteControl"></param>
  <param name="record" var="recordRemoteControl"></param>
  <param name="allowDelete" var="allowDelete"></param>
  <param name="allowAll" var="allowAll"></param>
```

```
<param name="sendResponses" var="sendResponses"></param>
<param name="crippleGui" var="crippleGui"></param>
<param name="connectSendInterval" var="connectSendInterval"></param>
<param name="heartbeatSendInterval"
    var="heartbeatSendInterval"></param>
</command>
```

You can install callbacks for when the interface receives the connect, disconnect, heartbeat, and timeout events, as well as when a Logger total state message is received. The following code (from the remote control example) outputs the content of a Logger connection when it has timed out:

```
...
remoteControl->addCallbackConnectionTimedOut(timeoutCallback, 0);
...

void timeoutCallback( const DtRemoteLoggerConnection& loggerConnection,
    void* usr )
{
    DtWarn << "Logger Connection timed out\n";
    loggerConnection.printDataToStream(DtWarn);
}
```

13.3. Inspecting Logger State

Loggers send heartbeats periodically. They send total state interactions by request. They contain the overall state of the Logger. This information is maintained in a list of remote connection records. Entries for these records can be inspected. The entries are updated when heartbeats are received or the Loggers can be polled using a request state interaction (`DtRemoteControlSystemInterface::requestStateUpdate()`). As mentioned in “[Connecting to Loggers](#),” on page 13-3, you can register callbacks for the interactions. Alternatively, the system interface `receivedTotalState()` method can be overridden to process the list of connections when all of them have been updated by a total state interaction. The following code is from the `lgrControl` application.

```
virtual void receivedTotalState()
{
    DtEntityIdSet connectedLoggers;
    myInterface->getConnectionIds( connectedLoggers );

    if (connectedLoggers.size() == 0)
    {
        DtInfo << "No Loggers Connected.\n";
    }
    else
    {
        DtInfo << "Connected Loggers\n";
        DtEntityIdSet::const_iterator idIter = connectedLoggers.begin();
        DtEntityIdSet::const_iterator lgrEnd = connectedLoggers.end();
        for ( ; idIter != lgrEnd; ++idIter)
        {
            DtRemoteLoggerConnection* connection =
                myInterface->getConnection(*idIter);
            if (connection)
            {
                connection->printDataToStream(DtInfo);
            }
        }
    }
}
```

```
        }  
    }  
    DtInfo << std::endl;  
}  
}
```

13.4. Sending Commands Using the Remote Control Interface

The remote control interface provides a method to send Logger commands to remote Loggers (constructed using Logger command factories, for example, open file). The commands can be addressed to individual Loggers or to groups of Loggers using DIS-like wildcard values. (The default is to send to all remote Loggers.) Each sent command has an associated sequence number passed all the way up to the invoking remote control application. The remote Loggers return an acknowledge message with the sequence number when a command is received. In some cases, the remote Logger may indicate that the command is rejected, but usually the acknowledge message merely indicates that the command message was received. The remote control interface collects the acknowledgements from all the Loggers and invokes a single accept callback when they have been received. If one or more Loggers fail to acknowledge, the remote control interface invokes a “command timed out” callback. Each command rejection results in a “command reject” callback.

13.5. Invoking Actions Using Domain Interfaces

A remote control application invokes Logger actions on the appropriate domain interface (for example, play, on the playback interface). The domain interface constructs the Logger command and passes it to the remote control interface. The domain interface passes a reference to itself along with the command. The remote control interface takes a snapshot of the connected Loggers when a command is sent and associates the set with the sequence number, the invoking domain interface, and a timeout parameter. When acknowledgements have been received as accepted for all the connected Loggers, an “acknowledge accepted” callback is called on the interface that sent the command. The command sequence number is provided as a parameter. Each rejected acknowledge message generates a separate “acknowledge rejected” callback to the invoking domain interface. A “command timeout” callback is called if the remote control interface does not receive the acknowledge messages within the timeout period.

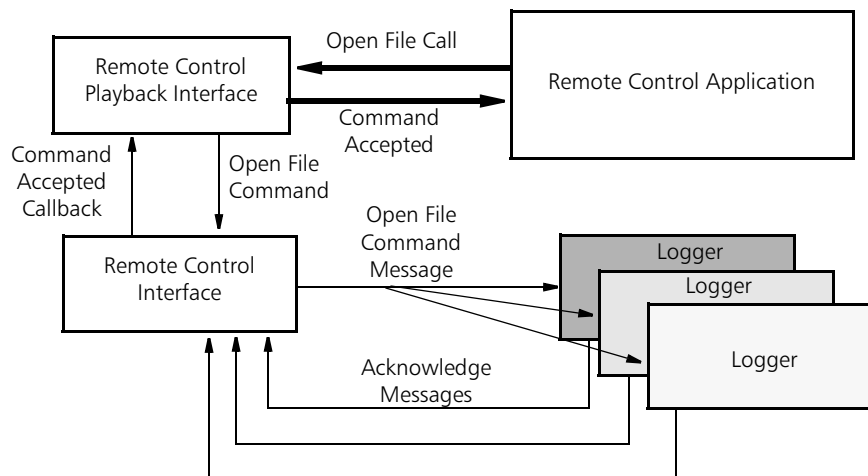


Figure 13-2. Issuing commands

13.6. Processing Responses

When the Logger processes a command, it generates internal responses indicating state changes (for example, file opened). It also sends these responses to the network, where the remote control interface receives them. The responses are domain specific and the remote control interface forwards them to the appropriate domain interface. The domain interfaces unify the separate Logger responses into a single response call after receiving one from each connected Logger. In some cases, a response indicates a failure condition (for example, file invalid when expecting file opened). Failure responses are invoked separately for each Logger and they may short circuit an expected response. The responses may modify the domain interfaces state information to reflect the reported state changes (for example, filename, firstTime, lastTime). The state changes can alter the behavior of the interfaces to allow or not allow a given command to become active (for example, play is not active until a fileOpened response has been received). When a command is not active, it returns a sequence number of 0. The state information may be inspected at any time.

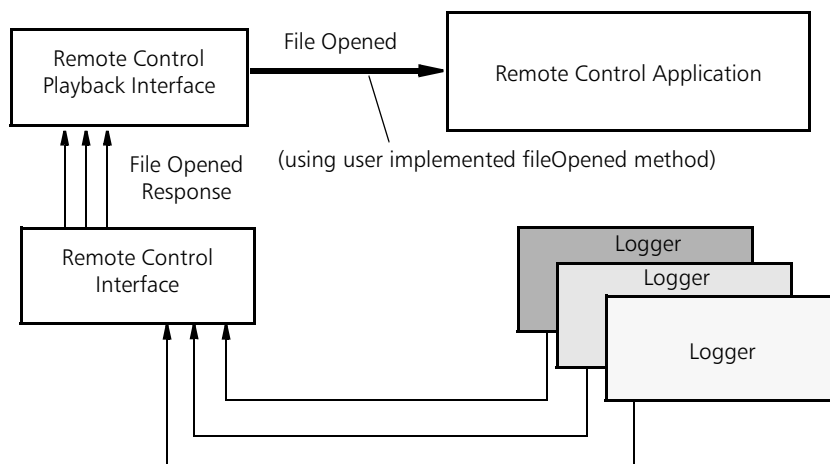


Figure 13-3. Handling responses

The remote control application can process the responses by overriding the appropriate method in the domain specific interface. The method override must make a call to the base class method so that it can perform its operation first.

```

void DtPlaybackInterfacePrinter::fileOpened( const DtString& fileName,
      const DtAbsoluteTime& firstTime, const DtAbsoluteTime& lastTime)
{
    DtRemoteControlPlaybackInterface::fileOpened(
        fileName, firstTime, lastTime);
    DtInfo << "All Loggers report fileOpened: " << fileName << " first:
        " << firstTime << " last: " << lastTime << std::endl;
}
  
```

13.7. File Naming Conventions

When sending a command to open a file (`newFile` for recording or `openFile` for playback), the remote control interface can specify a specific filename or it can construct a filename. If a specific filename is chosen, each connected Logger will attempt to open the file provided in the open file command. This approach will be sufficient if there is a single connected Logger or if each connected Logger is storing its file in a separate location (either separate installs or started with separate default Logger file directory locations).

For example, given the default directory `./logger_tapes`, the command

```
playbackInterface->openFile("maklanddemoHLA1516_vrlink_xml.lgr");
```

opens the file `./logger_tapes/maklanddemoHLA1516_vrlink_xml.lgr`.

Alternatively, the Logger can be instructed to construct unique filenames for each remotely controlled Logger using the Logger's remote control ID. When filenames are constructed, the Logger will create an exercise directory in the default file directory. It will then create filenames using the specified name and appending `_{id}.lgr` to the end.

For example, the command

```
playbackInterface->openFile("testExercise1");
```

will open the file `./logger_tapes/testExercise1_1.lgr` for the remote Logger with remote control ID equal to 1.

13.8. Using the Remote Control Interface

The example Logger control application demonstrates the use of the interface. It is a console application with a simple text command interface. It can open files for record or playback and perform the basic commands such as pause, stop, play, set time, and so on. It overrides the domain specific response methods to simply print out the response it received.

```
void DtPlaybackInterfacePrinter::fileOpened(
    const DtString& fileName, const DtAbsoluteTime& firstTime,
    const DtAbsoluteTime& lastTime)
{
    DtRemoteControlPlaybackInterface::fileOpened(
        fileName, firstTime, lastTime);
    DtInfo << "All Loggers report fileOpened: " << fileName << " first: "
        << firstTime << " last: " << lastTime << std::endl;
}
```

13.9. Configuring the Logger to use the Remote Control API

The following variables and commands should be added to the Logger configuration file (*lgrConfig.xml*). Each Logger must have a unique remote control ID. The connect and heartbeat send intervals can be adjusted if necessary. Set the defaultTapeDir to the directory where Logger files are stored. The Logger's execution or working directory will be used in conjunction with the default file directory if a relative path is specified.

```
<defaults>
...
<var name="remoteControlId" type="int" value="1"/>
<var name="processRemoteControl" type="bool" value="true"/>
<var name="recordRemoteControl" type="bool" value="false"/>
<var name="allowDelete" type="bool" value="true"/>
<var name="allowAll" type="bool" value="false"/>
<var name="sendResponses" type="bool" value="true"/>
<var name="connectSendInterval" type="double" value="1.0"/>
<var name="heartbeatSendInterval" type="double" value="10.0"/>
<var name="defaultTapeDir" type="string" value="..\logger_tapes"/>
<var name="crippleGui" type="bool" value="true"/>
...
</defaults>

<commands name="init">
...
<command domain="System" protocol="None"
  command="SetDefaultTapeDirectory">
  <param name="defaultTapeDir" var="defaultTapeDir"></param>
</command>
...
</commands>

<commands name="startup">
...
<command domain="Simulation" protocol="None"
  command="SetRemoteControlSettings">
  <param name="process" var="processRemoteControl"></param>
  <param name="record" var="recordRemoteControl"></param>
  <param name="allowDelete" var="allowDelete"></param>
  <param name="allowAll" var="allowAll"></param>
  <param name="sendResponses" var="sendResponses"></param>
  <param name="crippleGui" var="crippleGui"></param>
  <param name="connectSendInterval" var="connectSendInterval"></param>
  <param name="heartbeatSendInterval"
    var="heartbeatSendInterval"></param>
</command>

<command domain="Simulation" protocol="HLA 1516"
  command="SetRemoteControlId">
  <param name="remoteControlId" var="remoteControlId"></param>
</command>
...
</commands>
```

13.10. Configuring the Remote Control Interface

The remote control interface is configured through several methods provide by the core remote control interface class. The number of Loggers can be specified by the constructor.

You can configure the number of Loggers in the constructor, as follows:

```
DtRemoteControlInterface(int numberOfLoggersToManage=0)
```

The following methods provide further configuration:

- ♦ `setConnection()`. This method provides the interface with an exercise connection and specifies the remote control identifier. This method must be called after the remote control interface is instantiated and before any Logger commands are invoked.

```
void DtRemoteControlInterface::setConnection(DtExerciseConn* exconn,  
const DtEntityIdentifier& id)
```

- ♦ `setID()`. Change the remote control identifier to the given value.

```
void DtRemoteControlInterface::setId( const DtEntityIdentifier& id )
```

- ♦ `setAcceptAnyId()`. By default, the remote control interface will only accept messages whose receiver ID matches its ID. Setting this parameter to true will allow the remote control interface to accept any Logger control message.

```
void DtRemoteControlInterface::setAcceptAnyId( bool acceptAny)
```

- ♦ `setConnectionTimeout()`. Set the interval used to determine that a connected Logger is no longer alive. The remote control interface will remove a Logger connection if it has not received a heartbeat message from the Logger for a time greater than the given interval.

```
void DtRemoteControlInterface::setConnectionTimeout(DtTime val)
```

- ♦ `setNumberOfLoggers()`. Change the number of Loggers that are expected to connect.

```
void DtRemoteControlInterface::setNumberOfLoggers(  
int numberOfLoggersToManage )
```

- ♦ `addCallback()` and `removeCallback()`. Add or remove a callback for when a given message type is received for processing (** represents the message type, for example, `LgrConnect`, `LgrHeartbeat`, and so on).

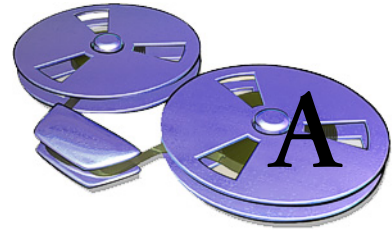
```
void DtRemoteControlInterface::addCallback( DtLgr**Cb cb, void* user )  
void DtRemoteControlInterface::removeCallback( DtLgr**Cb cb)
```

- ♦ `addCallbackConnectionAdded()`, `addCallbackConnectionRemoved()`, `addCallbackConnectionTimedOut()`, and `removeCallbackConnectionAdded()`, `removeCallbackConnectionRemoved()`, `removeCallbackConnectionTimedOut()`. Add or remove a callback for when a Logger connection is added, removed, or timed out. The callbacks for adding and removing a connection are somewhat redundant to the callbacks for the *DtLgrConnectPdu* and *DtLgrDisconnectPdu*. However, the callback for connection added is called only when the connection is added. It may happen that the remote control interface receives multiple *DtLgrConnectPdus* from a single Logger. Since the connect message does not contain the Logger total state information, the connection info is initialized with default values. The callback is called after the connection is added to the list. The connection removed callback is included for symmetry and a convenience. It is called before the connection is removed from the list.

```
void DtRemoteControlInterface::  
    addCallbackConnectionAdded(DtLgrConnectionAddedCb cb, void* user )  
void DtRemoteControlInterface::  
    removeCallbackConnectionAdded(DtLgrConnectionAddedCb cb)
```

i

You would use the same syntax for `addCallbackConnectionRemoved()`, `addCallbackConnectionTimedOut()`, `removeCallbackConnectionRemoved()`, and `removeCallbackConnectionTimedOut()`.



Troubleshooting

This appendix describes how to identify and correct the most common network problems that prevent the Logger from sending and receiving data properly.

Troubleshooting Logger Network Communications (DIS Only).....	A-2
Using the Correct Broadcast Address and UDP Port	A-2

A.1. Troubleshooting Logger Network Communications (DIS Only)

In a typical DIS exercise, VR-Link applications (including the Logger) communicate with other applications using broadcast UDP/IP over a local area network. All applications in the exercise use the same IP broadcast address, and each application's host machine has a network interface device that is configured for that address. All machines must share a common netmask.

In addition, all participating applications must use the same UDP port.

If the Logger uses an IP address, netmask or port that is different from that used by your other applications, communication problems occur.

A.1.1. Using the Correct Broadcast Address and UDP Port

By default, the Logger communicates to the host machine's default broadcast address, using UDP port 3000. If the Logger appears to start normally, but fails to communicate with other DIS applications, the Logger may be using an incorrect broadcast address or UDP port. You can correct this problem by running the Logger with arguments that explicitly specify the correct address and port:

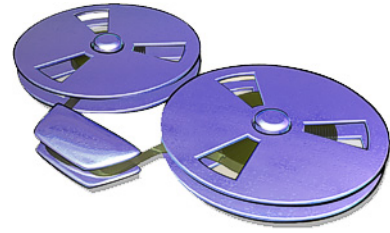
1. Determine the broadcast address used by your other DIS applications. This may be the network's default broadcast address, or may be specified through the application's startup command or configuration file at startup. For more information, please see your System Administrator, or use the Linux `ifconfig` command to display the broadcast address.
2. Determine the UDP port used by your other applications.
3. Start the Logger and notice the status message that appears at startup:

```
NetSocket ok for port 3000 to 123.456.789.255
```
4. If the port and address in the status message do not match those used by your other applications, exit the Logger and restart using the Address option (`-A`), the DIS port option (`-P`), or both, to specify the appropriate values. Enter:

```
logger -A address -P port
```

where address and portnum are the broadcast address and UDP port used by your other applications, for example:

```
logger -A 123.456.999.255 -P 3020
```



MÄK Products Glossary

This glossary defines terms used in MÄK product documentation.

absolute dead reckoning

A method of [dead-reckoning](#) in which an application uses the time stamps contained within state updates if the sender is using absolute time stamping. Contrast with [relative dead reckoning](#).

aggregate

The combination of individual entities, aggregates, or both into a single object. For example, an organizational group such as a platoon.

AMO

See [Application Management Object](#).

Application Management Object

An Application Management Object (AMO) is a standard TENA object that is supposed to be used by every TENA application. It gives other applications information about the local application. The VR-Link exercise connection, by default, publishes an AMO object, updates it at a specified rate (the rate is part of the AMO's state repository; the default is 30 seconds), and updates the number of objects published and proxies (TENA reflected objects) held.

API

Application Programming Interface.

articulated part

A part of an entity that is capable of movement relative to the entity, such as a turret or gun.

attached part

An object that is attached to another object, such as a rocket attached to a jet.

body coordinate frame

A coordinate framework centered on an entity. To represent the orientation of an entity, an application uses Euler angles or a rotation matrix that rotates from a reference frame to the body coordinate frame.

channel

A channel renders the view of an observer in a 3D visualization application.

clipping

A feature that prevents the display of terrain and objects or parts of objects, that are closer than or farther than specific distances from the observer.

clipping planes

The range of distances from the observer (near clipping and far clipping) in which an application clips objects.

culling

The process of discarding database objects which are not within view, and therefore need not be rendered and displayed.

Cartesian coordinates

A system for indicating location by means of three planes intersecting at right angles to one another at the origin.

Data Distribution Management

The set of HLA services used to specify the routing of data between publishers and subscribers. DDM adds greater control over the Declaration Management (DM) services that only specify filtering based on the class of the data (e.g., ground vehicle versus aircraft). DDM uses the data content to express regions of interest and regions of affect whose overlap establish a communication channel. An example is geographic filtering where a publisher indicates a region around the entity or effect and a subscriber indicates a region expressing its sensor range.

dead-reckoning

A process by which an application calculates the expected location of an entity during periods between state updates, based on velocity, acceleration, and rotational velocity.

DDM

[Data Distribution Management.](#)

Defense Modeling and Simulation Office

The former name of the executive secretariat for the Executive Council on Modeling and Simulation (EXCIMS), which provided a full-time focal point for information concerning Department of Defense modeling and simulation (M&S) activities. Renamed Modeling and Simulation Coordination Office. Currently the MSCO promulgates M&S policy, initiatives, and guidance to promote cooperation among DoD components to maximize efficiency and effectiveness.

DIS

[Distributed Interactive Simulation](#).

DIS data packets

See [Protocol Data Unit](#).

display engine

An object in an application that can render 3D data.

Distributed Interactive Simulation

The DIS protocol is a set of standards that govern how participating applications share information about the virtual world. The protocol specifies a set of packets, called protocol data units (PDUs), that communicate this information. Each PDU identifies the sender and contains other information depending on the PDU type. The DIS protocol also specifies when and how frequently PDUs are sent.

The DIS protocol has been superseded by the [High-Level Architecture](#) for use by the Department of Defense.

DMSO

See [Defense Modeling and Simulation Office](#).

emitter

A simulated device on an entity that emits electromagnetic radiation, for example, radar.

entity

An element in a simulation, such as a vehicle or a person, that is represented in the simulation through the issuance of state messages.

entity maintenance

A process by which the MÄK Data Logger compensates for discontinuities following a time jump. The Logger sends out interim state messages for entities that were present before the time jump so that they do not time out before the next update message arrives.

entity-type

A list of seven components as specified by the DIS protocol and the HLA RPR FOM. If none of the seven components contains a wildcard (-1) value, then the entity type refers to a specific, narrowly-defined entity. If some components contain a wildcard, then the entity type refers to a class of entities. A larger number of wildcards indicates a broader, more general class.

The components of an entity-type are:

- ♦ Entity kind
- ♦ Domain
- ♦ Country
- ♦ Category
- ♦ Subcategory
- ♦ Specific
- ♦ Extra.

environmental process

According to the IEEE 1278.1a specification (DIS), environmental process PDUs communicate simple environment variables, small scale environmental updates, and embedded processes.

Euler angles

A set of three angles used to describe the orientation of an entity as a set of three successive rotations about three different orthogonal axes (x, y, and z). These angles specify successive rotations needed to transform from a reference coordinate system to the entity's body coordinate system.

event

An interaction between objects or between an object and the terrain, such as firing of a munition, or a collision of entities.

execution

The TENA term for one or more interacting simulation applications.

Execution Manager

An Execution Manager governs a TENA execution for applications joining, resigning, or changing subscriptions to that execution.

exercise

The DIS term for a one or more interacting simulation applications. Compare to [federation execution](#) in HLA.

exercise connection

The object (*DtExerciseConn*) through which a VR-Link application connects to the network. State messages are sent through the exercise connection and information about remote entities is received through the exercise connection. (All MÄK products are VR-Link applications.)

exercise ID

A numeric identification for a DIS simulation exercise.

FED file

federate

A connection to the [RTI](#). Typically a single simulation application can be thought of as a federate.

federation

A group of HLA federates capable of playing in the same [federation execution](#).

Federation Object Model

Defines the data content of a federation execution.

federation execution

The federation execution represents the actual operation, over time, of a subset of the federates and the RTI initialization data taken from a particular federation. A federation execution is the logical equivalent of a DIS simulation exercise.

field of view

Controls the perspective of the observer.

A wide field of view creates an effect like that of a wide-angle camera lens. Objects appear smaller and farther away from the observer, since the observer coverage spans a wider area. Depth become exaggerated.

A narrow field of view creates an effect like that of a telephoto lens. Objects appear larger and closer to the observer, and the overall scene depth appears flattened. The distances between objects appears compressed.

filter range

A setting that prevents distant entities from being processed while allowing distant terrain to appear normally.

FOM

See [Federation Object Model](#).

FOM Module

In HLA Evolved, defines additional modular data content for a federation execution, usually extensions to an existing FOM. (Also supported by the HLA 1.3 and HLA 1516 versions of the MÄK RTI. Per the HLA Evolved interface specification:

“A partial FOM (containing some or all OMT tables) that specifies a modular component of a FOM. A FOM module may contain classes not inherent to it but upon which the FOM module depends, i.e., superclasses to the modular components. These superclasses will be included in the FOM module either as complete or scaffolding definitions.”

frame rate

The rate at which the application displays updated images.

ground clamping

A process by which the a 3D visualization application keeps an entity anchored to the surface of its terrain database, regardless of the altitude data contained in its entity state message.

geocentric coordinates

A coordinate system calculated with respect to the earth's center. The origin of the geocentric coordinate system is the center of the earth. The positive X-axis passes through the prime meridian at the equator; the positive Y-axis passes through 90 degrees east longitude at the equator; and the positive Z-axis passes through the north pole.

geodetic coordinates

A coordinate system in which position is determined relative to a reference ellipsoid, such as the surface of the earth at sea level. In MÄK applications, geodetic coordinates consist of latitude and longitude in radians, and altitude in meters above the reference ellipsoid.

gridded data

Data that has been processed in a rectangular array of points, in X, Y or latitude/longitude, at which single data values define a two dimensional function. According to the IEEE 1278.1a specification, gridded data transmits information about large-scale or high-fidelity spatially and temporally varying ambient fields and about environmental processes and features.

guise

An alternative entity type used to display an object depending on the force ID. For example, a tank could look like an M1A1 to friendly forces and a T72 to hostile forces.

head-up display

A set of indicators and readouts superimposed onto a graphics display. Also called an overlay.

heartbeat

In DIS, the frequency with which current PDUs are sent to the network regardless of whether or not the entity's state has changed.

High-Level Architecture

The High Level Architecture (HLA) for simulations is a U. S. Department of Defense (DOD)-wide initiative to provide an architecture to support interoperability and reuse of simulations. The HLA supersedes DIS for the DOD.

HLA

See [High-Level Architecture](#).

interaction

A [message](#) describing a simulation event. An interaction describes an event, it does not update an object's state.

Logger control PDUs

logical range

A suite of TENA Resources, sharing a common object model, that work together for a given range event. Similar in concept to the HLA Federation.

Local RTI Component

The libraries on a computer that implement an RTI. A federate communicates with the HLA network through the LRC.

Logical Range Object Model

The data definition file for TENA exercises. It contains the set of object and message definitions that may be used in the exercise.

LRC

See [Local RTI Component](#).

LROM

See [Logical Range Object Model](#).

MÄK Technologies Lisp

An adaptation of the Lisp language used in configuration files for MÄK products.

message

A general term used to refer to [DIS PDUs](#) and [HLA interactions](#) and state updates. In TENA, a message is similar to an HLA interaction.

MIM

The MOM & Initialization Module (MIM) allows for extensions to be made to the HLA standard MOM. It is a subset of the FOM that contains OMT tables that describe the HLA MOM. The MIM also contains additional predefined HLA constructs such as object and interaction roots, data types, transportation types and dimensions. HLA specifies a standard MIM that is incorporated into all FDDs automatically by the RTI. The standard MIM can be replaced with a user-supplied MIM containing the standard MIM plus extensions.

Modeling and Simulation Coordination Office

The executive secretariat for the Executive Council on Modeling and Simulation (EXCIMS), which provided a full-time focal point for information concerning Department of Defense modeling and simulation (M&S) activities. The MSCO promulgates M&S policy, initiatives, and guidance to promote cooperation among DoD components to maximize efficiency and effectiveness. (Formerly DMSO)

MSCO

Modeling and Simulation Coordination Office. (Formerly DMSO.)

MTL

See [MÄK Technologies Lisp](#).

Network Naming Service

Acts as a registry for TENA Execution Managers.

NNS

See [Network Naming Service](#).

orientation clamping

Adjusts the pitch and roll of an entity so that it appears properly seated when the terrain is inclined. For example, if a tank is moving horizontally across the face of a hill, orientation clamping prevents the tank from appearing level, and therefore, partially embedded into the hillside. Used with [ground clamping](#).

object

An element in a simulation that has persistence, as opposed to an interaction, which is a transient element.

object handle

An integer that an application uses to identify a particular object in RTI service calls. An object handle is meaningful only to a particular federate. The same object can be known to different federates by different object handles.

object name

A character string that can be used to identify an object. In HLA, the object name is known to the RTI, and the RTI provides functions to find out an object's name, given its handle, and vice versa. Object names can be chosen by applications that register the objects with the RTI, however if you do not want to choose names for objects, the RTI will assign names for you.

observer

In MÄK 3D applications, the point of view into the scene. (Called the eyepoint in MÄK Stealth 6.x and earlier.)

PDU

See [Protocol Data Unit](#).

Protocol Data Unit

A unit of data message (packet) that is passed on a network between DIS simulation applications.

proxy

In TENA, a reflected object ([Stateful Distributed Object](#)).

radio transmitter

A simulated device on an entity, capable of transmitting radio communications.

radio receiver

A simulated device on an entity, capable of receiving radio communications.

range resource

A participant in a TENA execution (similar in concept to the HLA federate).

Realtime Platform Reference FOM

An [HLA](#) reference [FOM](#) based on the [DIS](#) protocol.

recording

A Data Logger file that stores a history of the interactions in a simulation for playback.

reflected entity list

A list of entities simulated by remote applications (federates in HLA) and about which the local simulation has received information over the network.

reference FOM

A [FOM](#) designed to be used as a whole, or with modification, by a wide variety of similar [federation executions](#).

relative dead reckoning

A method of dead reckoning in which an application approximates the location of an object based on the local time of receipt of the last state update message.

remote display engine

A display engine running in an application that is separate from the master application, and which is controlled by the master application.

RID file

See [RTI Initialization Data](#).

RTI Initialization Data

The initialization data required by the RTI for operation.

Data required by an RTI during initialization, independent of the FOM being used. RID data is usually dependent on a specific implementation of the RTI.

RPR FOM

See [Realtime Platform Reference FOM](#).

RTI

See [Run-Time Infrastructure](#).

Run-Time Infrastructure

A library and other supporting software that implements the HLA interface specification. All federates communicate with one another in an HLA environment through RTI functions.

SDO

See [Stateful Distributed Object](#).

servant

A published object (SDO) in TENA.

Simulation Interoperability Standards Organization

A group that seeks to promote modeling and simulation interoperability and reuse for the benefit of diverse M&S communities, including developers, procurers, and users, world-wide.

simulation time

In VR-Forces, simulation time is used in dead-reckoning of remote entities and thresholding of local entities. Typically, simulation time is set once during each iteration of the application's main simulation loop so that all entities are dead-reckoned based on the same value of current time.

SISO

See [Simulation Interoperability Standards Organization](#).

smooth period

The period of time over which [trajectory smoothing](#) takes place.

smoothing

A method of ensuring that transitions from an entity's dead-reckoned position to its actual position are not so abrupt as to be visually disconcerting.

state

The current status of an object, including location, direction of movement, extent of damage, and so on.

Stateful Distributed Object

An object in a TENA exercise.

tape

An alternative term for a Logger recording. Not a physical magnetic tape.

TENA

See [Test and Training Enabling Architecture](#).

TENA Middleware

terrain following

In a 3D visualization application, causes the observer (eyepoint) to maintain a constant distance above the terrain surface. The observer's height changes in tandem with the peaks and valleys as it passes over the geography.

Test and Training Enabling Architecture

Test and Training Enabling Architecture (TENA) is an interoperability architecture used to pass data in the form of object updates and messages (like HLA objects and interactions) from one application to another.

timeout

The period of time in which an application continues to display an entity after that entity's update messages have stopped appearing on the network. Time-outs are usually not used in HLA because there is no set frequency (no heartbeat) for transmitting messages.

timescale

A factor by which time is accelerated or slowed during playback of a Data Logger recording.

topographic coordinates

A right-handed Cartesian coordinate system whose X-Y plane is tangent to the earth's surface at the origin, with the positive X axis pointing north, the positive Y axis pointing east, and the positive Z axis pointing down. There are an infinite number of topographic coordinate systems – one for each point on the earth's surface.

topographic coordinate frame

A coordinate frame in the context of the terrain.

trajectory smoothing

A method used by to smooth positional discontinuities that could occur when new state updates arrive.

UDP port

A network channel through which an application sends and receives data for DIS exercises.

Universal Transverse Mercator

In general, a non-cartesian coordinate system in which the X, Y, and Z correspond to easting (nearly east), northing (nearly north), and height above an ellipsoid that approximates the surface of the earth.

UTM

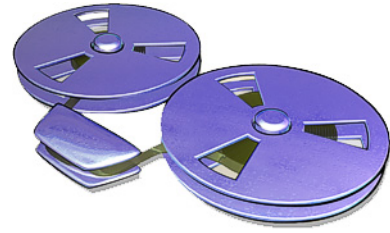
See [Universal Transverse Mercator](#).

view control messages

A set of programmatic messages that let you control the VR-Vantage remotely.

view floor

In MÄK Stealth 6.x and earlier, a minimum height above the terrain in Absolute or Track mode, below which the observer is not allowed to go. The view floor is measured relative to the terrain surface.



Index

- command-line option 5-23
 - appNumber command-line option 5-24
 - configFile command-line option 5-18, 5-23
 - destAddrString command-line option 5-24
 - deviceAddrString command-line option 4-21, 5-24
 - disPort command-line option 5-24
 - disVersion command-line option 5-24
 - execName command-line option 5-24
 - exercisId command-line option 5-24
 - federateName command-line option 5-24
 - federateType command-line option 4-7, 5-23
 - fomMapperInitData command-line option 4-12, 5-23
 - fomMapperLib command-line option 5-23
 - fomModules command-line option 4-9, 5-24
 - ignore_rest command-line option 5-23
 - ignoreAdvisories command-line option 5-24
 - logFileName command-line option 5-23
 - mcastAddresses command-line option 5-25
 - mcastTtl command-line option 4-21, 5-24
 - messageLevel command-line option 5-23
 - MIM command-line option 4-9, 5-24
 - notifyLevel command-line option 5-23
 - receiveBufferSize command-line option 4-20
 - recvBufferSize command-line option 5-24
 - rprFomRevision command-line option 4-10, 5-24
 - rprFomVersion command-line option 4-10, 5-24
 - runCommand command-line option 5-23
 - sendBufferSize command-line option 4-20, 5-25
 - showConsole command-line option 5-23
 - siteId command-line option 5-25
 - useEmptyFomMapper command-line option 5-24
 - version command-line option 5-23
- ## A
- A command-line option 4-19, 5-24
 - a command-line option 4-18, 5-24
 - acceleration scaling 3-7
 - accept, callback 13-5
 - Access
 - configuring data source for 10-8
 - exporting to 10-7
 - importing data to Excel 10-18
 - acknowledge, message 13-5
 - acknowledge accepted, callback 13-6
 - acknowledge rejected, callback 13-6
 - addCallback() function 13-10
 - addCallbackConnectionAdded() function 13-11
 - addCallbackConnectionRemoved() function 13-11
 - addCallbackConnectionTimedOut() function 13-11
 - addCommandHandler() function 12-4
 - AddCustomPDU command 11-15
 - adding
 - annotation 7-2
 - command to *lgrConfig.xml* 5-21
 - instant annotation 7-3
 - variable to *lgrConfig.xml* 5-21
 - address, multicast 5-25
 - advisory message
 - configuring 11-14
 - ignoring 4-11
 - allowPlaybackWithoutRegions parameter 5-15, 11-13

- annotating, file 7-2
- annotation 7-2
 - bounding playback to 7-9
 - converting type 7-6
 - creating 7-2
 - deleting 7-6
 - editing 7-5
 - importing annotation file 7-5
 - instant, adding 7-3
 - loading automatically 6-13
 - opening annotation file 7-5
 - saving file 7-4
 - selecting 7-5
 - updating 7-4
 - zooming timeline to 7-8
- API
 - Logger 12-3
 - plug-in 12-10
 - remote control 13-2
- AppendFileToEnd command 11-17
- appending, file 9-4, 11-17, 11-18
- appEnv library 12-18
- application
 - compiling and linking 12-19
 - embedding Logger in 12-7, 12-10
 - embedding plug-ins in 12-7
 - number 4-18, 11-16
 - specifying 5-24
- architecture, Logger API 12-4
- argument type 11-5
- asynchronous I/O 11-16
 - configuring 4-19
- attribute
 - annotation 7-5
 - connection 4-4
 - scaling, enabling or disabling 11-11
- automatic
 - playback 6-13
 - recording 6-8

B

- backup
 - enabling or disabling 11-7
 - file 6-17, 11-7
- bak file 6-17, 11-7
- beginning, moving to 6-20
- berkeleyDemo 1-4
- broadcast address 4-19, A-2
- buffer, send and receive 4-20

- building, example 12-21
- button bar 6-4

C

- c command-line option 5-23
- callback 13-4
 - accept 13-5
 - acknowledge accepted 13-6
 - acknowledge rejected 13-6
 - command reject 13-5
 - command timed out 13-5
 - command timeout 13-6
- canceling, recording 11-8
- CancelRecording command 11-8
- changing, connection settings 4-4
- checkpoint, skipping to 6-16
- checkpoint policy 5-2, 11-8
- class
 - DtAnyValue* 12-15
 - DtCommand* 12-5
 - DtCommandDefinition* 12-5
 - DtCommandSP* 12-14
 - DtDisLogger* 12-7
 - DtHlaLogger* 12-7
 - DtLgrAppGuiInterface* 12-18
 - DtLgrBaseSimManager* 12-6
 - DtLgrCommandHandler* 12-4
 - DtLgrDisSimulationManager* 12-6
 - DtLgrFullInterface* 12-5
 - DtLgrHlaSimulationManager* 12-6
 - DtLgrInterface* 12-3
 - DtLgrPlaybackCommand Factory* 12-5
 - DtLgrPlaybackDriver* 12-4
 - DtLgrPlaybackInterface* 12-5
 - DtLgrProtocolRegistration* 12-6
 - DtLgrSimInput* 12-6
 - DtLgrSimOutput* 12-6
 - DtLgrSystemDriver* 12-4
 - DtLogger* 12-3, 12-4, 12-6, 12-7, 12-10, 12-11, 12-12
 - DtLoggerApplication* 12-10, 12-11
 - DtMessageInput* 12-6
 - DtMessageOutput* 12-6
 - DtMessageState* 12-6
 - DtMutex* 12-16
 - DtObserveConnection* 12-17
 - DtObserveEvent* 12-17
 - DtObserver* 12-17
 - DtProtocolRegistration* 12-18

class (continued)

- DtRemoteControlInterface* 13-2
- DtRemoteControlInterface* 13-3
- DtRemoteControlPlaybackInterface* 13-2
- DtRemoteControlSystemInterface* 13-4
- DtResponse* 12-5, 12-14
- DtSimInput* 12-6
- DtString* 12-15
- DtThreadSafe* 12-16
- DtValue* 12-15
- DtValueSignal* 12-16
- factory 12-5
- filtering by (HLA) 8-6
- interface 12-5

classes, subscribing to 4-14

clearing

- file segment 9-2
- packets from file 11-17

ClearSection command 11-17

ClosePlaybackFile command 11-7

CloseRecordFile command 11-8

closing

- file 6-12, 11-7
- recording 11-8

command 5-2

- AddCustomPDU 11-15
- adding to *lgrConfig.xml* 5-21
- AppendFileToEnd 11-17
- CancelRecording 11-8
- ClearSection 11-17
- ClosePlaybackFile 11-7
- CloseRecordFile 11-8
- Connect 11-10
- ConvertFile 11-17
- CutSection 11-17
- defining, creating, and decoding 12-5
- DeleteFile 11-6
- Disconnect 11-10
- Editor domain 11-17
- Exit 11-6
- ExportFile 11-17
- ExportSection 11-18
- Help 11-6
- HLA simulation domain 11-11
- in configuration file 5-20
- list of 11-18
- LoadPlugin 11-6
- LoadSubscriptionListFile 4-16, 5-10, 11-11
- MergeInFile 11-18
- NewFile 11-8

command (continued)

- OffsetTime 11-18
- OpenFile 11-7
- Pause 11-7
- PauseRecording 11-8
- Play 11-7
- Player domain 11-7
- PlayPause 11-7
- PrefixFileToStart 11-18
- Record 11-8
- Recorder domain 11-8
- Restart 11-7
- RevertFile 11-7
- running at startup 5-23
- SaveFile 11-7
- SaveFileAs 11-7
- SaveSubscriptionListFile 4-15, 11-11
- sending 12-5
- sending to remote Logger 13-5
- SetBackupOnSave 6-17, 11-7
- SetCheckpointParameters 5-3, 11-8
- SetConnectionParameters 4-6, 4-17, 11-10, 11-12, 11-16
- SetCurrentTime 11-7
- SetDdmParameters 5-10, 5-11, 5-14, 11-13
- SetDisableTimeManagementTest 11-14
- SetDrainInput 11-14, 11-16
- SetDynamicNamePrefix 3-3, 11-14
- SetEntityTracking 5-4, 5-5, 11-6
- SetFastJumpsWithoutDiff 5-4, 5-5, 11-6
- SetHeartbeatIntervals 5-4, 5-7, 11-14
- SetHeartbeatRate 4-22, 11-16
- SetIgnoreAdvisories 4-11, 11-14
- SetLoopAtEnd 11-7
- SetNotifyLevel 5-17, 11-6
- SetPassiveSubscription 4-16, 11-15
- SetPauseOnRecordAs 11-9
- SetPlaybackReadTimeLimit 5-4, 5-6, 11-6
- SetPlaybackSpeed 11-7
- SetPlayOnLoad 11-8
- SetRemoteControlId 11-10
- SetRemoteControlSettings 11-10, 13-3
- SetScaleAttributes 11-11
- SetStartStopTimes 11-8
- SetStopCondition 11-9
- SetStopDuration 11-9
- SetStopSize 11-9
- SetSubscriptionLists 4-15, 5-22, 11-15
- SetTempDir 11-6
- SetTimeManagement 4-13, 11-15

command (continued)

- SetTimeoutIntervals 5-4, 5-7, 11-15
- SetTimeoutRate 4-23, 11-16
- SetVRFFiltering 11-11
- Simulation domain 11-10
- StopPlayback 11-8
- StopRecording 11-9
- syntax in *lgrConfig.xml* 5-20
- System domain 11-6
- TrimStart 11-18
- Unpause 11-8

command reject, callback 13-5

command timed out, callback 13-5

command timeout, callback 13-6

command-line interface 11-3

- domains 11-4

command-line option 5-2

- 5-23
- A 4-19, 5-24
- a 4-18, 5-24
- appNumber 5-24
- c 5-23
- configFile 5-18, 5-23
- D 5-24
- destAddrString 5-24
- deviceAddrString 4-21, 5-24
- disPort 5-24
- disVersion 5-24
- e 4-18, 5-24
- execName 5-24
- exerciseId 5-24
- F 4-8, 4-10, 5-24
- f 4-11, 5-23
- federateName 5-24
- federateType 4-7, 5-23
- filename 5-23
- fomMapperInitData 4-12, 5-23
- fomMapperLib 5-23
- fomModules 4-9, 5-24
- h 5-23
- ignore_rest 5-23
- ignoreAdvisories 5-24
- l 5-17, 5-23
- logFileName 5-23
- m 5-17, 5-23
- mcastAddresses 5-25
- mcastTtl 4-21, 5-24
- messageLevel 5-23
- MIM 4-9, 5-24
- N 4-7, 5-24

command-line option (continued)

- n 5-17, 5-23
- notifyLevel 5-23
- P 4-17, 5-24
- r 5-23
- receiveBufferSize 4-20
- recvBufferSize 5-24
- rprFomRevision 4-10, 5-24
- rprFomVersion 4-10, 5-24
- runCommand 5-23
- S 4-24, 5-25
- s 4-18, 5-25
- sendBufferSize 4-20, 5-25
- showConsole 5-23
- siteId 5-25
- table of 5-23
- useEmptyFomMapper 5-24
- v 5-23
- version 5-23
- x 4-8, 5-24

commonEntityTypes parameter 8-3

compiling

- API code 12-19

- application 12-19

component files 2-3

concatenating, files 9-4, 11-18

configuration file 5-18

- adding string list 5-22

- RTI 2-7

- specifying 5-23

configuration parameter 5-2

- useAsyncIO 4-19

configuring

- advisory messages 11-14

- application number 4-18

- asynchronous I/O 4-19

- data source for Microsoft Access 10-8

- data source for MySQL 10-11

- DDM 5-16

- DIS 4-20

- heartbeat 11-16

- timeout 11-16

- FED file 5-24

- filtering 11-11

- high speed playback 6-16

- HLA connection 11-12

- indexing 5-2

- list of common entity types 8-3

- configuring (continued)
 - Logger 5-2
 - for HLA 4-6
 - remote control interface 13-9
 - network read time 5-9
 - passive subscription 11-15
 - performance 5-9
 - playback, fast 5-4
 - Record As 11-9
 - recording 11-9
 - remote control interface 13-10
 - response to Logger Control messages 11-10
 - site ID 4-18
 - system for MÄK RTI 2-7
 - time management 11-15
- Connect command 11-10
- connecting 11-10
 - to exercise 4-4
 - to network 4-3
- connection
 - changing settings 4-4
 - configuring DIS 11-16
 - configuring HLA 11-12
 - editing 4-4
 - exercise 13-3
 - settings 4-3
 - status 4-3
- Connection dialog box 4-3
- console window 11-3
 - displaying 5-23
- constructing, filename 13-8
- control, Logger 6-4
- controlling
 - Logger, remotely 13-2
- ConvertFile command 11-17
- converting, annotation type 7-6
- conveyed region 5-13
- create() function 12-5, 12-15
- creating
 - annotation 7-2
 - command 12-5
 - subscription control list 4-14
- cursor
 - setting to time 11-7
 - timeline 6-20, 6-21
- custom PDU, filtering 8-4
- customPdus** parameter 8-4
- CutSection command 11-17
- cutting
 - file segment 9-2

- cutting (continued)
 - packets 11-17

D

- D command-line option 5-24
- data
 - exporting 10-2
 - exporting to database 10-3
 - exporting to spreadsheet 10-14
 - importing to Excel 10-18
- Data Distribution Management 5-10
- data source
 - Access 10-8
 - configuring for ODBC 10-5
 - MySQL 10-11
- data type, HLA 10-7
- Data View 6-21
- database
 - configuring ODBC data source 10-5
 - exporting data to 10-3
 - exporting HLA data to 10-7
 - importing to Excel 10-18
 - schema 10-3
- DDM 5-10
 - configuring 5-16
 - conveyed region 5-13
 - playback 5-13
 - subscription list, programming 5-12
- dead reckoning
 - acceleration scaling 3-7
 - velocity scaling 3-7
- dead space, removing 11-18
- decode() function 12-5
- decoding, command 12-5
- defining, command 12-5
- DeleteFile command 11-6
- deleting
 - annotation 7-6
 - file segment 9-2
 - packets 11-17
- descriptor, region 5-10
- destination address 4-19, 4-20, 11-16
- device address 11-16
- dialog box
 - Connection 4-3
 - License Setup 2-5
 - Set Start and End Time 6-14
 - Set Zoom 6-19

- directory
 - ./bin* 2-3
 - ./doc* 2-3
 - ./logger_tapes* 2-3
- DIS 1-6
 - application number 4-18
 - asynchronous I/O 4-19
 - configuring 4-20
 - configuring connection 11-16
 - exercise ID 3-3, 4-18
 - export to database 10-6
 - filtering
 - entity types 11-15
 - PDU's 11-15
 - heartbeat 4-22, 11-16
 - port
 - Logger default A-2
 - specifying 5-24, A-2
 - protocol version 5-24
 - site ID 4-18
 - timeout 4-23, 11-16
- disabling
 - attribute scaling 11-11
 - backup 11-7
 - entity tracking 11-6
 - high speed playback 6-16
 - looping playback 11-7
 - Pause on Record As 6-8
 - Play On Load 6-13
 - time management 11-14
- Disconnect command 11-10
- disconnecting 11-10
 - from exercise 6-6
- displaying
 - console window 5-23
 - object statistics 6-22
 - plug-in view 6-5
 - toolbar 6-5
 - version information 5-23
- docking, toolbars and plug-in views 6-5
- domain 5-21
 - associated with command 11-18
 - command 11-4
 - interface 13-2, 13-6
- domain interface, sending commands 13-6
- dragging, timeline cursor 6-21
- driver, in Logger API 12-4
- DtAnyValue* class 12-15
- DtCommand* class 12-5
- DtCommandDefinition* class 12-5
- DtCommandSP* class 12-14
- DtDisLogger* class 12-7
- DtHlaLogger* class 12-7
- DtLgrAppGuiInterface* class 12-18
- DtLgrBaseSimManager* class 12-6
- DtLgrCommandHandler* class 12-4
- DtLgrDisSimulationManager* class 12-6
- DtLgrFullInterface* class 12-5
- DtLgrHlaSimulationManager* class 12-6
- DtLgrInterface* class 12-3
- DtLgrPlaybackCommand Factory* class 12-5
- DtLgrPlaybackDriver* class 12-4
- DtLgrPlaybackInterface* class 12-5
- DtLgrProtocolRegistration* class 12-6
- DtLgrSimInput* class 12-6
- DtLgrSimOutput* class 12-6
- DtLgrSystemDriver* class 12-4
- DtLogger* class 12-3, 12-4, 12-6, 12-7, 12-10, 12-11, 12-12
- DtLoggerApplication* class 12-10, 12-11
- DtMessageInput* class 12-6
- DtMessageOutput* class 12-6
- DtMessageState* class 12-6
- DtMutex* class 12-16
- DtObserveConnection* class 12-17
- DtObserveEvent* class 12-17
- DtObserver* class 12-17
- DtProtocolRegistration* class 12-18
- DtRemoteControlInterface* class 13-2
- DtRemoteControlInterface* class 13-3
- DtRemoteControlPlaybackInterface* class 13-2
- DtRemoteControlSystemInterface* class 13-4
- DtResponse* class 12-5, 12-14
- DtSimInput* class 12-6
- DtString* class 12-15
- DtThreadSafe* class 12-16
- DtValue* class 12-15
- DtValueSignal* class 12-16

E

- e command-line option 4-18, 5-24
- editing
 - annotation 7-5
 - connection 4-4
 - file 9-2
 - lgrConfig.xml* 5-18
 - point annotation 7-6
 - range annotation 7-6
 - undoing changes 9-2

Editor domain, commands 11-17
 embarkdemo 1-4
 embedding
 Logger in application 12-7, 12-10
 plug-ins in application 12-7
 empty space, trimming 9-5
 enableDdm parameter 5-14
 EnableLoggerAppPlugin() function 12-11
 EnableLoggerPlugin() function 12-11
 enabling
 attribute scaling 11-11
 backup 11-7
 entity tracking 11-6
 high speed playback 6-16
 looping playback 11-7
 Pause on Record As 6-8
 Play On Load 6-13
 time management 11-14
 end
 jumping to range annotation 7-7
 moving to 6-20
 end time
 setting dynamically 7-4
 setting for instant range annotation 7-3
 ending, recording 6-10
 end-of-file, setting 6-15
 entity
 common types 8-3
 filtering, entity tracking 5-5
 filtering by type 11-15
 maintenance 3-7
 entity ID, filtering by 8-5
 entity tracking 5-5
 enabling or disabling 11-6
 entity type
 filtering 11-15
 filtering by 8-2
 removing from filtering list 8-2
 environment variable
 MAKLMGRD_LICENSE_FILE 2-6
 RTL_CONFIG 2-7
 example
 API 12-7
 building 12-21
 lgrControl 1-5
 Logger control application 13-8
 loggerApp 1-5
 plug-in 1-5, 12-10
 programs 1-5
 sampleFilter 1-5

example (continued)
 stateObserver 1-5
 Excel 10-14
 importing Access data to 10-18
 exercise
 connecting to 4-4
 connection 4-3
 configuring DIS 11-16
 disconnecting from 6-6
 ID 3-3, 4-18, 11-16
 playback 3-3
 recording 6-7
 exercise connection 13-3
 Exit command 11-6
 expanding, playback segment 6-15
 export, log file 10-7
 ExportFile command 11-17
 exporting
 Access to 10-7
 data, naming convention 6-9
 data to database 10-3
 file 10-2, 11-17
 ExportSection command 11-18

F

-F command-line option 4-8, 4-10, 5-24
 -f command-line option 4-11, 5-23
 factory 12-5
 FDD file 2-7
 features 1-2
 FED file 1-6, 2-7, 4-9
 changing 3-8
 specifying 4-8, 4-12, 5-24
 federate, specifying Logger federate name 4-7
 federate type, specifying 4-7, 5-23
 federation execution 1-6
 name 4-8, 5-24
 file
 annotating 7-2
 appending 9-4, 11-17, 11-18
 backup 6-17, 11-7
 clearing, packets 11-17
 clearing segment 9-2
 closing 6-12, 11-7
 configuration 5-18
 deleting
 packet 11-17
 segment 9-2
 editing 9-2

file (continued)
 exporting 10-2, 11-17
 to database 10-3
 FED 2-7, 4-12
 jumping to beginning 6-20
 jumping to end 6-20
lgrConfig.xml 5-2
 loading 6-11
 merging 9-4, 11-18
 naming convention 13-8
 offsetting 9-5
 opening 11-7
 pausing 6-13
 playing 6-13
 recording to multiple 6-8
 repairing 6-12
 reverting to saved version 9-2, 11-7
 RID 2-7
 saving 6-17, 11-7
 segment to new file 6-17
 to new name 6-17
 trimming 11-18
 empty space 9-5
 upgrading format 6-18
VR-Link.fed 4-8
 filename
 constructing 13-8
 logger.log 5-17
 filename command-line option 5-23
 filter
 entity ID 8-5
 HLA object name 8-5
 message 1-3
 Filter By Class dialog box 8-6
 Filter By Entity Type dialog box 8-2
 Filter By PDU Kind dialog box 8-3
 filtering
 by entity type 11-15
 by PDU kind 11-15
 classes (HLA) 8-6
 configuring 11-11
 configuring common entity types 8-3
 custom PDUs 8-4
 entities 5-5
 entity type 8-2, 11-15
 input and output 8-2
 PDU kind 8-3
 PDUs 11-15
 VR-Forces messages 8-7
 FLEXlm 2-4

FOM 2-7
 database export 10-7
 flexibility 3-8
 primary 4-9
 supported 1-6
 FOM Mapper 1-6, 4-11, 5-23
 initialization data 5-23
 specifying 4-12
 specifying at startup 4-10
 FOM module, specifying 4-9, 5-24
 format, upgrading 6-18
 frequency, packets for indexing 5-2
 function
 addCallback() 13-10
 addCallbackConnectionAdded() 13-11
 addCallbackConnectionRemoved() 13-11
 addCallbackConnectionTimedOut() 13-11
 addCommandHandler() 12-4
 create() 12-5, 12-15
 decode() 12-5
 EnableLoggerAppPlugin() 12-11
 EnableLoggerPlugin() 12-11
 LoadPlugin() 12-11
 receivedTotalState() 13-4
 removeCallback() 13-10
 removeCallbackConnectionAdded() 13-11
 removeCallbackConnectionRemoved() 13-11
 removeCallbackConnectionTimedOut() 13-11
 requestStateUpdate() 13-4
 setAcceptAnyId() 13-10
 setConnection() 13-10
 setConnectionTimeout() 13-3, 13-10
 setCreator() 12-15
 setID() 13-10
 setNumberOfLoggers() 13-10
 tick() 12-4, 12-8
 UnloadPlugin() 12-11
 update() 12-5, 12-8

G

GUI, localizing 2-8

H

-h command-line option 5-23
 handle, start and end time 6-14
 header file
 lgrBaseSimManager.h 12-6
 lgrCommand.h 12-5

header file (continued)

- lgrDisSimManager.h* 12-6
- lgrDriver.h* 12-4
- lgrHlaSimManager.h* 12-6
- lgrPlaybackCommandFactory.h* 12-5
- lgrPlaybackDriver.h* 12-4
- lgrPlaybackInterface.h* 12-5
- lgrProtocolRegistration.h* 12-6
- lgrRuntimeCommandFactory.h* 12-5
- lgrSimInput.h* 12-6
- lgrSimOutput.h* 12-6
- lgrSystemDriver.h* 12-4
- logger.h* 12-14
- messageInput.h* 12-6
- messageOutput.h* 12-6
- messageState.h* 12-6
- observeConnection.h* 12-17
- observeEvent.h* 12-17
- observe.h* 12-17
- remoteControlDomainInterface.h* 13-2
- remoteControlInterface.h* 13-2
- remoteControlPlaybackInterface.h* 13-2
- remoteControlRecordInterface.h* 13-2
- runtimeValue.h* 12-15
- stateObController.h* 12-5
- stateObModel.h* 12-4
- virtualConstructor.h* 12-15
- heartbeat 11-16
 - defined 3-6
 - HLA 5-7, 11-14
 - setting 4-22
- Help command 11-6
- hiding
 - plug-in view 6-5
 - toolbar 6-5
- high speed, playback 6-16
- histogram 6-4, 6-21
- HLA 1-6
 - 1516
 - playback 3-4
 - advisory messages 4-11, 11-14
 - commands 11-11
 - configuring connection 11-12
 - configuring for 4-6
 - configuring time management 4-13
 - database schema 10-7
 - DDM 5-10
 - Evolved, playback 3-4
 - exporting to database 10-7
 - FOM flexibility 3-8

HLA (continued)

- heartbeat 5-7, 11-14
- object name, filtering by 8-5
- passive subscription 4-16, 11-15
- specifications supported 1-6
- subscription control list 4-14
- subscription list 11-15
- support for 3-8
- time management 5-17
- timeout 5-7, 11-15
- HLA Evolved
 - exporting to database 10-7
 - federate type 4-7
 - FOM 4-8
 - FOM module 4-9
 - MIM 4-9
- hostname, license server 2-5

I

ID

- entity, filtering by 8-5
- remote control 13-3, 13-8
- IEEE 1516 1-6
- ignoring, advisory messages 4-11
- IISEC_VPMM_Demo_Final-DIS 1-5
- immediate
 - playback 6-13
 - recording 6-7, 6-8
- importing, annotation file 7-5
- incomplete file, repairing 6-12
- indexing, configuring 5-2
- informational message 5-17
- input
 - filtering 8-2
 - stage 12-6
- inserting, file 11-18
- installing, RTI 2-7
- instant annotation 7-3
- interaction
 - passive subscription 4-16
 - subscribing to 4-14
- interface 1-3
 - class 12-5
 - domain 13-2, 13-6
 - remote control 13-2
 - signal slot 12-16
 - text 11-3
- interim PDU 3-6, 3-7
- IP address 5-24

J

jumping

- range annotation end 7-7
- range annotation start 7-6
- to beginning of file 6-20
- to end of file 6-20
- to specific time 6-20

K

keyboard, setting annotation times from 7-4

L

-l command-line option 5-17, 5-23

lgrAnnotations library 12-18

lgrApp library 12-18

lgrBaseSimManager.h header file 12-6

lgrCommand.h header file 12-5

lgrConfig.xml 6-16, 13-3, 13-9

lgrConfig.xml 11-3

- configuration file 5-2

- syntax and editing 5-18

lgrControl, example 1-5

lgrCore library 12-18

lgrDisSimManager.h header file 12-6

lgrDriver.h header file 12-4

lgrDump 1-5

lgrGui library 12-18

lgrHlaSimManager.h header file 12-6

lgrPlaybackCommandFactory.h header file 12-5

lgrPlaybackDriver.h header file 12-4

lgrPlaybackInterface.h header file 12-5

lgrProtocolRegistration.h header file 12-6

lgrRuntimeCommandFactory.h header file 12-5

lgrSim library 12-18

lgrSimGui library 12-18

lgrSimInput.h header file 12-6

lgrSimOutput.h header file 12-6

lgrSystemDriver.h header file 12-4

lgrUtil library 12-18

library

- API 12-18

- appEnv 12-18

- lgrAnnotations 12-18

- lgrApp 12-18

- lgrCore 12-18

- lgrGui 12-18

- lgrSim 12-18

library (continued)

- lgrSimGui 12-18

- lgrUtil 12-18

- Logger API 12-19

- plug-in 12-7

- plug-ins 12-11

License Manager 2-4

license server, hostname 2-5

License Setup dialog box 2-5

linking

- API code 12-19

- application 12-19

Linux

- compiling and linking API 12-19

- installing on 2-2

loading

- annotation file automatically 6-13

- configuration file 5-23

- file 6-11, 11-7

- plug-ins 12-12

- subscription list 4-16, 11-11

LoadPlugin command 11-6

LoadPlugin() function 12-11

LoadSubscriptionListFile command 4-16, 5-10,
11-11

localization 1-3

localizing, GUI 2-8

log file 5-17, 5-23

- sql_export.txt* 10-7

logFileName parameter 5-17

Logger

- embedding in application 12-7, 12-10

- recording

 - samples 1-4

 - viewing with Stealth 1-4

- window 6-4

Logger API 12-3

- architecture 12-4

- coding conventions 12-13

- compiling and linking 12-19

- compiling and linking applications 12-19

- drivers 12-4

- embedding Logger 12-7, 12-10

- examples 12-7

- libraries 12-18

- runtime values 12-15

Logger command 5-2

Logger Control

- configuring 11-10

- message, configuring response to 11-10

Logger control application, example 13-8
 Logger State Observer 12-10
 loggerApp, example 1-5
logger.h header file 12-14
logger.log 5-23
 loop, enabling or disabling 11-7
 looping, playback 6-15

M

-m command-line option 5-17, 5-23
 macro, file naming convention 6-9
 MAK RTI, configuring for name reservation 3-4
 MÄK RTI, configuring 2-7
 maklanddemo 1-4
 MAKLMGRD_LICENSE_FILE, environment variable 2-6
 Max Read Time 5-9
 MergeInFile command 11-18
 merging, files 9-4, 11-18
 message 11-10
 acknowledge 13-5
 advisory 4-11
 filtering 1-3
 filtering VR-Forces 8-7
 informational 5-17
 log file 5-23
 maximum to read per tick 5-9
 pipeline 12-6
 VR-Link 5-17
messageInput.h header file 12-6
messageOutput.h header file 12-6
messageState.h header file 12-6
 method. See function
 Microsoft Access, configuring data source for 10-8
 Microsoft Excel 10-14, 10-18
 MIM, specifying 4-9, 5-24
 Min Read Time 5-9
 MOM and Initialization Module, specifying 4-9
 moving
 to beginning of file 6-20
 to beginning or end of playback range 6-20
 to end of file 6-20
 to specific time 6-20
 multicast address 4-24, 5-25, 11-16
 multicast time-to-live 4-21, 11-16
 specifying 5-24
 multiple
 files, recording to 6-8
 Logger, using 13-3

multiple (continued)
 network card 4-21, 5-24
 MySQL 10-14
 configuring data source for 10-11

N

-N command-line option 4-7, 5-24
 -n command-line option 5-17, 5-23
 name
 file, convention for 13-8
 Logger federate 4-7
 name reservation 3-4
 configuring MAK RTI 3-4
 namespace 12-13
 naming, partial files 6-9
 naming convention, for database tables and rows 10-3
 network
 connection 4-3
 read time, configuring 5-9
 requirements for Logger A-2
 network card, multiple 4-21, 5-24
 network device 11-16
 network interface address 4-20
 specifying 4-21
 new file 11-8
 NewFile command 11-8
 notification level, setting 5-17
 Num PDUs Per Check 5-9

O

object
 displaying statistics 6-22
 name 11-14
 passive subscription 4-16
 subscribing to 4-14
 object name, filtering by 8-5
observeConnection.h header file 12-17
observeEvent.h header file 12-17
observe.h header file 12-17
 observer pattern 12-17
 ODBC, configuring data source 10-5
 OffsetTime command 11-18
 offsetting
 file 9-5
 packets 11-18
 OMT file 10-7
 OpenFile command 11-7

- opening
 - annotation file 7-5
 - file 6-11, 11-7
 - incomplete file 6-12
- output
 - filtering 8-2
 - stage 12-6

P

- P command-line option 4-17, 5-24
- packet
 - clearing 9-2
 - count 6-21
 - deleting 9-2
 - from file 11-17
 - dumping to text 10-2
 - offsetting 9-5
 - time 11-18
 - preventing dropped 4-19
 - removing
 - empty 9-5
 - from file 11-17
 - skipping 6-16
- parameter
 - `allowPlaybackWithoutRegions` 5-15, 11-13
 - `commonEntityTypes` 8-3
 - `customPdus` 8-4
 - `enableDdm` 5-14
 - `logFileName` 5-17
 - `playbackDynamicAssociation` 5-14
 - `playbackDynamicExtents` 5-14
 - `processRemoteControl` 13-3
 - `recordConveyedRegions` 5-15
 - `requestUpdates` 5-15
 - `sendResponses` 13-3
 - `showConsole` 11-3
 - `subscribeMode` 5-10
 - `SubscribeRegionsAreGlobal` 5-10
 - `SubscribeRegionsAreSpecified` 5-10, 5-15, 11-14
 - `subscribeUnspecifiedAttributes` 5-12, 5-15, 11-14
 - `useAsyncIO` 4-19
- partial file, naming convention 6-9
- passive subscription 4-16
 - configuring 11-15
- path, specifying for RTI 2-7
- Pause command 11-7
- Pause on Record As 6-7, 6-8
- PauseRecording command 11-8

- pausing
 - playback 6-13, 11-7
 - recording 6-10, 11-8
- PDU
 - filtering 11-15
 - filtering by 8-3, 11-15
 - filtering custom 8-4
 - maximum to read per tick 5-9
 - supported for database export 10-6
- performance, configuring 5-9
- pipeline 12-6
- Play button 6-13
- Play command 11-7
- Play On Load 6-11, 6-13
- playback 3-3, 6-11
 - automatic 6-13
 - bounding to annotation 7-9
 - DDM 5-13
 - described 3-3
 - exercise ID 3-3
 - fast 5-4
 - faster, configuring 11-6
 - high speed 6-16
 - HLA 1516 3-4
 - HLA Evolved 3-4
 - loading file 6-11
 - looping 6-15, 11-7
 - mode 3-3
 - moving to beginning 6-20
 - moving to end 6-20
 - pausing 6-13, 11-7
 - range, moving to beginning or end 6-20
 - segment 6-13, 6-14
 - expanding 6-15
 - setting speed 6-15, 11-7
 - starting 11-7
 - stopping 6-15, 11-8
 - time-managed 5-17
 - unpausing 11-8
- `playbackDynamicAssociation` parameter 5-14
- `playbackDynamicExtents` parameter 5-14
- `playbackMode` 5-14, 11-13
- `PlaybackRegionNotEnabled` 5-14
- `PlaybackRegionsUseConveyedExtents` 5-14
- `PlaybackRegionsUseData` 5-14, 11-13
- Player domain, commands 11-7
- playFile example
 - example, playFile 12-8
- playing
 - file 6-13

- playing (continued)
 - part of a file 6-13, 6-14
- PlayPause command 11-7
- plug-in 6-4
 - API 12-10
 - docking and undocking views 6-5
 - embedding in application 12-7
 - example 1-5, 12-10
 - hiding or displaying 6-5
 - loading 12-12
 - object statistics 6-22
- plug-in library 12-11
- point annotation
 - adding instantly 7-3
 - converting to range 7-6
 - editing 7-6
- Point Annotation dialog box 7-5
- pointer 12-14
- port 11-16
 - specifying UDP 4-17
- preferences, setting 5-2
- Preferences dialog box 5-3
- PrefixFileToStart command 11-18
- prefixing, file 11-18
- processing, responses 13-7
- processRemoteControl** parameter 13-3
- property
 - annotation 7-5
 - connection 4-3
- protocol
 - supported 1-6
 - version, DIS 5-24

Q

- Qt Linguist 2-8

R

- r command-line option 5-23
- range annotation
 - adding instantly 7-3
 - bounding playback to 7-9
 - converting to point 7-6
 - editing 7-6
 - jumping to end 7-7
 - jumping to start 7-6
 - setting end time 7-4
- Range Annotation dialog box 7-5

- receive buffer 4-20, 11-16
 - size, specifying 5-24
- receivedTotalState() function 13-4
- receiving application responses 12-5
- record
 - mode 3-3
 - remote connection 13-4
- Record As, configuring 11-9
- Record command 11-8
- recordConveyedRegions** parameter 5-15
- Recorder domain, commands 11-8
- recording 3-3
 - automatically 6-8
 - canceling 11-8
 - closing 11-8
 - configuring 11-9
 - described 3-3
 - exercise 6-7
 - ID 3-3
 - immediately 6-7
 - multiple files 6-8
 - pausing 6-10, 11-8
 - repairing 6-12
 - resuming 6-10
 - saving 6-17
 - starting 6-7
 - stopping 6-10
 - time-managed 5-17
- region
 - conveyed 5-13
 - subscription 5-10
- region descriptor 5-10
- region set 5-10
- remote connection, records 13-4
- remote control
 - ID 13-3, 13-8
 - interface 13-2
- remote control API 13-2
- remote control ID, specifying 5-25, 11-10
- remote control interface
 - configuring 13-10
 - configuring Logger for 13-9
 - connecting to Logger 13-3
 - Logger state 13-4
 - processing responses 13-7
 - sending commands 13-5
- remote Logger, sending commands to 13-5
- remoteControlDomainInterface.h* header file 13-2
- remoteControlInterface.h* header file 13-2
- remoteControlPlaybackInterface.h* header file 13-2

remoteControlRecordInterface.h header file 13-2
 removeCallback() function 13-10
 removeCallbackConnectionAdded() function 13-11
 removeCallbackConnectionRemoved() function 13-11
 removeCallbackConnectionTimedOut() function 13-11
 removing
 dead space 11-18
 empty packets 9-5
 packets from file 11-17
 repairing, incomplete file 6-12
 requestStateUpdate() function 13-4
requestUpdates parameter 5-15
 resetting, annotation time point 7-4
 response 12-5
 processing 13-7
 Restart command 11-7
 resuming, recording 6-10
 RevertFile command 11-7
 reverting
 file 11-7
 to saved file 9-2
 RID file 2-7
rid.mtl file 2-7
 row, naming conventions 10-3
 RPR FOM 3-8, 4-10
 revision, specifying at startup 4-10
 RTI 2-7
 definition of 2-7
 specifications supported 1-6
 RTI_CONFIG environment variable 2-7
 runtime value 12-15
runtimeValue.h header file 12-15

S

-S command-line option 4-24, 5-25
 -s command-line option 4-18, 5-25
 sample recording 1-4
 sampleFilter, example 1-5
 Save 6-17
 Save As 6-17
 SaveFile command 11-7
 SaveFileAs command 11-7
 SaveSubscriptionListFile command 4-15, 11-11
 saving
 annotation file 7-4

saving (continued)
 file 11-7
 to new name 6-17
 files 6-17
 segment as new file 6-17
 subscription list 4-15, 11-11
 scaling, time 3-7
 schema, database 10-3
 segment
 playing 6-13, 6-14
 saving as new file 6-17
 setting playback to annotation 7-9
 selecting, annotation 7-5
 send buffer 4-20, 11-16
 size, specifying 5-25
 sending, commands 12-5
sendResponses parameter 13-3
 set, region 5-10
 Set Start and End Time dialog box 6-14
 Set Time 3-7, 6-20
 Set Zoom dialog box 6-19
 setAcceptAnyId() function 13-10
 SetBackupOnSave command 6-17, 11-7
 SetCheckpointParameters command 5-3, 11-8
 SetCommonEntityTypes command
 command, SetCommonEntityTypes 11-15
 setConnection() function 13-10
 SetConnectionParameters command 4-6, 4-17, 11-10, 11-12, 11-16
 setConnectionTimeout() function 13-3, 13-10
 setCreator() function 12-15
 SetCurrentTime command 11-7
 SetDdmParameters, command 11-13
 SetDdmParameters command 5-10, 5-11, 5-14
 SetDisableTimeManagementTest command 11-14
 SetDrainInput, command 11-14, 11-16
 SetDynamicNamePrefix command 3-3, 11-14
 SetEntityTracking command 5-4, 5-5, 11-6
 SetFastJumpsWithoutDiff command 5-4, 5-5, 11-6
 SetHeartbeatIntervals, command 5-4, 5-7, 11-14
 SetHeartbeatRate command 4-22, 11-16
 setID() function 13-10
 SetIgnoreAdvisories command 4-11, 11-14
 SetLoopAtEnd command 11-7
 SetNotifyLevel command 5-17, 11-6
 setNumberOfLoggers() function 13-10
 SetPassiveSubscription command 4-16, 11-15
 SetPauseOnRecordAs command 11-9

- SetPlaybackReadTimeLimit command 5-4, 5-6, 11-6
- SetPlaybackSpeed command 11-7
- SetPlayOnLoad command 11-8
- SetRemoteControlId command 11-10
- SetRemoteControlSettings, command 13-3
- SetRemoteControlSettings command 11-10
- SetScaleAttributes command 11-11
- SetStartStopTimes command 11-8
- SetStopCondition command 11-9
- SetStopDuration command 11-9
- SetStopSize command 11-9
- SetSubscriptionLists command 4-15, 5-22, 11-15
- SetTempDir command 11-6
- SetTimeManagement command 4-13, 11-15
- SetTimeoutIntervals, command 5-4, 5-7, 11-15
- SetTimeoutRate command 4-23, 11-16
- setting
 - checkpoint policy 11-8
 - cursor to time 11-7
 - DIS heartbeat 4-22
 - DIS timeout 4-23
 - end-of-file action 6-15
 - notification level 5-17
 - playback segment to annotation 7-9
 - playback speed 6-15, 11-7
 - preferences 5-2
- SetVRFFiltering command 11-11
- shared pointer 12-14
- showConsole** parameter 11-3
- signal 12-16, 12-17
 - slot 12-16
- simulation
 - manager 12-6
 - recording 6-7
 - standard, supported 1-6
- Simulation domain, commands 11-10
- SISO 1516 API 1-6
- site ID 4-18, 5-25, 11-10, 11-16
 - specifying 5-25
- size, indexing 5-2
- skip command, entity maintenance 3-7
- Skip Time 3-7
- skipping, packets 6-16
- skippingPlaybackThreshold 6-16
- slider
 - start and end time 6-14
 - zoom 6-14
- slot 12-17
 - signal 12-16
- slow-speed playback 3-6
- smart pointer 12-14
- specialization, template 12-13
- specifying
 - application number 5-24
 - configuration file 5-23
 - FED file 4-8, 4-12, 5-24
 - federate type 4-7, 5-23
 - federation execution name 4-8
 - FOM Mapper 4-12
 - FOM modules 4-9, 5-24
 - MIM 4-9, 5-24
 - MOM and Initialization Module 4-9
 - multicast time-to-live 5-24
 - network interface address 4-21
 - receive buffer size 5-24
 - remote control ID 5-25, 11-10
 - send buffer size 5-25
 - site ID 5-25
 - subscription list 11-15
 - subscription region 5-10
 - time management 5-17
 - UDP port 4-17
- speed
 - control 6-4
 - setting for playback 6-15
 - setting playback 11-7
- spreadsheet, exporting data to 10-14
- SQL, exporting data to 10-3
- sql_export.txt* log file 10-7
- stage, input, output, state 12-6
- start
 - jumping to range annotation 7-6
 - moving to 6-20
- start time, setting dynamically 7-4
- starting
 - Logger 6-3
 - playback 11-7
 - recording 6-7
- startup
 - playback mode 3-3
 - record mode 3-3
 - running commands 5-23
 - UNIX 6-3
 - Windows 6-3
- state
 - Logger, inspecting 13-4
- state stage 12-6
- stateObController.h* header file 12-5
- stateObModel.h* header file 12-4

- stateObserver, example 1-5
- status, field 6-7
- stopping
 - playback 6-15, 11-8
 - recording 6-10
- StopPlayback command 11-8
- StopRecording command 11-9
- string list 4-15
 - adding to configuration file 5-22
- subnet mask 4-20, 11-16
- subscribeMode 5-14
- subscribeMode** parameter 5-10
- SubscribeRegionNotEnabled 5-14, 11-13
- SubscribeRegionsAreGlobal 5-14
- SubscribeRegionsAreGlobal** parameter 5-10
- SubscribeRegionsAreSpecified 5-14
- SubscribeRegionsAreSpecified** parameter 5-10, 5-15, 11-14
- subscribeUnspecifiedAttributes** parameter 5-12, 5-15, 11-14
- subscribing, passively 4-16
- subscription, region 5-10
- subscription control list, creating 4-14
- subscription list
 - DDM, setting programmatically 5-12
 - loading 4-16, 11-11
 - saving 4-15, 11-11
 - specifying 11-15
- subscription region, specifying 5-10
- syntax, *lgrConfig.xml* 5-18
- System domain, commands 11-6

T

- table, naming conventions 10-3
- tape
 - annotating 7-2
 - editing 9-2
 - loading 6-11
 - playing 6-13
 - starting 6-7
- TCP 5-24
- template 12-13
 - specialization 12-13
- temporary file, recording to 6-7
- text, dumping file to 10-2
- text interface 11-3
- thread safety
 - mutex, use of 12-16
- tick() function 12-4, 12-8

- time
 - command, entity maintenance 3-7
 - indexing 5-2
 - jumping to 6-20
 - navigating during playback 6-20
 - navigation 6-20
 - offsetting packet 11-18
 - setting 6-21
 - setting cursor to 11-7
 - sliders 6-14
- time jump, updating entities after 3-7
- time management
 - configuring 11-15
 - in GUI 4-13
 - enabling or disabling 11-14
 - specifying use of 5-17
- time scale
 - described 3-4, 3-5, 3-6
 - during recording 6-7
 - effects 3-6
- time-constrained, setting 4-13
- timeline
 - cursor 6-21
 - moving 6-20
 - zooming 6-18
 - out 6-19
 - to annotation 7-8
- timeout 11-16
 - defined 3-6
 - HLA 5-7, 11-15
 - setting 4-23
- timepoint, setting 6-20
- time-regulating, setting 4-13
- timestamp, offsetting 9-5
- toolbar 6-4
 - docking and undocking 6-5
 - hiding or displaying 6-5
- translating, GUI text 2-8
- translation 1-3
- trimming, file 9-5, 11-18
- TrimStart command 11-18

U

- UDP port 5-24
 - specifying 4-17, 5-24
- undo, edits 9-2
- undocking, toolbars and plug-in views 6-5
- UNIX
 - starting, Logger 6-3

UnloadPlugin() function 12-11
Unpause command 11-8
unpausing 11-8
Update Annotation 7-4
update() function 12-5, 12-8
updating, annotation 7-4
Upgrade File Format 6-18
upgrading, file format 6-18
`useAsyncIO` parameter 4-19
using, multiple Logger 13-3

V

-v command-line option 5-23
variable
 adding to *lgrConfig.xml* 5-21
 syntax in *lgrConfig.xml* 5-20
VCR controls 6-4
velocity, scaling 3-7
version, displaying 5-23
View Annotations window 7-2
view control PDUs 1-5
View menu 6-5
viewing
 connection properties 4-3
 object statistics 6-22
virtual constructor 12-14
virtualConstructor.h header file 12-15
VR-Forces message 8-7
VR-Link, messages 5-17
VR-Link.fed file 1-6, 4-8
VR-TheWorld_FirstExperienceDIS 1-5
VR-Village-CarBombDemo 1-4

W

window, Logger 6-4
Windows
 compiling and linking API 12-19
 installing on 2-2
 starting Logger 6-3

X-Y-Z

-x command-line option 4-8, 5-24
XML, RTI configuration file 2-7
XML file 10-7
zoom area 6-14
zoom bar 6-19

zoom slider 6-18
zooming
 timeline 6-18
 timeline out 6-19
 timeline to annotation 7-8

MÄK Data Logger Quick Reference Card

Logger commands

Command	Description
System domain commands	
DeleteFile <i>filename</i> (string)	Deletes the specified file.
Exit	Shuts down the Logger.
Help	Displays a list of available commands.
LoadPlugin <i>pluginName</i> (string)	Loads the specified plug-in.
SetEntityTracking	Allows entity tracking to be processed.
SetFastJumpsWithoutDiff	Uses checkpoints to establish the relative state of objects, but the packets between the checkpoint and jump time are not processed
SetNotifyLevel <i>vlLevel</i> (int) <i>lgrLevel</i> (int)	Sets the notification level of VR-Link and the Logger.
SetPlaybackReadTimeLimit	Limits how long the Logger can spend reading packets during playback.
SetTempDir <i>directory</i> (string)	Sets the directory in which the Logger stores temporary Logger files.
Player domain commands	
ClosePlaybackFile	Closes the current playback file.
OpenFile <i>filename</i> (string) <i>constructFilename</i> (boolean)	Opens the specified file for playback.
Pause	Pauses playback.
Play	Starts playback. If the Logger is already playing this command has no effect.
PlayPause	Toggles between the paused and play state.
Restart	Starts playing from the beginning of the file.
RevertFile	Reverts a previously saved, but currently unsaved file to the previously saved state.
SaveFile	Saves a previously saved, but currently unsaved file to its current file name.
SaveFileAs <i>filename</i> (string)	Saves the current file to new file.
SetBackupOnSave <i>backup</i> (bool)	Enables or disables automatic backup of previously saved files when saving edited files.
SetCurrentTime <i>time</i> (time)	Moves the timeline cursor to the specified time.
SetLoopAtEnd <i>value</i> (bool)	Enables or disables looping to start at the end of playback.
SetPlaybackSpeed <i>speed</i> (double) <i>skippingPlayback</i> (bool)	Sets the playback speed of the Logger.
SetPlayOnLoad <i>value</i> (bool)	Enables starting playback immediately after opening a file.
SetStartStopTimes <i>startTime</i> (time) <i>stopTime</i> (time)	Sets the start and stop time of the Logger.
StopPlayback	Stops playback. This command has no effect if the Logger is not in the playback state.
Unpause	Resumes playback. This command has no effect if the Logger is not paused.
Recorder domain commands	
CancelRecording	Cancels the current recording.
CloseRecordFile	Closes the file currently recording.
NewFile <i>filename</i> (string) <i>overwrite</i> (bool)	Opens a new file for recording.
PauseRecording	Pauses the recording process.
Record	Starts recording to a new file.

Logger commands

Command	Description
SetCheckpointParameters <i>policy</i> (int) { <i>size</i> (int) <i>count</i> (int) <i>time</i> (time)}	Sets the checkpoint policy parameters on when checkpoints are generated. The Policy is one of the following: ♦ 0 — count, in number of packets ♦ 1 — size of packets, in bytes ♦ 2 — time, in seconds between checkpoints.
SetPauseOnRecordAs <i>record</i> (bool)	Determines how the Logger behaves when given the Record As command.
SetStopCondition <i>condition</i> (int)	Specifies the stop condition for recording as follows: ♦ 0 — no condition ♦ 1 — duration (enables SetStopDuration) ♦ 2 — file size (enables SetStopSize).
SetStopDuration <i>duration</i> (time)	Stops recording at the specified time if SetStopCondition is set to 1.
SetStopSize <i>filesize</i> (int64)	Stops recording when the file reaches <i>filesize</i> size if SetStopCondition is set to 2.
StopRecording <i>closeAndOpenForPlayback</i> (bool)	Stops recording and ends.

Simulation domain commands (protocol-independent commands)

Connect	Connects to the network simulation using the previously set parameters.
Disconnect	Disconnects from the network simulation.
SetRemoteControlSettings <i>process</i> (bool) <i>process</i> (bool) <i>recordskippingPlayback</i> (bool) <i>allowDelete</i> (bool) <i>allowAll</i> (bool) <i>sendResponses</i> (bool) <i>crippleGui</i> (bool) <i>connectSendInterval</i> (double) <i>heartbeatSendInterval</i> (double)	Specifies how the Logger responds to Logger Control messages.
SetScaleAttributes <i>scale</i> (bool)	Enables or disables attribute scaling due to changes in playback speed.
SetVRFFiltering <i>filter</i> (bool) <i>loId</i> (int) <i>hiId</i> (int)	Configures VR-Forces message filtering.

HLA-specific commands

LoadSubscriptionListFile <i>filename</i> (string)	Loads a subscription list file.
SaveSubscriptionListFile <i>filename</i> (string)	Saves the subscription settings to an XML file with the specified name.
SetConnectionParameters <i>federateName</i> (string) <i>federateType</i> (string) <i>federationName</i> (string) <i>fedFile</i> (string) <i>fomModules</i> (string) <i>MIM</i> (string) <i>ignoreAdvisories</i> (bool) <i>fomVersion</i> (double) <i>fomRevision</i> (int) <i>fomMapper</i> (string) <i>fomMapperData</i> (base64) <i>useEmptyFomMapper</i> (bool)	The <i>federateType</i> , <i>fomModules</i> , and <i>MIM</i> parameters apply only to HLA Evolved. Do not include them for HLA 1.3 or 1516-2000.
	Specifies HLA connection parameters indicated by the argument names.
SetDdmParameters <i>enableDdm</i> <i>playbackMode</i> <i>subscribeMode</i> <i>playbackDynamicAssociation</i> <i>playbackDynamicExtents</i> <i>allowPlaybackWithoutRegions</i> <i>recordConveyedRegions</i> <i>requestUpdates</i> <i>subscribeUnspecifiedAttributes</i>	Specifies DDM parameters.
SetDisableTimeManagementTest <i>disableTesting</i> (bool)	Disables testing for time management if that test causes the Logger to behave badly.
SetDrainInput(double <i>maxTime</i> , double <i>minTime</i>)	Specifies the minimum and maximum amount of time, in seconds, to spend reading messages from the exercise connection (that is, the network) each tick.
SetDynamicNamePrefix <i>prefix</i> (string)	Specifies the prefix that the Logger attaches to the beginning of object names that begin with HLA_ when it is recording exercises.
SetHeartbeatIntervals "className(string):interval(float):inheritable(bool) ..."	Specifies a heartbeat interval for playback. You can specify multiple classes.

MÄK Data Logger Quick Reference Card

Logger commands

Command	Description
<code>SetIgnoreAdvisories mode(bool)</code>	Specifies the Logger's response to HLA advisory messages. Default: false.
<code>SetPassiveSubscription mode(bool)</code>	Configures passive subscription for HLA. Default: false.
<code>SetSubscriptionLists objectClasses(stringList) interactionClasses(stringList)</code>	Sets the class subscription list.
<code>SetTimeManagement timeConstrainedRecording(bool) timeConstrainedPlayback(bool) timeRegulatingRecording(bool) timeRegulatingPlayback(bool) lookahead(double)</code>	Configures time management options.
<code>SetTimeoutIntervals "className(string):interval(float):inheritable(bool) ..."</code>	Specifies an object timeout for recording. You can specify multiple classes.
DIS-specific commands	
<code>AddCustomPDU pduName(string) pduKind(string) entityIdOffset(string) entityTypeOffset(string)</code>	Specifies a PDU to add to the Filter by PDU Kind dialog box.
<code>SetCommonEntityTypes entityType1 ... entityTypeN(string)</code>	Specifies a list of entity types to list in the Filter by Entity Type dialog box.
<code>SetConnectionParameters port(int) exerciseId(int) siteId(int) appNumber(int) useAsynchIO(bool) sendBufferSize(int) recvBufferSize(int) multicastTtl(int) destAddr(string) subnetMask(string) deviceAddr(string) mcastAddresses(stringList)</code>	Specifies parameters for the DIS exercise connection.
<code>SetDrainInput(double maxTime, int numPdusToRead)</code>	Specifies the maximum amount of time, in seconds, to spend reading messages from the exercise connection (that is, the network) and the number of PDUs to read between each check of the maximum time.
<code>SetHeartbeatRate rate(double)</code>	Specifies the DIS heartbeat rate.
<code>SetTimeoutRate rate(double)</code>	Specifies the DIS timeout rate.
Editor domain commands	
<code>AppendFileToEnd filename(string)</code>	Appends the file <i>filename</i> to the end of the open file.
<code>ClearSection start(time) end(time)</code>	Removes all packets between the <i>start</i> and <i>end</i> times in a file.
<code>ConvertFile</code>	Converts the current file to the latest file format.
<code>CutSection start(time) end(time)</code>	Removes all packets between the <i>start</i> and <i>end</i> times in a file.
<code>ExportFile exportFormat(string) destination(string) checkpoint(bool) openAfterExport(bool)</code>	Exports a file using a given <i>exportFormat</i> to a <i>destination</i> .
<code>ExportSection exportFormat(string) destination(string) start(time) end(time) offset(double) checkpoint(bool) openAfterExport(bool)</code>	Exports a section of a file using a given <i>exportFormat</i> to a <i>destination</i> .
<code>MergeInFile filename(string) offset(double)</code>	Merges the Logger file <i>filename</i> into the open file.
<code>OffsetTime offset(double)</code>	Offsets the time of the packets in the open file by the <i>offset</i> value.
<code>PrefixFileToStart filename(string)</code>	Inserts the file named <i>filename</i> at the start of the open file.
<code>TrimStart</code>	Removes any dead time from the start of the file and offset all packets so that the first packet time stamp is 0.0.

Keyboard Shortcuts

Action	Key	Action	Key
Annotations	Shift + A	Offset Time	Ctrl + Shift + O
Append to End	Ctrl + Shift + E	Open	Ctrl + O
Clear Section	Ctrl + Shift + Delete	Pause	Ctrl + P
Close	Ctrl + W	Play	Ctrl + P
Connection	Ctrl + Shift + C	Preferences	Ctrl + Shift + P
Create Instant Point Annotation	\	Prefix to Start	Ctrl + Shift + W
Create Instant Range Annotation	[	Properties	Return
Create Point Annotation	Ctrl + \	Record	Ctrl + R
Create Range Annotation	Ctrl + [	Record As	Ctrl + N
Data View	Shift + D	Record To Multiple Files	Ctrl + M
Decrease Speed	Ctrl + Down	Revert	Ctrl + Shift + R
Delete Annotation	Delete	Save	Ctrl + Shift + S
Delete Section	Ctrl + X	Save As	Ctrl + Shift + A
Disconnect	Ctrl + D	Save Recorded Data	Ctrl + Shift + S
Exit	Ctrl + Q	Set Start and End Times	Ctrl + B
Increase Speed	Ctrl + Up	Set Zoom	Ctrl + I
Jump to End	End	Stop	Ctrl + S
Jump to Start	Home	Toggle Loop at End	Ctrl + L
Jump to Time	Ctrl + T	Trim Start	Ctrl + Shift + T
Merge In File	Ctrl + Shift + I	Update Annotation	]
Modify Annotation	Return	Upgrade File Format	Ctrl + Shift + U
Object Statistics	Shift + O		

MÄK Data Logger Quick Reference Card

Command-line Options

Option	Description
<code>--allowAllRemoteControl mode</code>	Allow (1) or do not allow (0) remote control PDUs.
<code>-c mode</code>	If <i>mode</i> = 1, show the Logger console.
<code>--configFile filename</code>	Specifies the configuration file to load.
<code>--ignore_rest</code>	Ignore all arguments after this one.
<code>-h</code>	Displays a brief summary of Logger command-line arguments.
<code>-l logfile</code>	Specifies a file to send notification messages to. Default: <i>logger.log</i> .
<code>--loggerTickRate rate</code>	The interval for the Logger tick rate, in milliseconds.
<code>-m level</code>	Specifies the notification level for VR-Link messages. Range 0-4.
<code>-n level</code>	Specifies the notification level for Logger messages. Range 0-4.
<code>-r command</code>	Specifies a Logger command to run. Can be used multiple times.
<code>-v</code>	Displays version information and exits.
HLA Only	
<code>-f libname</code>	Specifies the FOM Mapper to use if you are not using the default FOM.
<code>--federateType type</code>	Specifies the federate type for the Logger federate. (HLA Evolved only)
<code>--fomMapperInitData data</code>	Specifies FOM Mapper initialization data, if required.
<code>--fomModules "module1 module2 ... moduleN"</code>	Specifies one or more FOM modules for the exercise. (HLA Evolved only)
<code>-F fedfile</code>	Specifies the FED file for the exercise.
<code>--ignoreAdvisories mode</code>	If 1, ignore HLA advisory messages. If 0, do not ignore.
<code>--MIM MIM_file</code>	Specifies the MIM for the exercise. (HLA Evolved only)
<code>-N federate_name</code>	Specifies the federate name for the Logger federate.
<code>--rprFomVersion version</code>	Specifies the version of the RPR FOM to use.
<code>--rprFomRevision revision</code>	If using RPR FOM 2.0017, specifies the revision to use.
<code>--ignoreAdvisories {0 1}</code>	Specifies whether (1) or not (0) the Logger ignores advisory messages.
<code>--useEmptyFomMapper {0 1}</code>	Specifies whether (1) or not (0) the Logger uses an empty FOM Mapper.
<code>-x ExName</code>	Sets federation execution name. Default: VR-Link.
DIS Only	
<code>-A address</code>	Sets the default IP address to which the Logger sends PDUs. If this option is not used, the Logger uses the computer's broadcast address.
<code>-a appnumber</code>	Specifies the application number.
<code>(-D --deviceAddString) address</code>	Specifies the network device address when a computer has more than one network card.
<code>--disVersion num</code>	Sets the DIS protocol version for PDUs sent onto the network. <i>num</i> can be 4 (DIS 2.0.4), 5 (IEEE 1278.1), or 6 (DIS 2.1.4, IEEE 1278.1a).
<code>-e ID</code>	Sets the exercise ID.
<code>--mcastTtl ttl</code>	Specifies the multicast time-to-live.

Command-line Options

Option	Description
<code>-P <i>port</i></code>	Specifies which DIS port (UDP port if -A not specified, otherwise, TCP) the Logger uses to send or receive data. Default: 3000.
<code>--recvBufferSize <i>size</i></code>	Specifies the receive buffer size.
<code>--remoteControlId <i>ID</i></code>	Specifies the Logger's remote control ID. By default the value is the Logger's site ID.
<code>--sendBufferSize <i>size</i></code>	Specifies the send buffer size.
<code>-s <i>ID</i></code>	Specifies the site ID.



Link - Simulate - Visualize