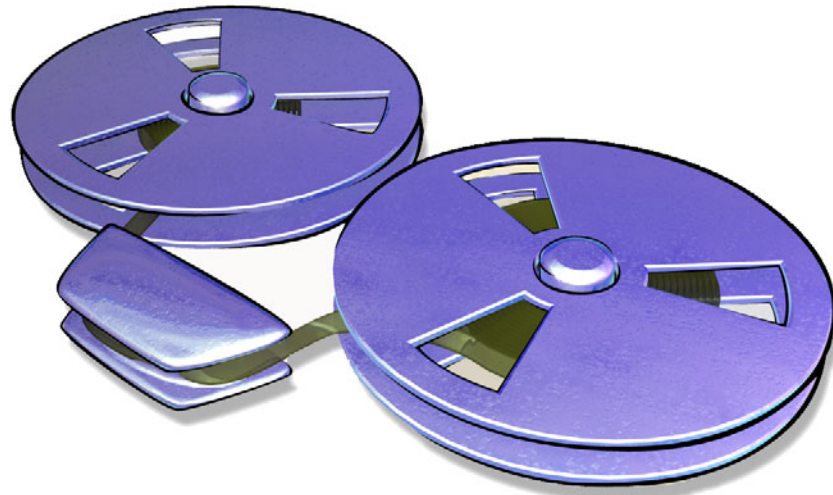




MÄK Data Logger

User's Guide





MÄK Data Logger

User's Guide

Copyright © 2007 MÄK Technologies
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 MÄK Technologies, Inc.

VR-Exchange™ is a trademark of MÄK Technologies. MÄK Technologies®, VR-Forces®, RTIspy®, and VR-Link® are registered trademarks of MÄK Technologies, Inc.

All other trademarks are owned by their respective companies.

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

MÄK Technologies
68 Moulton St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revision LOG-4.2-1-070829



Contents

Preface

How the Manual is Organized	ix
Logger Documentation	x
MÄK Products	xi
How to Contact Us	xiii
Document Conventions	xiv
Hardware and Operating System Naming Conventions	xiv
Mouse Button Naming Conventions.....	xv
Third Party Licenses	xv

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 MÄK Data Logger on a UNIX System	2-3
2.1.3. Silent Installation and Uninstallation on Windows	2-4
2.2. The Logger Directory Structure	2-5

2.3. Setting Up and Using the FLEXlm License Manager	2-6
2.3.1. The FLEXlm Client-Server Architecture	2-6
2.3.2. Installing and Configuring the License Manager	2-7
2.3.3. Installing the License Manager Software	2-7
2.3.4. Requesting a License File	2-8
2.3.5. Putting the License File in the <i>flexlm</i> Directory	2-9
2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable	2-9
2.3.7. Adding Additional License Files	2-10
2.3.8. Specifying a Port for the License Server and Daemon	2-11
2.3.9. Running the License Server	2-12
2.3.10. Running FLEXlm as a Service on Windows	2-13
2.3.11. Troubleshooting the FLEXlm License Manager	2-14
2.4. Installing an RTI	2-16
2.4.1. Installing the MÄK RTI	2-16
2.5. Localizing the Logger Interface	2-17
2.5.1. Translating Other Interface Files	2-18
2.5.2. Applying the Language Files	2-19

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

4.1. Configuring the Logger	4-3
4.2. Configuring the Exercise Connection	4-4
4.2.1. Viewing Connection Properties	4-4
4.2.2. Disconnecting from an Exercise	4-4
4.2.3. Connecting to an Exercise	4-5
4.3. Configuring Indexing	4-6
4.3.1. Configuring Indexing Using the Command Interfaces	4-7
4.4. Setting the Notification Level	4-7
4.4.1. Specifying the Log File	4-8
4.5. Configuring the Logger for HLA	4-9
4.5.1. Specifying the Name of the Logger Federate	4-10
4.5.2. Specifying the Federation Execution Name	4-10
4.5.3. Specifying a FED File	4-11

4.5.4. Specifying the RPR FOM Version	4-11
4.5.5. Specifying a FOM Mapper	4-12
4.5.6. Ignoring Advisory Messages	4-13
4.5.7. Configuring Time Management	4-14
4.5.8. Specifying the Logger Subscription List	4-15
4.5.9. Subscribing Passively to Objects and Interactions	4-17
4.6. Configuring the Logger for DIS	4-18
4.6.1. Specifying a UDP Port	4-19
4.6.2. Setting the Exercise ID	4-19
4.6.3. Setting the Site ID	4-19
4.6.4. Setting the Application Number	4-20
4.6.5. Using Asynchronous I/O	4-20
4.6.6. Specifying a Destination Address for Outgoing and Incoming Data	4-21
4.6.7. Setting the Buffer Size for Sending and Receiving PDUs	4-21
4.6.8. Setting the Multicast Time to Live Value	4-22
4.6.9. Specifying the Timeout	4-22
4.6.10. Specifying the Heartbeat	4-23
4.6.11. Subscribing to Multicast Groups	4-24
4.7. Editing lgrConfig.xml	4-25
4.7.1. Adding a Command to <i>lgrConfig.xml</i>	4-27
4.8. Summary of Command-line Options for the Logger	4-29

Chapter 5 Recording and Playing Logger Files

5.1. Starting the Logger	5-3
5.1.1. Starting the Logger in UNIX	5-3
5.1.2. Starting the Logger in Windows	5-3
5.2. The Logger Window	5-4
5.2.1. Hiding and Displaying Plug-in Views	5-5
5.2.2. Docking and Undocking GUI Plug-in Views and Toolbars	5-5
5.3. Recording Exercises	5-6
5.3.1. Recording Immediately (to a Temporary File)	5-6
5.3.2. Recording to a Specified File	5-6
5.3.3. Recording to Multiple Files	5-7
5.3.4. Pausing the Recording Process	5-9
5.3.5. Stopping the Recording Process	5-9
5.4. Opening Logger Files	5-10
5.4.1. Closing a Logger File	5-10
5.5. Playing Logger Files	5-11
5.5.1. Playing a Logger File	5-11
5.5.2. Pausing Playback of a Logger File	5-11
5.5.3. Playing a Segment of a File	5-12
5.5.4. Looping Playback	5-13
5.5.5. Stopping Playback of a Logger File	5-13
5.5.6. Setting the Playback Speed	5-14

5.6. Saving Files	5-15
5.6.1. File Backups	5-15
5.6.2. Saving a File to a New Name	5-15
5.6.3. Saving a Portion of a File to a New File	5-15
5.6.4. Upgrading a Logger File's Format	5-16
5.7. Zooming the Timeline	5-16
5.7.1. Using the Zoom Sliders	5-16
5.7.2. Setting the Zoom from the Set Zoom Dialog Box	5-17
5.7.3. Resetting the Zoom Level	5-17
5.8. Moving the Timepoint to a Specific Time	5-18
5.8.1. Jumping to the Beginning or End of the Playback Range	5-18
5.8.2. Moving to a Specific Time in a File	5-18
5.8.3. Setting the Time by Dragging the Timeline Cursor	5-18
5.9. Filtering Logger Input and Output	5-19
5.9.1. Filtering DIS PDUs by Entity Type	5-19
5.9.2. Filtering DIS PDUs by PDU Kind	5-20
5.9.3. Including Custom PDUs in Filter Lists	5-21
5.9.4. Filtering HLA Exercises by Class	5-21
5.9.5. Filtering VR-Forces Messages	5-22
5.10. Viewing a File's Packet Count Histogram	5-23
5.11. Viewing Object Statistics	5-24

Chapter 6 Annotating Files

6.1. Annotating a Logger File	6-2
6.1.1. Adding an Annotation	6-2
6.1.2. Saving an Annotation File	6-4
6.1.3. Opening an Annotation File	6-4
6.1.4. Importing an Annotation File	6-5
6.1.5. Selecting an Annotation	6-5
6.1.6. Editing an Annotation	6-5
6.1.7. Deleting an Annotation	6-6
6.1.8. Moving the Timeline Cursor to the Start of an Annotation	6-6
6.1.9. Zooming the Timeline to an Annotation	6-7
6.1.10. Bounding Playback to an Annotation	6-8

Chapter 7 Editing Logger Files

7.1. Editing Files	7-2
7.1.1. Reverting to the Saved Version of a File	7-2
7.2. Clearing and Deleting Segments of a File	7-3
7.3. Merging Files	7-4
7.4. Concatenating Files	7-4
7.5. Trimming Files	7-5
7.6. Offsetting Files	7-5

Chapter 8 Exporting Logger Data

8.1. Exporting Data	8-2
8.2. Exporting Data to Multiple Files	8-3
8.3. Exporting Data to a Database	8-4
8.3.1. Exporting DIS Data to a Database	8-5
8.3.2. Exporting HLA Data to a Database	8-6
8.3.3. The Export Log	8-6
8.3.4. Configuring an ODBC Data Source	8-6
8.4. Configuring a Data Source for Microsoft Access	8-7
8.5. Configuring a Data Source for MySQL	8-10

Chapter 9 Using Logger Commands

9.1. Using Logger Commands on the Command Line	9-2
9.2. Using Logger Commands in the Configuration File	9-2
9.3. Using Logger Commands in the Text Interface	9-2
9.3.1. Entering Commands	9-3
9.3.2. Displaying a List of Commands	9-3
9.4. Command Descriptions	9-4
9.4.1. Argument Types	9-4
9.4.2. System Domain Commands	9-5
9.4.3. Player Domain Commands	9-5
9.4.4. Recorder Domain Commands	9-7
9.4.5. Simulation Domain Commands	9-8
9.4.6. Editor Domain Commands	9-11
9.5. Command List	9-13

Chapter 10 Using The Logger API

10.1. Introduction	10-3
10.2. The Logger Architecture	10-4
10.2.1. Drivers	10-4
10.2.2. Factories	10-5
10.2.3. Interfaces	10-5
10.2.4. The Logger Message Pipeline	10-6
10.2.5. The Simulation Manager	10-6
10.3. Embedding the Logger in an Application	10-7
10.3.1. Embedding Plug-ins in an Application	10-7
10.3.2. The playFile Example	10-8
10.4. Creating a Stand-Alone Logger Application	10-10
10.5. The Logger Plug-in API	10-10
10.5.1. Writing a Plug-In	10-11
10.5.2. Loading Plug-Ins	10-12
10.6. Logger API Coding Conventions	10-13
10.6.1. Namespaces	10-13
10.6.2. Templates	10-13

Contents

10.6.3. Shared Pointers	10-14
10.6.4. Virtual Constructors	10-14
10.6.5. Runtime Values	10-15
10.6.6. Thread Safety	10-16
10.6.7. Interface Signal Slots	10-16
10.6.8. Observer Pattern	10-17
10.7. Logger Libraries	10-17
10.7.1. Other Libraries and Plug-ins	10-18
10.8. Compiling and Linking with the Logger API	10-18
10.8.1. Compiling and Linking for UNIX	10-18
10.8.2. Compiling and Linking for Microsoft Windows	10-19

Chapter 11 Remote Control

11.1. Remote Control Messages	11-2
11.2. Deleting Logger Files	11-3
11.3. Example of a Remote Control Application	11-3

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



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*, explains how to configure the Logger for use with HLA or DIS.

Chapter 5, *Recording and Playing Logger Files*, shows how to record and play back HLA/DIS network activity using the Logger's graphical user interface.

Chapter 6, *Annotating Files* explains how to create and edit annotations in Logger files.

Chapter 7, *Editing Logger Files* explains how to edit Logger files, including cutting segments, inserting files, and offsetting packet timestamps.

Chapter 8, *Exporting Logger Data* explains how to export Logger files to other formats, such as text and to MySQL or Microsoft Access databases.

Chapter 9, *Using Logger Commands*, explains how to execute Logger commands from the command-line, from a console window, and from the configuration file.

Chapter 10, *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 11, *Remote Control*, explains how to control the Logger using view control messages sent from other applications.

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 User's Guide* describes how to install and use the Logger. It is available in printed format and 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 and in printed format.
- ♦ Online help, using the Qt Assistant help browser.
- ♦ *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 MÄK Technologies® line of software products designed to streamline the process of developing and using networked simulated environments. The MÄK Technologies product line includes the following:

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

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

- ♦ MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is optimized for high performance. It has an API, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface for the MÄK RTI rtiexec.
- ♦ VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a graphical user interface, that gives you a two dimensional view of a terrain database.

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.

- ♦ MÄK Stealth and StealthXR. The Stealth displays a realistic, three-dimensional 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.

StealthXR (exaggerated reality) is a separately-licensed component of the Stealth. It combines the features of traditional 2D and 3D displays into one exaggerated reality display. It provides a big picture understanding of a battlefield along with an immersive sense of perspective.

- ♦ **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 graphical user interface and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- ♦ **MÄK Plan View Display.** The Plan View Display (PVD) provides a two-dimensional, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display.
- ♦ **VR-Exchange™.** VR-Exchange allows simulations that use incompatible communications protocols to interoperate. For example, within the HLA world, using VR-Exchange, federations using the HLA RPR FOM 1.0 can interoperate with simulations using RPR FOM 2.0, or federations using different RTIs can interoperate. VR-Exchange supports HLA, TENA, and limited DIS translation.
- ♦ **MÄK Gateway.** The Gateway is a translator application that allows DIS exercises to operate as members of HLA federation executions that use the RPR FOM, thereby preserving the value of simulation code written to support DIS. The MÄK Gateway is a standalone application and does not require you to write additional code for its use.

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

Internet

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

Post

Send postal correspondence to:	MÄK Technologies 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.
/	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*.

Hardware and Operating System Naming Conventions

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

Mouse Button Naming Conventions

An instruction to click the mouse button, refers to clicking the primary mouse button, typically 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 website 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.

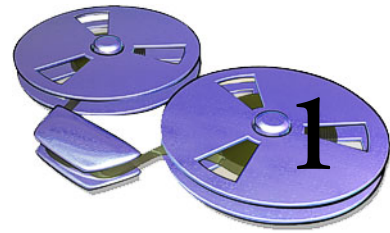
libXML and libICONV

VR-Link and all MÄK software which 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.

- ♦ 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/>



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 playback 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 lets 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 Logger Interface,”](#) on page 2-17 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 [“Filtering Logger Input and Output,”](#) on page 5-19.

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 6, [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.

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 8, [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 2.0.4 and HLA (RPR FOM 1.0) versions of 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 Stealth, MÄK Plan View Display, and VR-Forces.
- ♦ *maklanddemo* — DIS and HLA versions of a complex scenario with land, air, surface, and sub-surface entities. This file uses the Makland database, which is distributed with the MÄK Stealth, MÄK Plan View Display, and VR-Forces.
- ♦ *embarkdemo* — DIS and HLA versions of a complex scenario with land, air, and surface entities. It displays entities embarking and disembarking from parent entities. This file uses the Makland database.



The files contain View Control messages, which cause the MÄK Stealth to follow the action automatically, switching among many predetermined viewpoints. For information about manually controlling the eyepoint in this demo, please see your Stealth 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`.
- ♦ Example plug-ins: *stateObserver* displays a graph of object statistics. *lgrDump* prints details of each packet to the console.
- ♦ An example of a remote control application: *lgrControl*.

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 and the IEEE 1516 specification (SISO Dynamic Link Compatibility HLA API <http://www.sisostds.org/index.cfm>). 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.) Like other MÄK products, the Logger comes with FED files for HLA 1.3 and HLA 1516. 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.

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 MÄK Data Logger on a UNIX System.....	2-3
Silent Installation and Uninstallation on Windows.....	2-4
The Logger Directory Structure	2-5
Setting Up and Using the FLEXlm License Manager	2-6
The FLEXlm Client-Server Architecture	2-6
Installing and Configuring the License Manager.....	2-7
Installing the License Manager Software	2-7
Requesting a License File.....	2-8
Putting the License File in the flexlm Directory.....	2-9
Setting the MAKLMGRD_LICENSE_FILE Variable.....	2-9
Adding Additional License Files.....	2-10
Specifying a Port for the License Server and Daemon	2-11
Running the License Server	2-12
Running FLEXlm as a Service on Windows	2-13
Troubleshooting the FLEXlm License Manager	2-14
Installing an RTI.....	2-16
Installing the MÄK RTI	2-16
Localizing the Logger Interface	2-17
Translating Other Interface Files.....	2-18
Applying the Language Files.....	2-19

2.1. Installing the MÄK Data Logger

This section explains how to install the Logger on a Windows or a UNIX operating system. You must also install the license manager files (please see “[Setting Up and Using the FLEXlm License Manager](#),” on page 2-6.) For HLA, you must install an RTI (please see “[Installing an RTI](#),” on page 2-16.)

2.1.1. Installing the MÄK Data Logger on Windows

This section explains how to install the Logger from CD-ROM or from an ftp download. Before you install the Logger, please read *MÄK Data Logger Release Notes* to see if there are any special instructions for installation.

Installing the MÄK Data Logger from a CD

MÄK products are supplied on “All Products” CDs with a common installation interface.

To install the Logger:

1. Insert the MÄK Products CD into your CD-ROM drive. If the autorun feature on your computer is enabled, the MÄK PC Products Installation window opens. The PC Products Installation window is similar to [Figure 2-1](#).

If the MÄK PC Products Installation window does not open, open the Windows Explorer. In the root directory for your CD drive, double-click *makInst.exe*. The MÄK PC Products Installation window opens.

2. In the MÄK PC Products Installation window, click Data Logger.
3. Follow the instructions of the installation program.

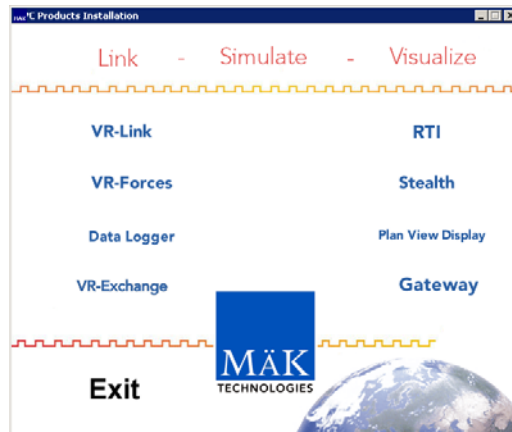


Figure 2-1. MÄK PC Products Installation window

Installing the MÄK Data Logger from a Downloaded File

If you obtained the Logger via ftp, you received a compressed file. It may be a zip file or an executable file.

To install the Logger from a zip file:

1. Copy the zip file to a temporary directory.
 2. Unzip the zip file.
 3. Run *setup.exe*.
 4. Follow the instructions of the installation program.
- To install the Logger from an executable file, run the executable.

2.1.2. Installing the MÄK Data Logger on a UNIX System

This section explains how to install the Logger on UNIX systems from CD-ROM or from an ftp download.

Installing the MÄK Data Logger from a CD-ROM

To install the Logger:

1. Mount the CD-ROM.
2. Run:

```
./install
```
3. Follow the on-screen instructions. You can install the Logger into any directory for which you have write permissions.

Installing the MÄK Data Logger from a Downloaded File

If you obtained the Logger via ftp, you received a compressed tar file.

To install the Logger:

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

```
gtar xzf loggerxx-os.tar.gz
```

where *xx-os* is the release, operating system, and, possibly, the compiler of the product being installed.



You must use *gtar* (from GNU), not the operating system's default *tar*.

2.1.3. Silent Installation and Uninstallation on Windows

If you need to install a MÄK application on many computers in a networked environment, you may want to do so from the command line or from a centralized location (a silent installation.)

To create a silent install:

1. Manually install the application on one computer. An installation configuration file is placed in the root directory of the installation folder. This file is named *install.sss* or *<product_installer>uninstall.sss*, (for example, *logger4.2-win32-msvc++7.1-uninstall.sss*). The *.sss* file is a configuration file containing information such as installation ID, installation directory, and environment variables. It is a text file and can be edited in a text editor.
2. To install the product from the command line install:
 - a. On each computer, put the installer (for example, *logger4.2-win32-msvc++7.1-setup.exe*) and the corresponding *.sss* file in the same folder.
 - b. Rename the *.sss* file the same as the installer (for example, *logger4.2-win32-msvc++7.1-setup.sss*.)
 - c. From the command line, run:

```
product-setup.exe -s
```
 - d. The installer will perform a silent installation based on the configuration in the *.sss* file.
3. To install a product from centralized location:
 - a. Follow steps a and b from the command line installation procedure.
 - b. Write a batch file that contains the installation command.
 - c. Put the batch file on each computer in the folder that has the installer and *.sss* file.
 - d. Run the batch file.

Uninstalling MÄK Products Silently

A silent uninstall runs without the uninstall wizard.

- To run a silent uninstall, in the root directory of the product installation, run:

```
Uninstall -s
```

2.2. The Logger Directory Structure

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

```

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

```

Figure 2-2. 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> and <i>binHLA1516</i>	Executables for use with the HLA RTI 1.3 and 1516.
<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>flexlmx.x</i>	License management software.
<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>loggerControlToolkit</i>	Header files and libraries for the Remote Logger Control API.
<i>translations</i>	This directory contains translations files for localizing the Logger.

2.3. Setting Up and Using the FLEXlm License Manager

MÄK products use the FLEXlm license manager. Before you can use a MÄK product, you must obtain a valid license file and configure the license server and client machines. Contact the MÄK Technologies sales department for information about special licensing agreements.

2.3.1. The FLEXlm Client-Server Architecture

This section explains the FLEXlm architecture to provide a context for the setup instructions that follow. FLEXlm uses a client-server architecture. The FLEXlm server daemon, *lmgrd*, runs on one computer, known as the license server. This machine services every machine (including possibly itself) on which you want to run a MÄK Technologies product. The machines on which the MÄK Technologies products are running are called clients.

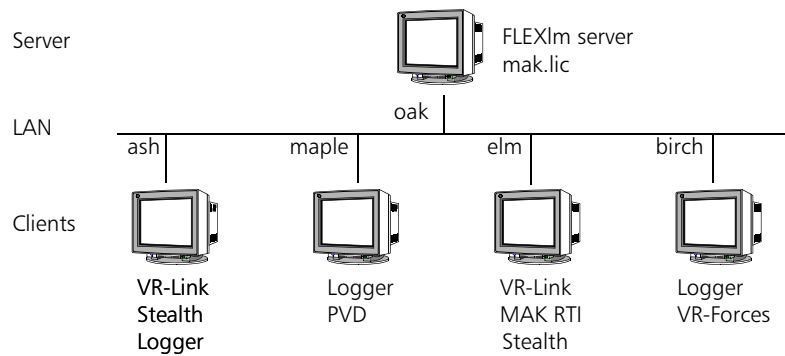
The license server executes vendor daemons (for MÄK Technologies, *maklmgrd*) and uses one or more license files, provided by individual vendors. The license file identifies the products for which you have licenses. On each client machine, the `MAKLMGRD_LICENSE_FILE` environment variable (which you must set) points to the server.

Licenses “float”, so you can install product software on as many computers as you want, but at any given time, you can run only as many instances of a product as you have licensed.



The license key in the license file is tied to the host ID of the server machine. Therefore, if you decide to change the server (or replace the network card), you need to get a new license file. If you change the name of the server, you must update the `MAKLMGRD_LICENSE_FILE` environment variable on every client. Depending on how many clients you have, this may not be a trivial process. Keep this in mind when you choose the machine to use as a license server.

Figure 2-3 illustrates the client-server architecture for FLEXlm and MÄK products.



The MAKLMGRD_LICENSE_FILE environment variable is set to @oak on each client.

Figure 2-3. FLEXlm example architecture

2.3.2. Installing and Configuring the License Manager

The general procedure for installing and configuring license management is as follows (each step is explained in the rest of this section):

1. Install license manager software.
2. Request a license file.
3. Put the license file in the *flexlm* directory on the license server.
4. Set the MAKLMGRD_LICENSE_FILE environment variable on all client machines.

If you have MÄK Technologies products installed and you purchase additional licenses or products, you must get a license file for the new products or licenses. For this procedure, please see [“Adding Additional License Files,”](#) on page 2-10.

2.3.3. Installing the License Manager Software

As part of the installation of MÄK products, the FLEXlm software is installed in *./flexlmx.x* (where *x.x* is the FLEXlm version).

1. Install your MÄK products on the machines on which you want to run them.
2. Copy the files in *./flexlmx.x* to the server machine. For Windows, copy the files to *C:\flexlm*. For a UNIX computer, a typical location is */usr/local/flexlm*.



Note the difference between the directory name of the supplied files and the directory to which you copy them.

2.3.4. Requesting a License File

When you request a license file, you need to know the:

- ♦ Host ID of the license server
- ♦ Name of the license server
- ♦ MÄK invoice number of your purchase
- ♦ Basic contact information for you and your company
- ♦ Platforms for which you have purchased MÄK products
- ♦ Number of licenses for each platform and product.

When you have this information, go to <http://www.mak.com/support/licenses.php> and complete the license key request form. MÄK will e-mail you a license file.

Identifying the Host ID and License Server Name in Windows

To find out the host ID and license server name in Windows:

1. In the *flexlm* directory, run *lmtools.exe*. The LMTOOLS application opens.
2. Select the System Settings tab. The host ID is the string in the Ethernet Address text box. The license server name is the name in the Computer/Hostname text box.

Identifying the Host ID and License Server Name on UNIX Computers

To find out your host ID:

1. On the license server, in a command window, change to the *flexlm* directory.
2. Execute *lmhostid*. This program reports a unique number that MÄK uses to generate your license activation codes. Write down the number.



The host ID is keyed to your ethernet card, so if you change your network card, you must request a new license file.

- To find out the name of the server machine, at a command prompt, type `hostname`.

2.3.5. Putting the License File in the *flexlm* Directory

The license file you receive may be an attachment or embedded in an email.

1. If the attachment is not named *mak.lic*, rename it *mak.lic*.
If the license is embedded in an email, copy it into an empty text file. Name the file *mak.lic*.
2. Put the license file in the *flexlm* directory on the license server.



If you already have a *mak.lic* file in the *flexlm* directory, do not overwrite the file. Please see the instructions in [“Adding Additional License Files,”](#) on page 2-10.

2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable

The MAKLMGRD_LICENSE_FILE environment variable identifies the server machine so that the FLEXlm software on a client can check out a license when you run a MÄK Technologies product. You must set MAKLMGRD_LICENSE_FILE on every client machine.



If you are running MÄK Technologies products on the license server machine, it is also a client, so you must set MAKLMGRD_LICENSE_FILE on that machine.

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 MÄK Technologies products run on.

Windows XP

To add the MAKLMGRD_LICENSE_FILE in Windows XP:

1. On the Start menu, choose My Computer.
2. In the My Computer window, under System Tasks, click View System Information. 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. See [“Running the License Server,”](#) on page 2-12.

2.3.7. Adding Additional License Files

If you buy additional MÄK products, or additional licenses for products you already have, you must get an additional license file.

- To request an additional license file, follow the same procedure as for your first license file. Please see [“Requesting a License File,”](#) on page 2-8.

To add a new license file to your license server:

1. When you get the new license file, name it `mak2.lic`, or some other name ending in `.lic`, so that it has a different name from the license files currently in the `flexlm` directory on the license server.
2. Put the renamed file in the `flexlm` directory on the license server, with the other license files.
3. Stop the license server daemon and restart it.

You can now use the new products or additional product licenses.

2.3.8. Specifying a Port for the License Server and Daemon

On some systems, specifying a port for the license server, the *maklmgrd* daemon, or both, can resolve connection problems. If you want to run MÄK software through a firewall, you must specify both ports. When used through a firewall, users typically open specific ports for simulation traffic and license verification messages. By specifying both ports, FLEXlm knows which ports have been opened up for use.

You specify a port for the license server by editing the license file and adding the license server port to the MAKLMGRD_LICENSE_FILE environment variable. To specify a port for the *maklmgrd* daemon, you just have to edit the license file.

To specify a port for the license server:

1. Open the license file in a text editor.
2. Add the port number to the SERVER line as follows:

```
SERVER host_name host_ID port
```

For example:

```
SERVER oak 12345ABCD957 9237
```

3. Add the port number to the MAKLMGRD_LICENSE_FILE environment variable on all computers that use this license server as follows:

```
port@host_name
```

For example:

```
9237@oak
```

To specify a port for the *maklmgrd* daemon:

1. Open the license file in a text editor.
2. Add the port number to the DAEMON line as follows:

```
DAEMON maklmgrd ./maklmgrd=port
```

For example:

```
DAEMON maklmgrd ./maklmgrd=2568
```

2.3.9. Running the License Server

The FLEXlm license server daemon must be running on the server machine before you can run your MÄK applications.

- To start the license server daemon, on the server machine only, execute `runLm` from the `./flexlm` directory.



Do not execute more than one instance of `runLm` at a time. If you aren't sure whether or not the license server is running, execute `lmstat`.

- To obtain the current status of the server, run `lmstat`.

Shutting Down the License Server

- To force all applications to check in their licenses and shut down the license server daemon, run `lmdown`.

2.3.10. Running FLEXlm as a Service on Windows

You can configure FLEXlm to run as a service on Windows XP.

To configure FLEXlm as a service:

1. Install the license management software and obtain a license as described previously in this chapter.
2. Run *C:\flexlm\lmtools.exe*.
3. On the Service/License File tab, select the Configuration Using Services option.
4. Select the Config Services tab.
5. In the Service Name box, enter a name for the service.
6. In the Path To The *lmgrd.exe* File box, enter the fully qualified path of the *lmgrd.exe* file (for example, *C:\flexlm\lmgrd.exe*), or use the Browse button to select the file.
7. In the Path To The License File box, enter the fully qualified path of the license file (for example, *C:\flexlm\mak.lic*), or use the Browse button to select the file. If you have multiple license files, enter the directory in which they are located.
8. In the Path To The Debug Log File box, enter the fully qualified path of the FLEXlm debug file, or use the Browse button to select a directory. If no debug file currently exists, enter a file name with a *.log* extension.
9. Select the User Services check box. The Start Server At Power Up check box becomes enabled.
10. Select the Start Server At Power Up check box.
11. Click the Save Service button and click Yes at the prompt.
12. Select the Start/Stop/Reread tab.
13. Click Start Server.

2.3.11. Troubleshooting the FLEXlm License Manager

For general troubleshooting information, refer to the FLEXlm FAQ at http://www.macrovision.com/support/by_category/FLEXlm_faqs.shtml. It describes common problems, including the majority that have been reported to MÄK support engineers. Please consult the FAQ first when you have a problem.

Unable to Get a License

If you are unable to get a license for a MÄK application and the FLEXlm log file (*LOG*) indicates that the port is in use, it is possible that you have more than one *lmgrd* or *maklmgrd* processes running. It is also possible that an application that was using a license is thought to have terminated, but is still alive.

To verify and resolve these possibilities:

1. Please see if more than one *lmgrd* or *maklmgrd* process is running.

On Windows, check the Task Manager. (To open the Task Manager, press **Ctrl+Alt+Delete** and click Task Manager.)

On UNIX, enter the following commands:

```
ps -aux | grep lmgrd
ps -aux | grep maklmgrd
```

If more than one instance of either process is running, write down the process ids (PID).



Some UNIX systems use `ps -aux`, others `ps -ef` to check processes. Use the command appropriate for your operating system.

2. If there are old processes present that may be using a license, kill them.

On UNIX, kill a process with the *kill* command, for example:

```
kill -9 1385 878
```

where 1385 and 878 are process IDs.

On Windows, select a process in the Task Manager, Processes tab and click End Process.

It is important that you kill all the processes to ensure a clean restart for the license server.

3. If in step 2 you killed the license server, start it with the *runLm* command.

Preventing Multiple License Manager Processes

We recommend that you keep the license server running and that you not start and stop it as you run applications. If you prefer to stop the license server when you are not using it, always use `lmdown` to stop it.

Whenever you start the license manager, first run `lmstat` to be sure that there is not already a license server process running. Following this practice will ensure that you do not start multiple license servers.

License Manager Cannot Find MÄK License Management Executable

If your license manager gives you an error message indicating that it cannot find the license management executable, try inserting the pathname in the license file as follows:

1. Open the license file (*mak.lic*) in a text editor.
2. Edit the license file to specify the location of the *maklmgrd* executable on the server machine. The file supplied by MÄK represents this location with a dot as shown below:

```
DAEMON maklmgrd .
```

Replace the dot with your absolute path to *maklmgrd*, for example:

```
DAEMON maklmgrd "C:\flexlm\maklmgrd"
```



On PCs, you must enclose the path in quotes.
The drive letter must be uppercase.

The License Manager Reports an Unsupported Product

If you receive an error message indicating an unsupported product, you are trying to use a MÄK product for which you do not have a license or you are trying to run a version of a product that is more recent than the product maintenance expiration date in the license file. Please contact sales@mak.com to purchase a license or update maintenance.

If you are trying to run your application with an RTI other than the MÄK RTI and you get the unsupported product message, it is possible that your application is finding the DLLs for an unlicensed version of the MÄK RTI before it finds the other RTI. To fix this problem do the following:

- ♦ Make sure that the other RTI path precedes the MÄK RTI path in your path environment variable.
- ♦ Check the directory in which the application is located to be sure that there are no DLLs from the MÄK RTI in that directory.

2.4. Installing an RTI

An RTI (Run-Time Infrastructure) is a software library (and perhaps supporting executables) that implements the 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 or the Product Versions page on the MÄK Technologies web site at: <http://www.mak.com/support/productversions.php>.

2.4.1. Installing the MÄK RTI

To install the MÄK RTI, follow the instructions in Chapter 2 of *MÄK RTI Reference Manual*. (If you do not have a printed version of the manual, you can view the PDF versions in *makrtix.x/doc*.)

Configuring Your System to Use the MÄK RTI

The RTI dynamic libraries need to be located somewhere on your dynamic library search path. The path is indicated by an environment variable as follows:

- For IRIX6n32 use LD_LIBRARYN32_PATH
- On other UNIX platforms use LD_LIBRARY_PATH
- On Windows, use 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 FLEXlm license server is running, then start the applications, and they should be able to communicate. (For information about the license server, please see “[Setting Up and Using the FLEXlm License Manager](#),” on page 2-6.) Please see *MÄK RTI Reference Manual* for information about the `rtiexec` and changing the networking parameters such as multicast addresses and UDP ports.

2.5. Localizing the Logger Interface

You can localize the Logger interface so that it uses languages other than English.

To localize the Logger, 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/logger_untranslated.ts`. The translation file is opened and a list of classes is displayed on the left side of the window, as illustrated.

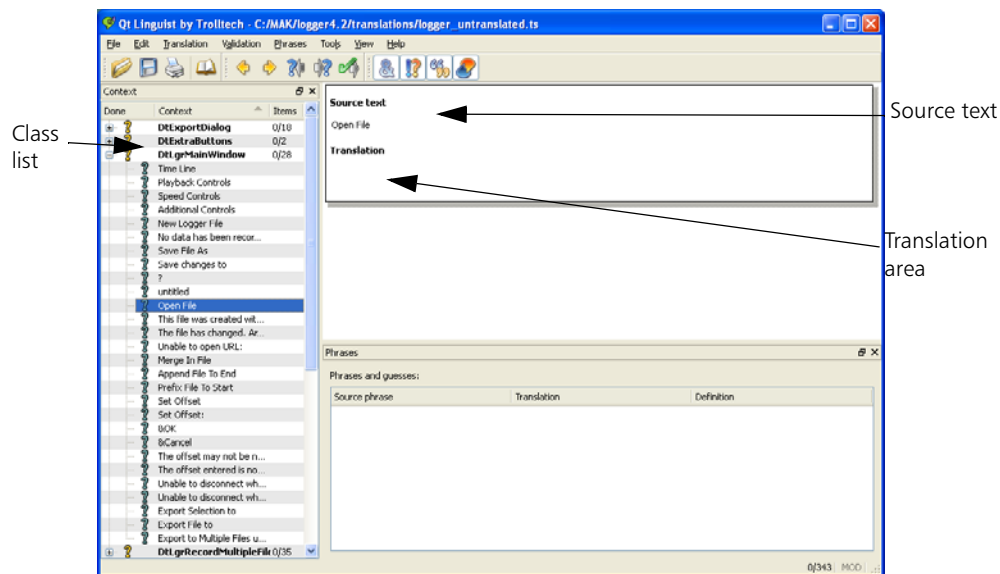


Figure 2-4. Qt Linguist window

3. 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. For example, in [Figure 2-4](#), the *DtLgrMainWindow* class is expanded and the entry *Open File* is selected. The blue question mark next to the term indicates that the text has not been translated yet.

4. Place the cursor in the space below the word “Translation”.
5. Type the translation for “Open File”. As you type, the translation is displayed in the Items column and the question mark changes to yellow (Figure 2-5).

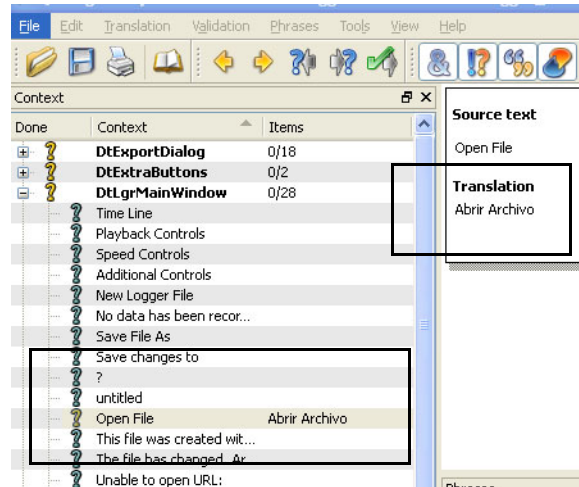


Figure 2-5. Translated text

6. 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. When you translate all elements for a class, a green check mark replaces the question mark next to the class name.
7. Repeat this process for each element of source text for each class.
8. Choose File → Release. A Save As dialog box opens. Change the name in the File Name box from *logger_untranslated.qm* 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 *logger_es.qm*. Save the *.qm* file in the *.translations* directory.
9. Save your translated *.ts* file.

2.5.1. Translating Other Interface Files

The *logger_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 system locale to the proper language.
- ♦ Set the **locale** variable in *lgrConfig.xml* to the two-letter country code for the local language.



Logger Concepts

This chapter describes Logger concepts that are common to all interfaces.

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.

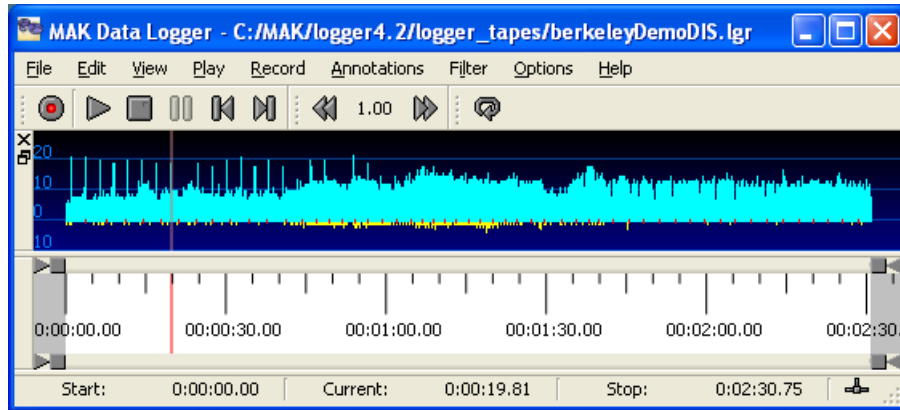


Figure 3-1. The Logger window

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 4-4.

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 either 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.

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.)

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 5-18.

DIS only

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 Settings 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 uses 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.

Recording Using Time Management

In non-time managed recording, the arrival time of a packet is the current Logger time. In time-constrained recording the arrival time of the packet is the time-stamp of the update (as received from the RTI) for Time Stamp Order (TSO) updates and it is the current value of the federation time for Receive Order (RO) updates. The Logger advances its federation time when it receives TSO updates. If there are no time-regulating federates, the Logger will not advance federation time.

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 5-14.

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

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 4-23.

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 4-22.

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 the MÄK 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

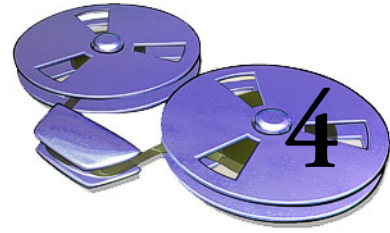
You can change the FED file between recording and playback. As long as the FED file you are using on playback contains object and interaction classes, and attributes and parameters with the same names and format as those that were recorded, you can play back your file.

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

This chapter explains how to configure connections settings for DIS and HLA.

Configuring the Logger.....	4-3
Configuring the Exercise Connection	4-4
Viewing Connection Properties	4-4
Disconnecting from an Exercise.....	4-4
Connecting to an Exercise	4-5
Configuring Indexing	4-6
Configuring Indexing Using the Command Interfaces	4-7
Setting the Notification Level.....	4-7
Specifying the Log File	4-8
Configuring the Logger for HLA	4-9
Specifying the Name of the Logger Federate	4-10
Specifying the Federation Execution Name.....	4-10
Specifying a FED File.....	4-11
Specifying the RPR FOM Version	4-11
Specifying a FOM Mapper	4-12
Ignoring Advisory Messages.....	4-13
Configuring Time Management	4-14
Specifying the Logger Subscription List	4-15
Subscribing Passively to Objects and Interactions	4-17
Configuring the Logger for DIS.....	4-18
Specifying a UDP Port	4-19
Setting the Exercise ID	4-19
Setting the Site ID.....	4-19
Setting the Application Number.....	4-20
Using Asynchronous I/O	4-20

Configuring the Logger

Specifying a Destination Address for Outgoing and Incoming Data	4-21
Setting the Buffer Size for Sending and Receiving PDUs	4-21
Setting the Multicast Time to Live Value	4-22
Specifying the Timeout	4-22
Specifying the Heartbeat	4-23
Subscribing to Multicast Groups	4-24
Editing lgrConfig.xml	4-25
Adding a Command to lgrConfig.xml	4-27
Summary of Command-line Options for the Logger	4-29

4.1. Configuring the Logger

Logger configuration consists mostly of configuring the protocol-specific connection to the simulation network. Additional configuration options include setting the notification level for messages, checkpointing, and the state in which the Logger starts up.

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 4-29 or the Quick Reference Card. For descriptions of Logger commands and how to use them, please see Chapter 9, [Using Logger Commands](#) or the Quick Reference Card. For details about the configuration file, please see [“Editing lgrConfig.xml,”](#) on page 4-25.

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

4.2. 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).

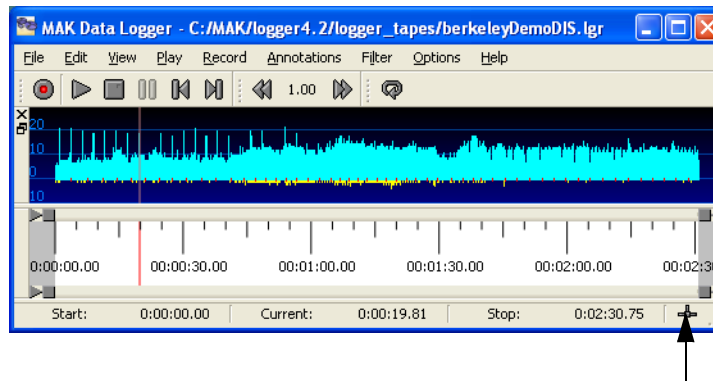


Figure 4-1. Connection status

4.2.1. 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 Settings dialog box opens (Figure 4-2).

4.2.2. Disconnecting from an Exercise

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.
- To disconnect from an exercise from the text interface, issue the `Disconnect` command.

4.2.3. Connecting to an Exercise

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 4-9 and “Configuring the Logger for DIS,” on page 4-18.

To connect to an exercise from the GUI:

1. Choose File → Connection. The Connection Settings dialog box opens (Figure 4-2). It displays the current settings for the protocol you are using (HLA or DIS.)

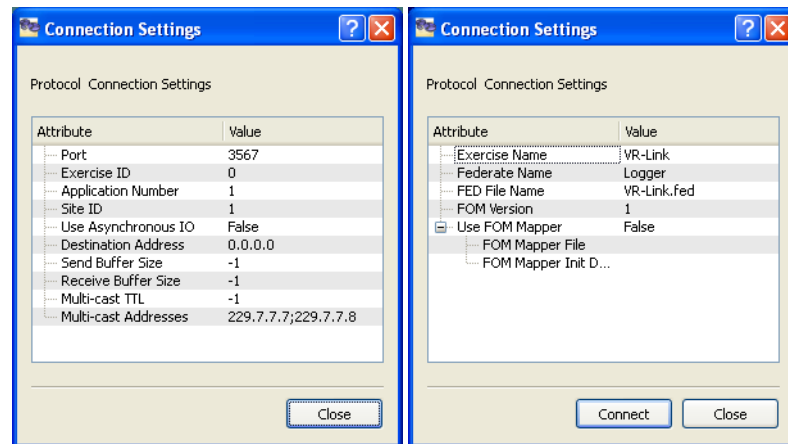


Figure 4-2. Connection Settings dialog box for DIS and HLA

2. To change an attribute value, click the attribute and type or select a new value.
 3. Click Connect.
- To quickly connect to an exercise using the stored connection settings, double-click the connection status icon.

Connecting to an Exercise Using the Command Interfaces

- To connect to an exercise using a command interface, issue the Connect command (Table 4-1).

Table 4-1: Connect command

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

4.3. Configuring Indexing

Logger files contain index data that enable quick jumps from one time point to another. 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.

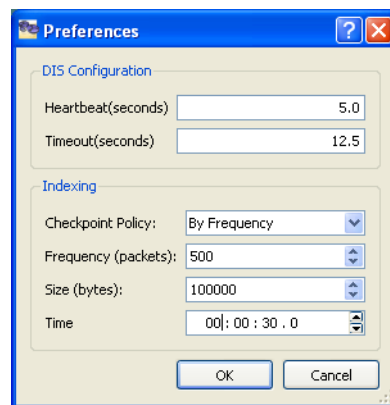


Figure 4-3. Preferences dialog box (DIS)

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.

4.3.1. Configuring Indexing Using the Command Interfaces

- To configure indexing using a command interface, issue the `SetCheckpointParameters` command ([Table 4-2](#)).

Table 4-2: Checkpointing command

Interface	Command
Command line	<code>logger -r "SetCheckpointParameters 0 0 500 0"</code>
Text interface	<code>SetCheckpointParameters 0 0 500 0</code>
<i>lgrConfig.xml</i>	Add <code>SetCheckpointParameters</code> to the configuration file.

4.4. 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 4-3](#)).

Table 4-3: 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.

4.4.1. 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 4-4](#)).

Table 4-4: 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.

4.5. 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 [“Connecting to an Exercise,”](#) on page 4-5.

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

```
SetConnectionParameters federateName federationName
                        fedFile fomVersion fomMapper fomMapperData
```

For example:

```
SetConnectionParameters Logger VR-Link VR-Link.fed 1.0
                        myFomMapper myInitData
```

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

```
logger -r "SetConnectionParameters federateName
           federationName fedFile fomVersion fomMapper
           fomMapperData"
```

For example:

```
logger -r "SetConnectionParameters Logger VR-Link
           VR-Link.fed 1.0 myFomMapper myInitData"
```

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 "" ""
```

On the command line, you must protect double-quotes with a backslash, as follows:

```
logger -r "SetConnectionParameters Logger VR-Link
           VR-Link.fed 1.0 \"\" \"\""
```

For details about commands and command syntax, please see Chapter 9, [Using Logger Commands](#).

4.5.1. 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 4-5](#)).

Table 4-5: Specifying the Logger federate

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

4.5.2. 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 4-11.

- To specify a federation execution name, use the `-x` command-line option or the `SetConnectionParameters` command ([Table 4-6](#)).

Table 4-6: Specifying the federation execution

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

4.5.3. 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. Please see *MÄK RTI Reference Manual* for details. MÄK products come with a FED (and XML) file for each version of the RPR FOM that they support.

- To specify a FED file, use the `-F` command-line option or the `SetConnectionParameters` command (Table 4-7).

Table 4-7: Specifying the FED file

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

4.5.4. 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 4-8).

Table 4-8: Specifying the RPR FOM version

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

4.5.5. Specifying a FOM Mapper

The Logger supports different FOMs using VR-Link FOM Mappers.



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

- To specify a FOM Mapper, use the `-f` command-line option or the `SetConnectionParameters` command (Table 4-9).

Table 4-9: Specifying a FOM Mapper

Interface	Command
GUI	Connection Settings dialog box: Set Use FOM Mapper to True and specify the FOM Mapper File attribute.
Command line	<code>logger -f myFomMapper.dll</code> Or use the <code>SetConnectionParameters</code> command.
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fomMapperLib</code> variable is 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 4-10).

Table 4-10: Specifying FOM Mapper initialization data

Interface	Command
GUI	Connection Settings 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. Or use the <code>SetConnectionParameters</code> command.
Text interface	Use the <code>SetConnectionParameters</code> command.
<i>lgrConfig.xml</i>	The <code>fomMapperInitData</code> variable is in the default configuration file.

4.5.6. Ignoring Advisory Messages

- To have the Logger ignore advisory messages, use the `SetIgnoreAdvisories` command (Table 4-11).

Table 4-11: Specifying response to advisory messages

Interface	Command
Command line	<code>logger -r "SetIgnoreAdvisories 1"</code>
Text interface	<code>SetIgnoreAdvisories 1</code>
<i>lgrConfig.xml</i>	Add <code>SetIgnoreAdvisories</code> to the configuration file.

4.5.7. 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).

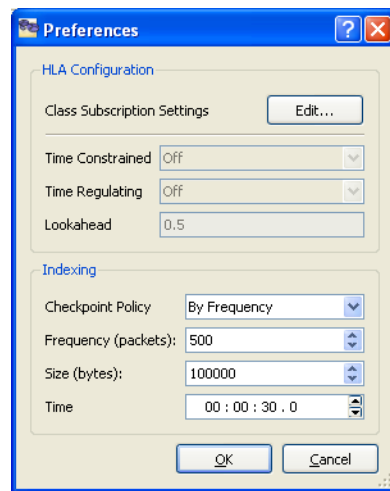


Figure 4-4. Preferences dialog box for HLA

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).

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.

4.5.8. 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.

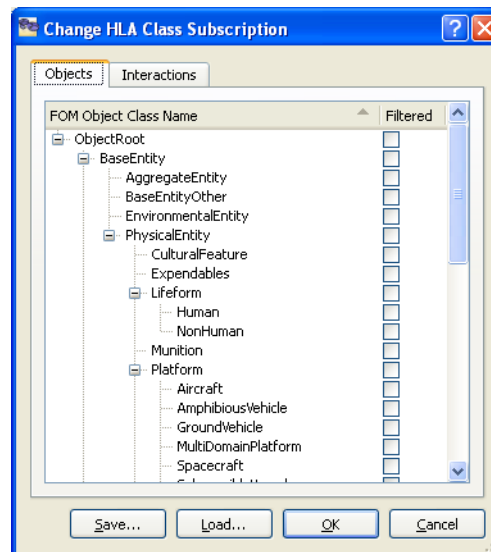


Figure 4-5. Change HLA Class Subscription dialog box

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).

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 4-28.
	 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 4-15.
 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).

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).

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>lgrConfig.xml</i>	Add <code>LoadSubscriptionListFile</code> to the configuration file.

4.5.9. 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).

Table 4-16: Specifying passive subscription

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

4.6. Configuring the Logger for DIS

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

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

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

For example:

```
SetConnectionParameters 3000 1 1 2 True 205.205.205.205 -1
                        -1 -1 "225.0.0.118" "225.0.0.202"
```

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

```
logger -r "SetConnectionParameters port exerciseId siteId
          appNumber useAsynchIO destAddr sendBufferSize
          recvBufferSize multicastTtl mcastAddresses"
```

For example:

```
logger -r "SetConnectionParameters 3000 1 1 2 True
          205.205.205.205 -1 -1 -1 \"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 3456 1 1 2 true 205.205.205.205 -1
                        -1 -1 ""
```

For details about commands and command syntax, please see Chapter 9, [Using Logger Commands](#).

4.6.1. 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](#)).

Table 4-17: Specifying the UDP port

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

4.6.2. Setting the Exercise ID

- To specify the exercise ID, use the `-e` command-line option or the `SetConnectionParameters` command ([Table 4-18](#)).

Table 4-18: Specifying the exercise ID

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

4.6.3. Setting the Site ID

- To specify the site ID, use the `-s` command-line option or the `SetConnectionParameters` command ([Table 4-19](#)).

Table 4-19: Specifying the site ID

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

4.6.4. Setting the Application Number

- To specify the application number, use the `-a` command-line option or the `SetConnectionParameters` command ([Table 4-20](#)).

Table 4-20: Specifying the application number

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

4.6.5. 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](#)).

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

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

4.6.6. 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).

Table 4-22: Specifying the destination address

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

4.6.7. 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 4-23).

Table 4-23: Specifying the send and receive buffers

Interface	Command
GUI	Connection Settings dialog box: Send Buffer Size and Receive Buffer Size attributes.
Command line	<code>logger --receiveBufferSize 64000 --sendBufferSize 64000</code> Or use the <code>SetConnectionParameters</code> command.
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.

4.6.8. 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 4-24](#)).

Table 4-24: Specifying the multicast time-to-live

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

4.6.9. 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](#),” 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 4-25](#)).

Table 4-25: 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.

4.6.10. 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](#),” on page 3-6.

To set the heartbeat in the GUI:

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

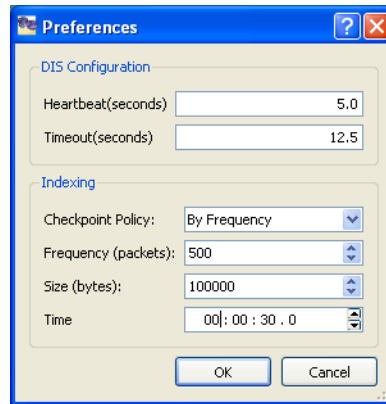


Figure 4-6. Preferences dialog box for DIS

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 4-26](#)).

Table 4-26: 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.

4.6.11. 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 4-27).

Table 4-27: Subscribing to multicast groups

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

4.7. 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 plug-ins 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 9-4.

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"/>
  </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="fomMapper" var="fomMapperLib"></param>
        <param name="fomMapperData" var="fomMapperInitData"></param>
      </command>
      <command domain="Simulation" protocol="None" command="Connect"/>
    </command>
  </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 9-4 for a list of commands by domain.
- ♦ *protocol* is None, DIS, HLA 13, or HLA 1516.
- ♦ *command* is the command you want to run, from the list of supported commands in Chapter 9, [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 9-4.
- ♦ *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 9-4.
- ♦ *value* is the value you want to assign to the variable.

For example:

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

4.7.1. 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 9-4. 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 9-2 and [“Using Logger Commands in the Text Interface,”](#) on page 9-2.

4.8. Summary of Command-line Options for the Logger

Table 4-28 lists command-line options. Command-line options override the comparable settings in the *lgrConfig.xml* file.

Table 4-28: Command-line options (Sheet 1 of 2)

Option	Description
{-c --showConsole} <i>mode</i>	Enables (1) or disables (0) display of a console window.
--configFile <i>filename</i>	Specifies the configuration file to load.
<i>filename</i>	Specifies a Logger file to load. This option uses an implicit -r OpenFile.
{-- --ignore_rest}	Ignore all arguments after this one.
{-h --help}	Displays a brief summary of Logger command-line arguments.
{-l --logFileName} <i>filename</i>	Specifies a file to send notification messages to. Default: <i>logger.log</i> .
{-m --messageLevel} <i>level</i>	Specifies the notification level for Logger messages for the log file. Range: 0-4.
{-n --notifyLevel} <i>level</i>	Specifies the notification level for VR-Link messages for the log file. Range: 0-4.
{-r --runCommand} <i>command</i>	Specifies a Logger command to run. You can specify -r multiple times. For more information, please see “Using Logger Commands on the Command Line,” on page 9-2.
{-v --version}	Displays version information and exits.
HLA Only	
{-f --fomMapperLib} <i>libname</i>	Specifies the FOM Mapper to use if you are not using the default FOM.
--fomMapperInitData <i>data</i>	Specifies FOM Mapper initialization data, if required.
{-F --fedFileName} <i>fedfile</i>	Specifies the FED file for the exercise.
{-N --federateName} <i>federate_name</i>	Specifies the federate name for the Logger federate.
--rprFomVersion <i>version</i>	Specifies the version of the RPR FOM to use.
{-x --execName} <i>ExName</i>	Specifies the federation execution name. Default: VR-Link.

Table 4-28: Command-line options (Sheet 2 of 2)

Option	Description
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 4-21.
<code>{-a --appNumber} appnumber</code>	Specifies the application number.
<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.
<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.....	5-3
Starting the Logger in UNIX.....	5-3
Starting the Logger in Windows.....	5-3
The Logger Window.....	5-4
Hiding and Displaying Plug-in Views	5-5
Docking and Undocking GUI Plug-in Views and Toolbars	5-5
Recording Exercises.....	5-6
Recording Immediately (to a Temporary File)	5-6
Recording to a Specified File	5-6
Recording to Multiple Files	5-7
Pausing the Recording Process.....	5-9
Stopping the Recording Process.....	5-9
Opening Logger Files.....	5-10
Closing a Logger File.....	5-10
Playing Logger Files	5-11
Playing a Logger File	5-11
Pausing Playback of a Logger File	5-11
Playing a Segment of a File.....	5-12
Looping Playback.....	5-13
Stopping Playback of a Logger File.....	5-13
Setting the Playback Speed	5-14
Saving Files	5-15
File Backups	5-15
Saving a File to a New Name.....	5-15
Saving a Portion of a File to a New File	5-15

Upgrading a Logger File's Format	5-16
Zooming the Timeline.....	5-16
Using the Zoom Sliders.....	5-16
Setting the Zoom from the Set Zoom Dialog Box	5-17
Resetting the Zoom Level.....	5-17
Moving the Timepoint to a Specific Time.....	5-18
Jumping to the Beginning or End of the Playback Range	5-18
Moving to a Specific Time in a File	5-18
Setting the Time by Dragging the Timeline Cursor	5-18
Filtering Logger Input and Output	5-19
Filtering DIS PDUs by Entity Type.....	5-19
Filtering DIS PDUs by PDU Kind.....	5-20
Including Custom PDUs in Filter Lists	5-21
Filtering HLA Exercises by Class	5-21
Filtering VR-Forces Messages	5-22
Viewing a File's Packet Count Histogram.....	5-23
Viewing Object Statistics	5-24

5.1. 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 4-29, lists command-line options for the Logger.

5.1.1. Starting the Logger in UNIX

To start the Logger:

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

```
logger [filename options]
```

5.1.2. Starting the Logger in Windows

Start the Logger in any of the following ways:

- ♦ On the Start Menu, choose Programs → MÄK Technologies → Logger X.X → Logger *protocol*, where *protocol* is DIS, HLA 1.3, or HLA 1516.
- ♦ 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.

5.2. The Logger Window

The Logger window has a set of control buttons, a timeline, and a status bar (Figure 5-1). 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.

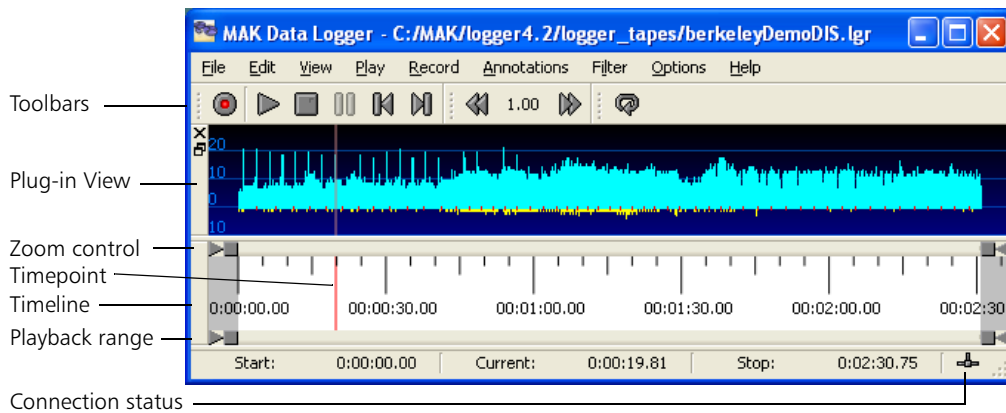


Figure 5-1. Logger window with a Logger file loaded

The Logger window controls, as identified in Figure 5-2, are as follows:

- ♦ 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).



Figure 5-2. Logger controls

5.2.1. 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 5-1).

5.2.2. 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.

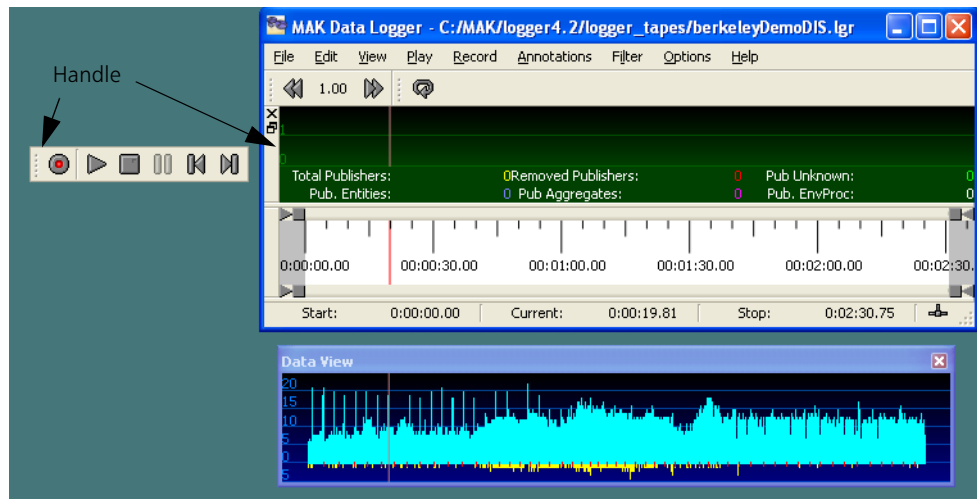


Figure 5-3. Undocked GUI components

- 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.

5.3. 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.

5.3.1. Recording Immediately (to a Temporary File)

- To begin recording, choose Record → Record, or click the Record button.

5.3.2. 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.

5.3.3. 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.

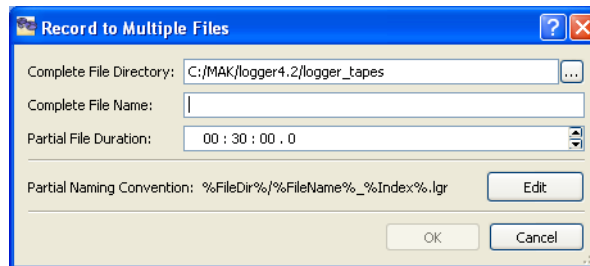


Figure 5-4. Record to Multiple Files dialog box

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 5-8.
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.

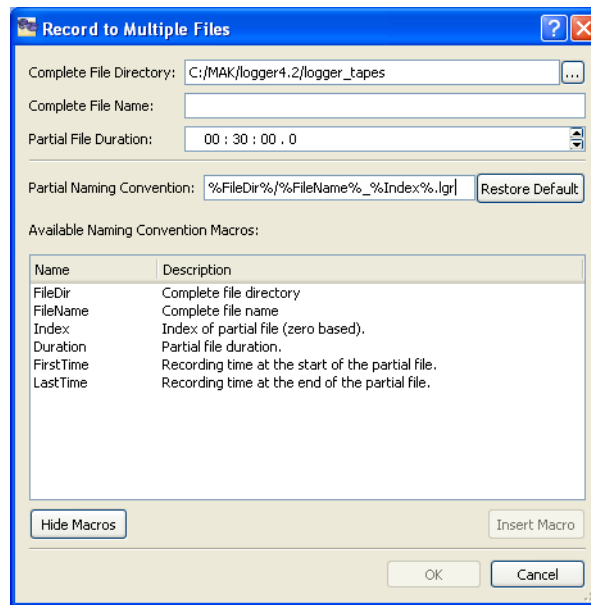


Figure 5-5. Record to Multiple Files dialog box in Edit mode

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 5-1](#).
4. Click Hide Macros to close the macro list.
5. Click Restore Default to revert to the default naming convention.

Table 5-1: 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.

5.3.4. 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.

5.3.5. 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.

5.4. 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 5-23.)

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 5-16.
 - 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.
-

5.4.1. Closing a Logger File

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

5.5. 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. See “[Some Logger Features are Restricted When You Use Time Management](#),” on page 3-5.)

5.5.1. 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.

5.5.2. Pausing Playback of a Logger File

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

5.5.3. 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 5-6) 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.

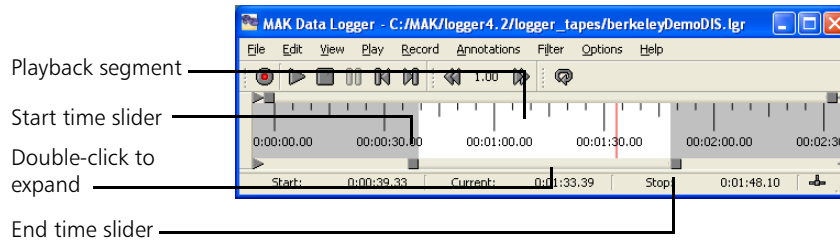


Figure 5-6. Timeline sliders

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 5-7).

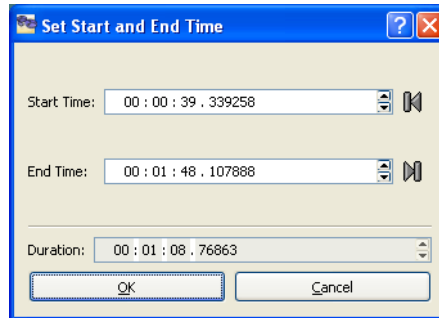


Figure 5-7. Set Start and End Time dialog box

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 5-6).

5.5.4. 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 5-2).

5.5.5. Stopping Playback of a Logger File

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

5.5.6. 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. The Set Playback Speed dialog box opens.

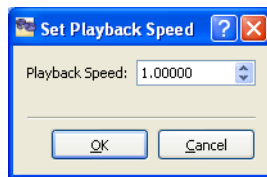


Figure 5-8. Set Playback Speed dialog box

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 Speed 2X (or Decrease Speed 1/2 X), or click the Increase Speed or Decrease Speed button on the icon bar (Figure 5-2).

5.6. 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.)

5.6.1. 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.

5.6.2. 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.

5.6.3. 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.

5.6.4. 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.

5.7. 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.

5.7.1. Using the Zoom Sliders

- To zoom the timeline in or out using a zoom slider, drag a zoom slider (Figure 5-9) to the left or right.
- To return a zoom slider to its default location, click the appropriate zoom out arrow.

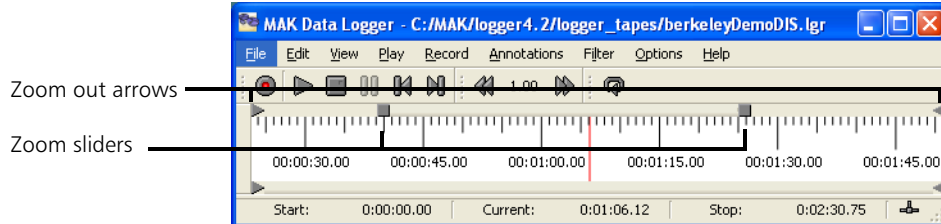


Figure 5-9. Zoom sliders

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 5-12 for details about the playback time.

5.7.2. 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 opens (Figure 5-10).

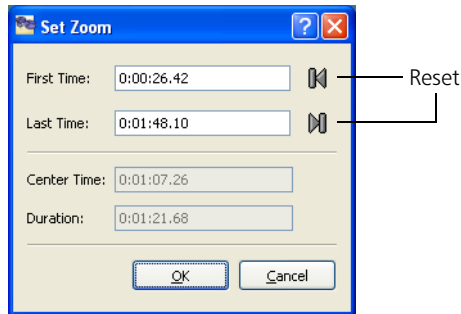


Figure 5-10. Set Zoom dialog box

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.

5.7.3. Resetting the Zoom Level

- To quickly zoom the timeline to its full length, double-click on the zoom bar between the zoom sliders (Figure 5-11).

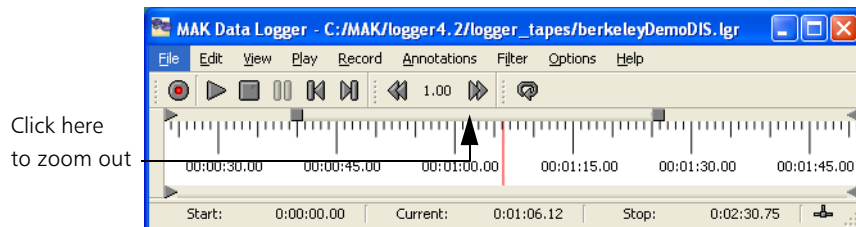


Figure 5-11. Zooming out

5.8. 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.
-

5.8.1. 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 ([Figure 5-2](#)).
- To jump to the end of a playback range, choose Play → Jump To End, or click the Jump to End button.

5.8.2. 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.

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.

5.8.3. 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 5-1](#)).

- 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.

5.9. 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 or PDU. In HLA, you can filter by class. During a Logger session, filtering settings apply to both file and playback. Settings do not persist between sessions.

5.9.1. 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 5-12). 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.

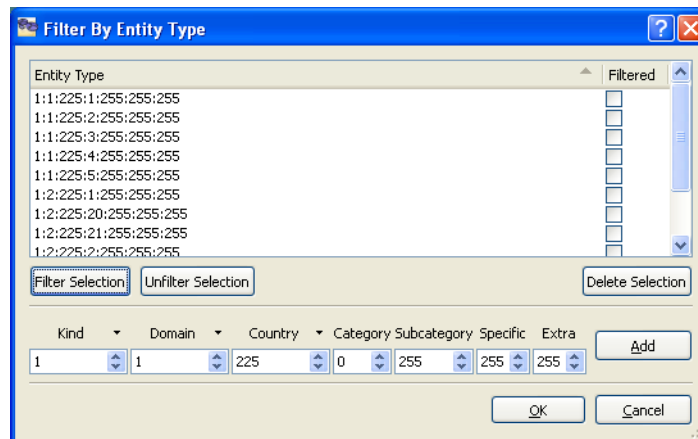


Figure 5-12. Filter By Entity Type dialog box

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.

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.

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>
```

5.9.2. 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 5-13).

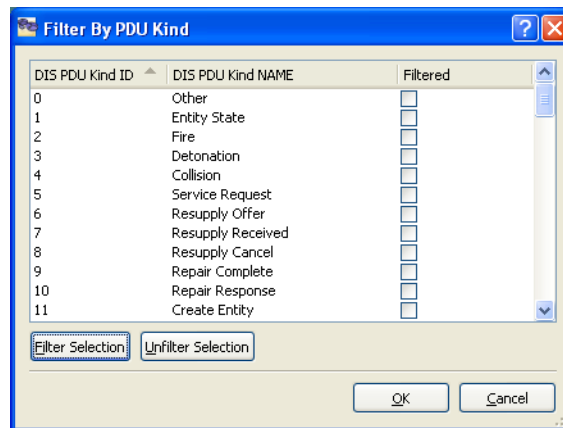


Figure 5-13. Filter By PDU Kind dialog box

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.

5.9.3. 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 do so, list them with the `customPdu` variable in *lgrConfig.xml*, 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>
```

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.

5.9.4. 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 5-14).

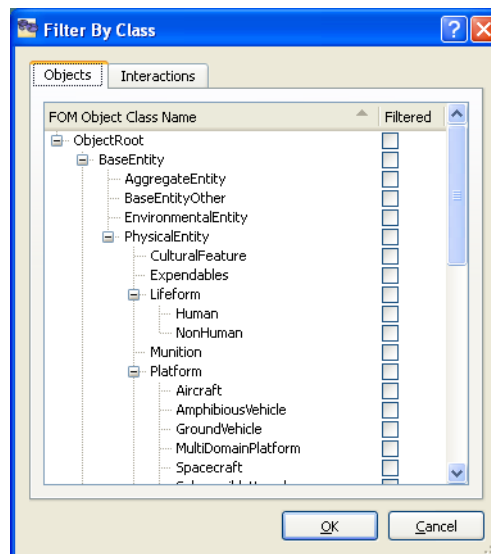


Figure 5-14. Filter By Class dialog box

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.

5.9.5. 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 5-15).



Figure 5-15. 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.

5.10. 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 5-1). When you record a simulation with the Data View enabled, the Logger updates the Data View as data is recorded (Figure 5-16).

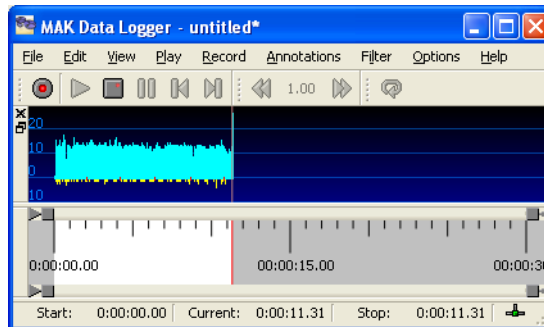


Figure 5-16. Data View in record mode

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.

5.11. 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 5-17).

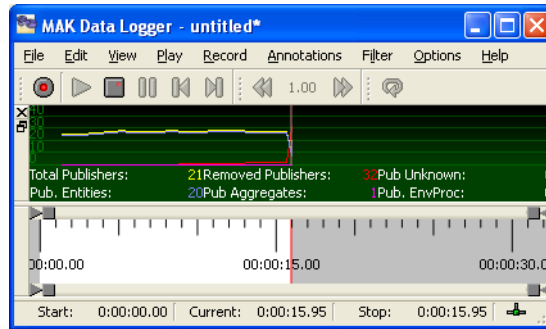
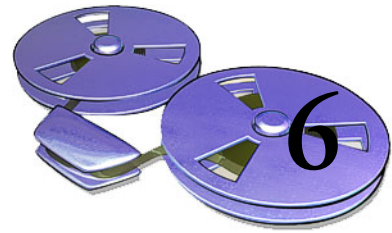


Figure 5-17. Object statistics

- 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.....	6-2
Adding an Annotation.....	6-2
Saving an Annotation File	6-4
Opening an Annotation File.....	6-4
Importing an Annotation File.....	6-5
Selecting an Annotation	6-5
Editing an Annotation.....	6-5
Deleting an Annotation.....	6-6
Moving the Timeline Cursor to the Start of an Annotation	6-6
Zooming the Timeline to an Annotation.....	6-7
Bounding Playback to an Annotation.....	6-8

6.1. 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.

6.1.1. Adding an Annotation

To annotate a Logger file:

1. Choose View → Annotations. Annotations, if any, are displayed in the Logger data display area (Figure 6-1).

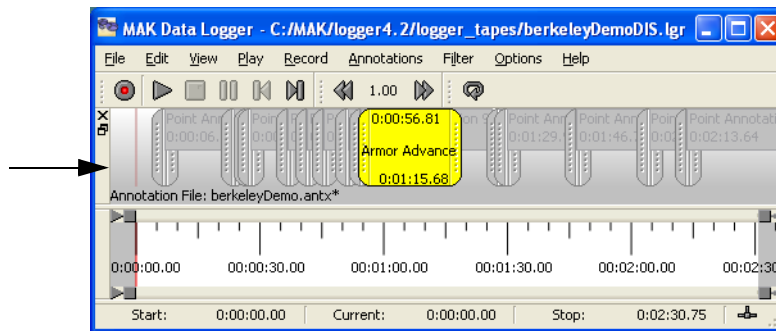


Figure 6-1. Annotations display

2. Choose Annotations → Create Annotation → *annotation_type*, where *annotation_type* type is Point Annotation, Range Annotation. The Point Annotation (or Range Annotation) dialog box opens.

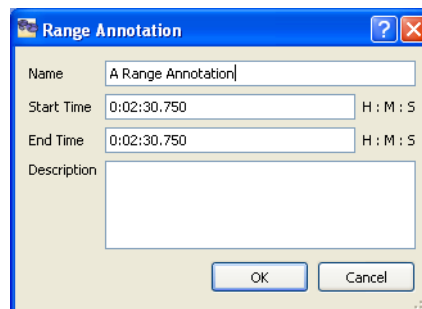


Figure 6-2. Range Annotation dialog box

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 6-1](#)).



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 Annotation → Instant Point Annotation (or Instant Range Annotation), or press \ (backslash) 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 Annotation or press **J** (right bracket). The appropriate timepoint of the annotation is reset to the time at which you issued the update command. The timepoint can be updated to an earlier or later time.

6.1.2. 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 6-2.
2. Choose Annotations → Save. The Save Annotation File dialog box opens.
3. Type a name for the annotation file.
4. Click Save.

6.1.3. 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.

6.1.4. 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.

6.1.5. Selecting an Annotation

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

6.1.6. 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 → Modify Annotation, 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 6-5.

6.1.7. Deleting an Annotation

To delete an annotation:

1. Select the annotation you want to delete.
2. Choose Annotations → Delete Annotation, 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.

6.1.8. 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.

6.1.9. Zooming the Timeline to an Annotation

You can zoom the timeline view to an annotation. For example, in [Figure 6-3](#), 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.

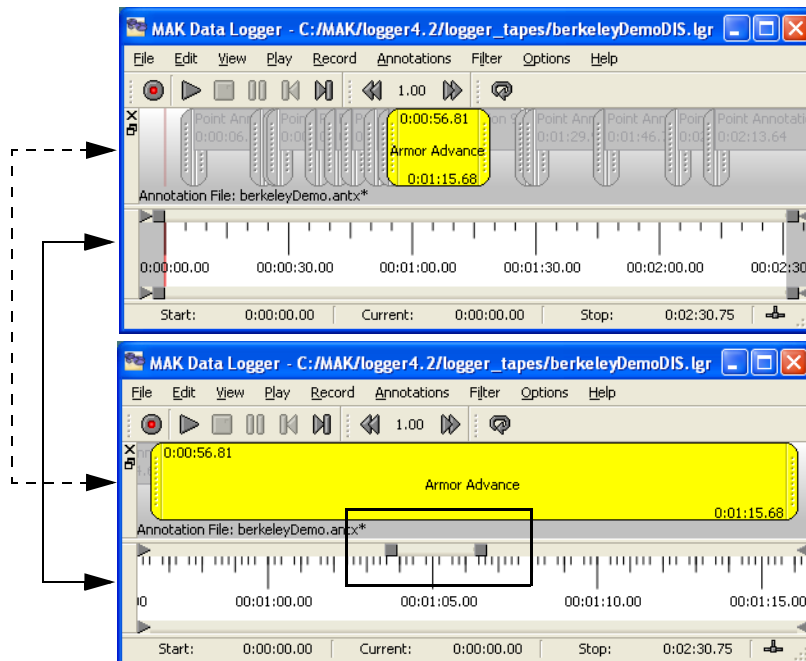


Figure 6-3. Zooming an annotation

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 6-3](#)).

6.1.10. Bounding Playback to an Annotation

You can set the playback segment to the range of an annotation. For example, in [Figure 6-4](#), 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 6-3](#), 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.)

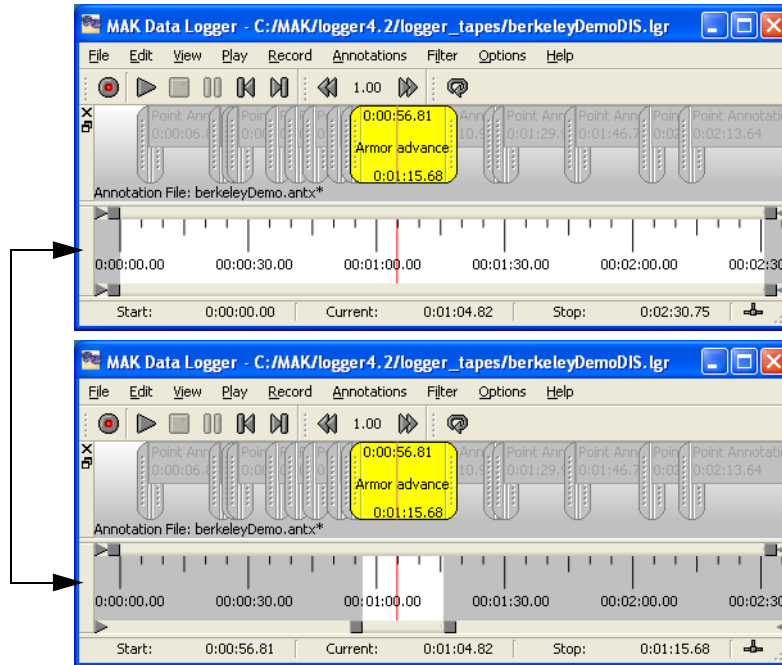


Figure 6-4. Bounding playback to an annotation

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 6-4](#)).



Editing Logger Files

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

Editing Files.....	7-2
Reverting to the Saved Version of a File.....	7-2
Clearing and Deleting Segments of a File.....	7-3
Merging Files	7-4
Concatenating Files.....	7-4
Trimming Files.....	7-5
Offsetting Files.....	7-5

7.1. 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.

7.1.1. 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.

7.2. 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.

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.

7.3. 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.

7.4. Concatenating Files

You can 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.

7.5. 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:

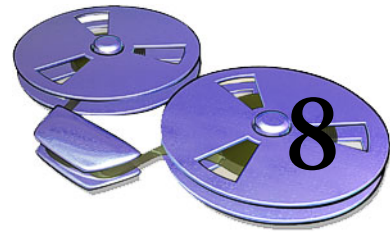
1. Open the file.
2. Choose Edit → Trim Start.
3. Save the edited file.

7.6. 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 data to various data formats.

Exporting Data	8-2
Exporting Data to Multiple Files.....	8-3
Exporting Data to a Database	8-4
Exporting DIS Data to a Database	8-5
Exporting HLA Data to a Database.....	8-6
The Export Log.....	8-6
Configuring an ODBC Data Source	8-6
Configuring a Data Source for Microsoft Access	8-7
Configuring a Data Source for MySQL.....	8-10

8.1. Exporting Data

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



You can export data while you are in Record mode. However this may adversely affect performance. You could experience dropped data or lose time fidelity.

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 section.

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.

8.2. Exporting Data to Multiple Files

If a Logger file is particularly large, you may want to break it up into a set of smaller files. Although you could do this manually by exporting sections of the file, the Logger can automatically export to multiple files.

i

You can also record to multiple files. For details, please see [“Recording to Multiple Files,”](#) on page 5-7.

To export a Logger file to multiple files:

1. Open the file.
2. Optionally, select a segment of the file using the timeline sliders.
3. Choose File → Export to → Multiple Files or File → Export Selection To → Multiple Files. The Export to Multiple Files dialog box opens ([Figure 8-1](#)).

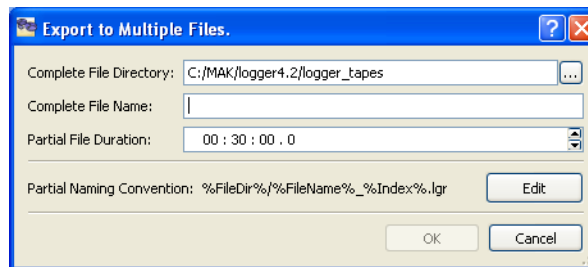


Figure 8-1. Export to Multiple Files dialog box

4. In the Complete File Directory box, type the complete pathname for the directory in which you want to store the complete Logger file or click the Browse button and select a directory.
5. In the Complete File Name box, type the name to which you want to save the complete file name.
6. Specify the length of each file in the Partial File Duration box.
7. Optionally, change the convention used to name the partial files. For details about changing the partial file naming convention, please see [“Specifying the Partial File Naming Convention,”](#) on page 5-8.
8. Click OK.

8.3. 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 sub table might be aggregateentity_silentaggregates.



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.

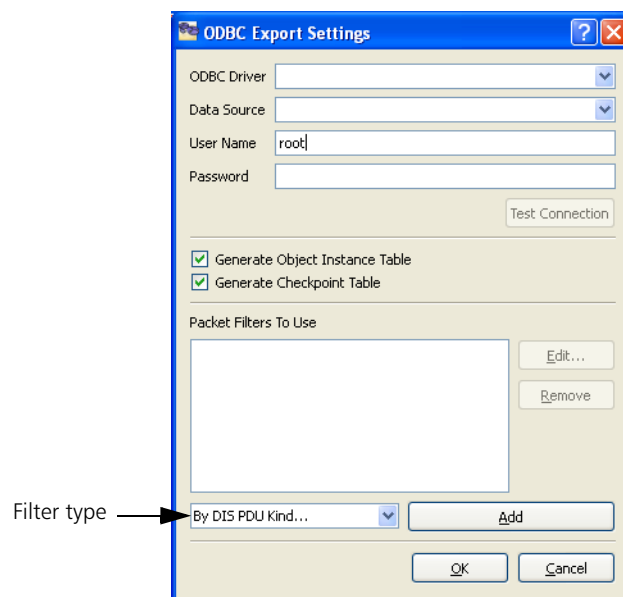


Figure 8-2. ODBC Export Settings dialog box

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. Optionally, check or clear the check boxes for the optional tables.
9. 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 5-22.

8.3.1. 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.

8.3.2. Exporting HLA Data to a Database

The Logger can export data from any FOM. 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.

8.3.3. 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.

8.3.4. 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 XP you create and configure data sources using the ODBC Data Source Administrator, as described in the following sections.

8.4. Configuring a Data Source for Microsoft Access

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

1. Open the Control Panel.
2. Double-click Administrative Tools. The Administrative Tools folder opens (Figure 8-3).

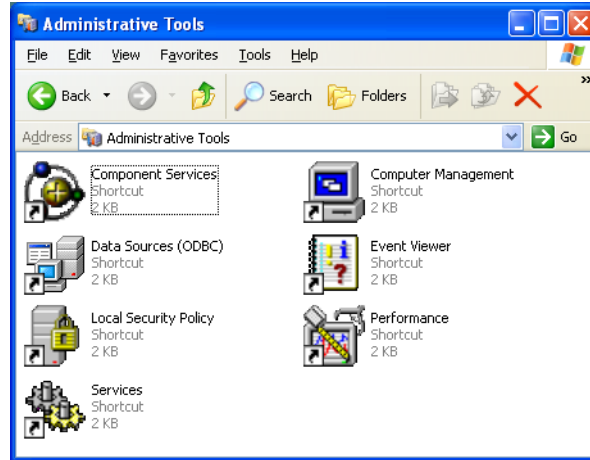


Figure 8-3. Administrative Tools applet window

3. In the Administrative Tools folder, double-click Data Sources (ODBC). The ODBC Data Source Administrator dialog box opens (Figure 8-4).

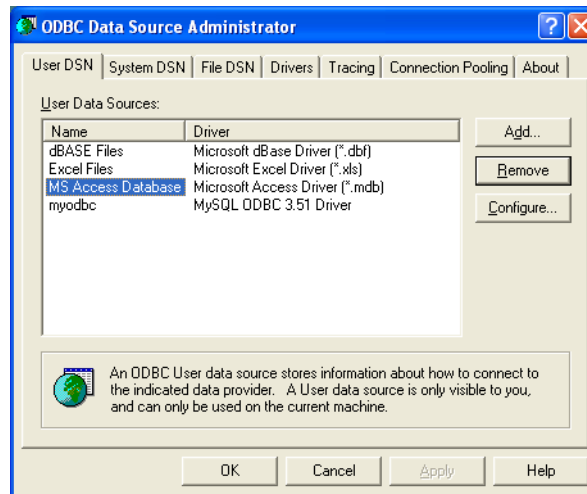


Figure 8-4. ODBC Data Sources Administrator

4. In the User Data Sources window on the User DSN tab or System DSN tab, select MS Access Database.

5. Click Configure. The ODBC Microsoft Access Setup dialog box opens (Figure 8-5).

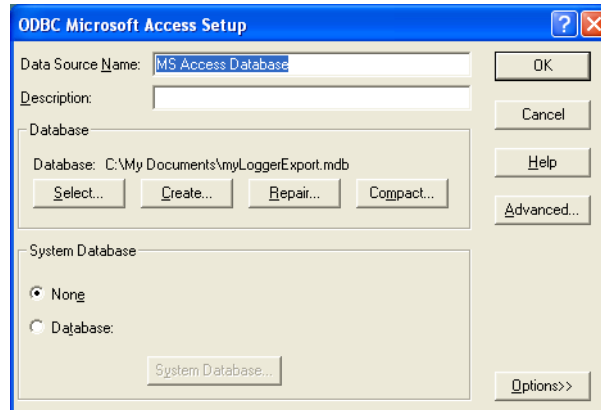


Figure 8-5. ODBC Microsoft Access Setup dialog box

6. In the Data Source Name box, type a name for the data source.
7. In the Database group box, click Create. The New Database dialog box opens (Figure 8-6).

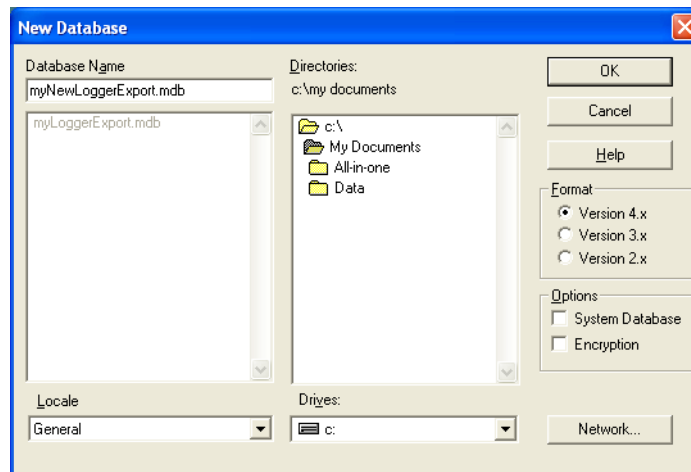


Figure 8-6. New Database dialog box

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

10. Click OK. An informational prompt advises you that the database has been created (Figure 8-7).

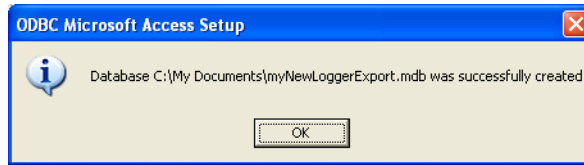


Figure 8-7. Information prompt

11. Click OK.
12. 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.

8.5. 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:

1. Open the Control Panel.
2. Double-click Administrative Tools. The Administrative Tools folder opens (Figure 8-3).
3. In the Administrative Tools folder, double-click Data Sources (ODBC). The ODBC Data Source Administrator dialog box opens (Figure 8-8).

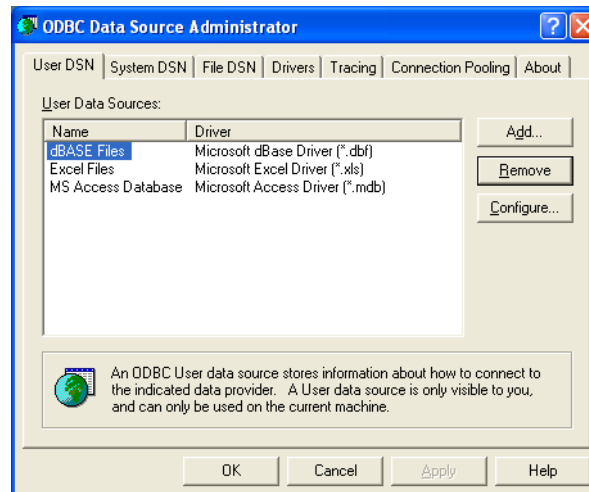


Figure 8-8. ODBC Data Source Administrator dialog box

4. Click Add. The Create New Data Source dialog box opens (Figure 8-9).

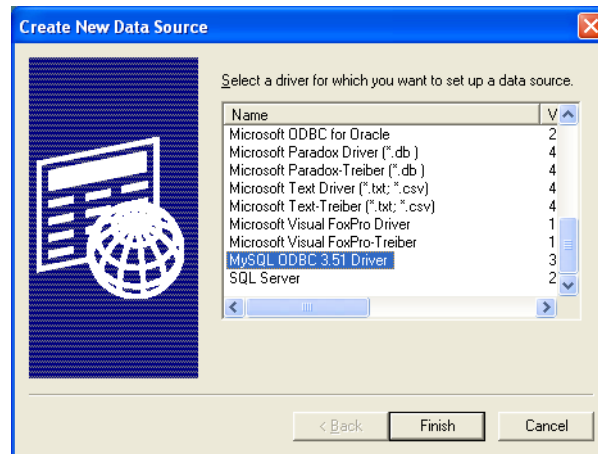


Figure 8-9. Create New Data Source dialog box

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

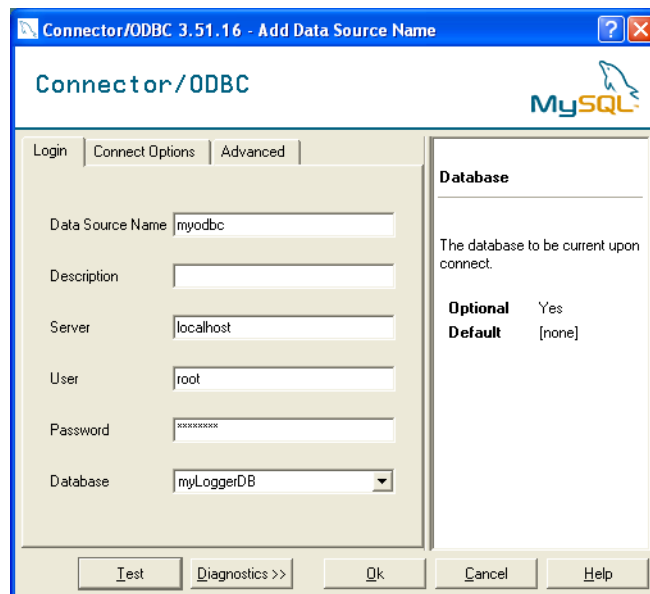


Figure 8-10. Connector ODBC *version* - Add Data Source Name dialog box

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 8-11).

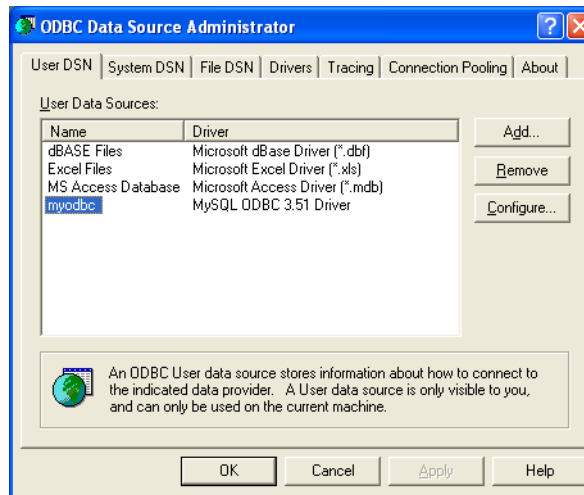


Figure 8-11. ODBC Data Source Administrator dialog box



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.....	9-2
Using Logger Commands in the Configuration File.....	9-2
Using Logger Commands in the Text Interface	9-2
Entering Commands	9-3
Displaying a List of Commands	9-3
Command Descriptions	9-4
Argument Types	9-4
System Domain Commands.....	9-5
Player Domain Commands	9-5
Recorder Domain Commands.....	9-7
Simulation Domain Commands.....	9-8
Editor Domain Commands.....	9-11
Command List.....	9-13

9.1. 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.

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.

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



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:

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\""
```

9.2. 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 4-25.

9.3. 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
```

9.3.1. 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
```

Commands must include all arguments and the arguments must be typed in the correct order. (For command descriptions, please see “[Command Descriptions](#),” on page 9-4.) 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.

9.3.2. 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**.

9.4. 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.

9.4.1. 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 9-1](#) describes the argument types used for commands.

Table 9-1: 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 in double quotes, separated by spaces. The individual strings cannot have spaces.
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).

9.4.2. System Domain Commands

Table 9-2 describes the commands in the System domain.

Table 9-2: 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.
SetNotifyLevel <i>vlLevel</i> (int) <i>lgrLevel</i> (int)	Sets the notification level of VR-Link and the Logger.
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.

9.4.3. Player Domain Commands

Table 9-3 describes the commands in the Player domain.

Table 9-3: Player domain commands (Sheet 1 of 2)

Command	Description
ClosePlaybackFile	Closes the current playback file.
OpenFile <i>filename</i> (string)	Opens the specified file for playback.
Pause	Pauses playback. This command has no effect if the Logger does not have a file open or is already paused.
Play	Starts playback. If the Logger is already playing this command has no effect.
PlayPause	Toggles between the paused and play state. If the Logger is stopped, it starts playback.
Restart	Starts playing from the beginning of the file.

Table 9-3: Player domain commands (Sheet 2 of 2)

Command	Description
<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 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)</code>	Sets the playback speed of the Logger.
<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.

9.4.4. Recorder Domain Commands

Table 9-4 describes the commands in the Recorder domain.

Table 9-4: Recorder domain commands (Sheet 1 of 2)

Command	Description
CancelRecording	Cancels the current recording. The Logger must be currently recording. The Logger is in an empty state, no file is loaded.
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. 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.
Record	Starts recording to a new file.
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. ! 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.
SetPauseOnRecordAs <i>record</i> (bool)	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.
SetStopCondition <i>condition</i> (int) <i>close</i> (bool)	Specifies the stop condition for recording as follows: ♦ 0 = no condition ♦ 1 = duration (enables SetStopDuration) ♦ 2 = filesize (enables SetStopSize). If the <i>close</i> parameter is True, the file is closed when recording is stopped.
SetStopDuration <i>duration</i> (time)	Stops recording at the specified time if SetStopCondition is set to 1.

Table 9-4: Recorder domain commands (Sheet 2 of 2)

Command	Description
<code>SetStopSize filesize(int64)</code>	Stops recording when the file reaches <code>filesize</code> size if <code>SetStopCondition</code> is set to 2.
<code>StopRecording closeAndOpenForPlayback(bool)</code>	Stops recording and ends. By default <code>closeAndOpenForPlayback</code> is true (1). If you set <code>closeAndOpenForPlayback</code> 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.

9.4.5. Simulation Domain Commands

[Table 9-5](#) describes 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 4, [Configuring the Logger](#).

Table 9-5: Simulation domain commands (Sheet 1 of 3)

Command	Description
Protocol-independent Commands	
<code>Connect</code>	Connects to the network simulation using the previously set parameters.
<code>Disconnect</code>	Disconnects from the network simulation.
<code>SetRemoteControlSettings process(bool) record(bool) allowDelete(bool)</code>	Specifies how the Logger responds to Logger Control messages. The <code>process</code> argument determines whether the Logger responds to Logger control messages. The <code>record</code> argument determines whether or not Logger control messages are included in the file. The <code>allowDelete</code> argument determines whether or not to respond to the delete file Logger control message.
<code>SetScaleAttributes scale(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 filter(bool) loId(int) hiId(int)</code>	Configures VR-Forces message filtering. The <code>filter</code> argument enables or disables filtering of VR-Forces messages. The <code>loId</code> argument specifies the lowest message ID to filter out. The <code>hiId</code> argument specifies the highest message ID to filter out.

Table 9-5: Simulation domain commands (Sheet 2 of 3)

Command	Description
HLA-specific Commands	
<code>LoadSubscriptionListFile filename(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 filename(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.
<code>SetConnectionParameters federateName(string) federationName(string) fedFile(string) fomVersion(double) fomMapper(string) fomMapperData(base64)</code>	Specifies HLA connection parameters: ♦ <i>federateName</i> – The name of the Logger federate ♦ <i>federationName</i> – The name of the federation execution ♦ <i>fedFile</i> – The name of the FED file (or FDD or XML file) ♦ <i>fomVersion</i> – The RPR FOM version being used ♦ <i>fomMapper</i> – The name of the FOM Mapper being used, if any ♦ <i>fomMapperData</i> – Initialization data for the FOM Mapper, if required.
<code>SetDisableTimeManagementTest disableTesting(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 by the user. 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 features are disabled.
<code>SetDynamicNamePrefix prefix (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>SetIgnoreAdvisories mode(bool)</code>	Specifies the Logger's response to HLA advisory messages. You must specify this command before the connect command. Default: false.
<code>SetPassiveSubscription mode(bool)</code>	Configures passive subscription for HLA. You must specify this command before the connect command. Default: false.

Table 9-5: Simulation domain commands (Sheet 3 of 3)

Command	Description
<code>SetSubscriptionLists</code> <i>objectClasses</i> (stringList) <i>interactionClasses</i> (stringList)	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 sufficiently long command line buffers 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</code> <i>timeConstrainedRecording</i> (bool) <i>timeConstrainedPlayback</i> (bool) <i>timeRegulatingRecording</i> (bool) <i>timeRegulatingPlayback</i> (bool) <i>lookahead</i> (double)	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.
DIS-specific Commands	
<code>AddCustomPDU</code> <i>pduName</i> (string) <i>pduKind</i> (string) <i>entityIdOffset</i> (string) <i>entityTypeOffset</i> (string)	Specifies a PDU to add to the Filter by PDU Kind dialog box.
<code>SetCommonEntityTypes</code> <i>entityType1</i> ... <i>entityTypen</i> (string)	Specifies a list of entity types to list in the Filter by Entity Type dialog box.
<code>SetConnectionParameters</code> <i>port</i> (int) <i>exerciseId</i> (int) <i>siteId</i> (int) <i>appNumber</i> (int) <i>useAsynchIO</i> (bool) <i>destAddr</i> (int) <i>sendBufferSize</i> (int) <i>recvBufferSize</i> (int) <i>multicastTtl</i> (int) <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>destAddr</i> – The destination address to send to ♦ <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>mcastAddresses</i> – A list of multicast addresses.
<code>SetHeartbeatRate</code> <i>rate</i> (double)	Specifies the DIS heartbeat rate.
<code>SetTimeoutRate</code> <i>rate</i> (double)	Specifies the DIS timeout rate.

9.4.6. Editor Domain Commands

Table 9-6 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 9-6: 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.

Table 9-6: Editor domain commands (Sheet 2 of 2)

Command	Description
<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 the 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.
<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>. This command works just like the ExportFile command except that is exports a segment of a file rather than the entire file.
<code>MergeInFile filename(string) offset(double)</code>	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 offset(double)</code>	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 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.

9.5. Command List

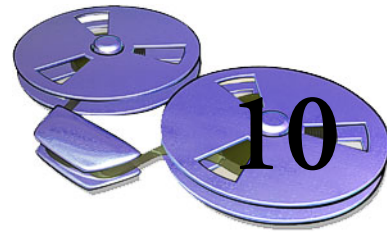
[Table 9-7](#) section lists all Logger commands and their domain:

Table 9-7: Logger commands

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

Table 9-7: Logger commands

Command	Domain
SaveSubscriptionListFile	Simulation
SetCheckpointParameters	Recorder
SetBackupOnSave	Player
SetCommonEntityTypes	Simulation
SetConnectionParameters	Simulation
SetCurrentTime	Player
SetDisableTimeManagementTest	Simulation
SetDynamicNamePrefix	Simulation
SetHeartbeatRate	Simulation
SetIgnoreAdvisories	Simulation
SetLoopAtEnd	Player
SetNotifyLevel	System
SetPassiveSubscription	Simulation
SetPauseOnRecordAs	Recorder
SetPlaybackSpeed	Player
SetPlayOnLoad	Player
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	10-3
The Logger Architecture	10-4
Drivers	10-4
Factories	10-5
Interfaces	10-5
The Logger Message Pipeline.....	10-6
The Simulation Manager.....	10-6
Embedding the Logger in an Application	10-7
Embedding Plug-ins in an Application	10-7
The playFile Example	10-8
Creating a Stand-Alone Logger Application	10-10
The Logger Plug-in API	10-10
Writing a Plug-In	10-11
Loading Plug-Ins	10-12
Logger API Coding Conventions	10-13
Namespaces	10-13
Templates	10-13
Shared Pointers.....	10-14
Virtual Constructors.....	10-14
Runtime Values	10-15
Thread Safety	10-16
Interface Signal Slots	10-16
Observer Pattern.....	10-17
Logger Libraries	10-17

Other Libraries and Plug-ins	10-18
Compiling and Linking with the Logger API	10-18
Compiling and Linking for UNIX	10-18
Compiling and Linking for Microsoft Windows	10-19

10.1. 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.

10.2. 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.

10.2.1. 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.

10.2.2. 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*.

10.2.3. 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.

10.2.4. 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.

10.2.5. 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.

10.3. 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/simple-FileActions*.

10.3.1. 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.

10.3.2. 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;
```

10.4. 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.

10.5. 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*.

10.5.1. 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;
} }
```

10.5.2. 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();
```

10.6. Logger API Coding Conventions

This section describes some of the coding mechanisms used in the Logger API.

10.6.1. 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( ) )
{
}
```

10.6.2. Templates

The Logger API uses C++ templates and template specializations. Therefore, to use the API, you should be familiar with their use.

10.6.3. 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 only there is one reference to aFoo.
    // aFoo will be deallocated on leaving the function
}
```

10.6.4. 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*.

10.6.5. 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.

10.6.6. 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;
    }
}
```

10.6.7. 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);
```

10.6.8. 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.

10.7. Logger Libraries

The minimum libraries necessary to build a functional Logger are the *lgrUtil*, *lgrCore*, and *lgrSim* libraries.

Table 10-1: 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.

10.7.1. 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*.

10.8. 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 UNIX and Windows. The Logger includes static and dynamic libraries. On Windows, it includes release (RT) and debug (RTd) 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.

10.8.1. Compiling and Linking for UNIX

To compile against the Logger API in UNIX, 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 or HLA 1516, 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
```

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.

10.8.2. 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 *RTd* versions of the libraries.

Linking for HLA

For HLA 1516, release configuration, link against:

```
lgrUtilRT.lib lgrCoreRT.lib lgrAppRT.lib appEnvRT.lib  
lgrGuiRT.lib lgrSimGuiHLA1516.lib (Logger libraries)  
vlHLA1516RT.lib vlRT.lib matrixRT.lib mtlRT.lib  
vlutilRT.lib (VR-Link libraries)  
librtl1516.lib libfedtime1516.lib (RTI libraries)
```

For HLA 13, release configuration, link against:

```
lgrUtilRT.lib lgrCoreRT.lib lgrAppRT.lib appEnvRT.lib  
lgrGuiRT.lib lgrSimGuiHLA13.lib (Logger libraries)  
vlHLA13RT.lib vlRT.lib matrixRT.lib mtlRT.lib vlutilRT.lib  
(VR-Link libraries)  
libRTI-NG.lib libFedtime.lib (RTI libraries)
```

Linking for DIS

For DIS, release configuration, link against:

```
lgrUtilRT.lib lgrCoreRT.lib lgrAppRT.lib appEnvRT.lib  
lgrGuiRT.lib lgrSimGuiDIS.lib (Logger libraries)  
vlDISRT.lib vlRT.lib matrixRT.lib mtlRT.lib vlutilRT.lib  
(VR-Link libraries)
```

Example makefiles and project files are in *./examples/mkwin32*.



Remote Control

You can control the Logger from a remote application using DIS or HLA update messages. Every command available in the Logger graphical user interface is available for remote control.

Remote Control Messages	11-2
Deleting Logger Files	11-3
Example of a Remote Control Application	11-3

11.1. Remote Control Messages

Logger Control PDU classes allow you to send messages to the Logger from an application based on VR-Link. There is a Logger Control PDU for each feature that can be activated remotely. All are derived from *DtLgrControlPdu*, (defined in *lcPdu.h*), and all are defined in header files that begin with “lc”.



In DIS we send PDUs. In HLA, we send interactions. For convenience, in this manual, we refer to PDUs and interactions collectively as messages.

The remote control classes are:

- ♦ *DtLgrDelFilePdu*
- ♦ *DtLgrEndPdu*
- ♦ *DtLgrEofActionPdu*
- ♦ *DtLgrHeartbeatPdu*
- ♦ *DtLgrPausePdu*
- ♦ *DtLgrPlayPdu*
- ♦ *DtLgrPlayActionPdu*
- ♦ *DtLgrPbFilePdu*
- ♦ *DtLgrQuitPdu*
- ♦ *DtLgrRecordActionPdu*
- ♦ *DtLgrRecFilePdu*
- ♦ *DtLgrRecordPdu*
- ♦ *DtLgrSetTimePdu*
- ♦ *DtLgrSkipTimePdu*
- ♦ *DtLgrStartPdu*
- ♦ *DtLgrStateFlagPdu*
- ♦ *DtLgrStopPdu*
- ♦ *DtLgrTimeScalePdu*
- ♦ *DtLgrTimeoutPdu*
- ♦ *DtLgrTotalStatePdu*.

HLA only

For HLA, *DtLgrControlInteraction*, defined in *hLcInter.h*, is a wrapper around *DtLgrControlPdu*.

11.2. Deleting Logger Files

The Logger Control PDU, *DtLgrDelFilePdu* instructs the Logger to delete a Logger file. The file to be deleted must be a Logger file of the correct protocol, that is, the DIS Logger can delete only DIS files and the HLA Logger can delete only HLA files.

If the Logger receives a *DtLgrDelFilePdu* that specifies the filename *foo.lgr*, it searches for *foo.lgr* in its working directory. If the PDU specifies a relative pathname, such as *../logger_tapes/foo.lgr*, the Logger searches for the file at that location relative to its working directory.

Diagnostic messages are sent to the *logger.txt* file.

To create a *DtLgrDelFilePdu*, use code similar to the following:

```
#include "lcDelFile.h"
...

DtLgrDelFilePdu deleteFile;
deleteFile.setFilename("logger_tape.lgr");
#ifdef HLA
DtLgrControlInteraction lcInter;
lcInter.setLcPdu(deleteFile);
exerciseConn.sendStamped(lcInter);
#else
exerciseConn.sendStamped(deleteFile);
#endif
```

11.3. Example of a Remote Control Application

The following example demonstrates programmatic remote control of the Logger. For the complete example, please see the *lgrControl* example in *.examples*.



The *lgrControl* example program is a VR-Link application and requires a VR-Link license to run. An unlicensed version of the program is available upon request.

```
#include "stdio.h"
#include "exerciseConn.h"
#include "ProcControl.h"
#include "EntityTypes.h"

#include "lcEnd.h"
#include "lcEofAction.h"
#include "lcHeartbeat.h"
#include "lcInter.h"
#include "lcPause.h"
#include "lcPdu.h"
#include "lcPlay.h"
#include "lcPlayAction.h"
#include "lcPlayFile.h"
#include "lcQuit.h"
#include "lcRecAction.h"
```

```
#include "lcRecFile.h"
#include "lcRecord.h"
#include "lcSetTime.h"
#include "lcSkipTime.h"
#include "lcStart.h"
#include "lcStop.h"
#include "lcTimeScale.h"
#include "lcTimeout.h"
#include "lcDelFile.h"

char * helpstr =
    " PDUTYPE OPT_ARG\n"
    "0 PbFile filename\n"
    "1 RecFile filename\n"
    "2 Quit \n"
    "3 Rewind \n"
    "4 End \n"
    "5 Play \n"
    "6 Pause \n"
    "7 Stop \n"
    "8 Record \n"
    "11 TimeScale float\n"
    "12 PlayAction {PLAY=0, STOP=3}\n"
    "13 RecordAction {RECORD=2, STOP=3}\n"
    "14 EofAction {PAUSE=1, STOP=3, LOOP=5}\n"
    "15 HeartBeat float\n"
    "16 Timeout float\n"
    "17 Delete filename\n"
    "18 SetTime float\n"
    "19 SkipTime float\n"
    "-1 Quit this application\n"
;

sendPdu(DtExerciseConn& exConn, DtLgrControlPdu* pdu,
        DtGlobalObjectDesignator& target, DtGlobalObjectDesignator& sender)
{
    if (pdu)
    {
        pdu->setReceiverId(target);
        pdu->setSenderId(sender);
        #if DtHLA
            DtLgrControlInteraction inter;
            inter.setLcPdu(*pdu);
            exConn.send(inter);
        #else
            exConn.sendStamped(*pdu);
        #endif
    }
}

//
// Application
//

main()
{
    #if DtHLA
        DtExerciseConn exConn("VR-Link", "Remote", new DtRprFomMapper(1.0));
    #else
        DtExerciseConn exConn(3000, 1, 39, 16);
    #endif
}
```

```
#endif

// Identify the receiving (remote) Logger
DtGlobalObjectDesignator target("9:8:7");

// Optionally identify yourself
DtGlobalObjectDesignator myId("3:4:5");

// Initialize simulation time
DtTimeInit();

char cmdline[128];
char optionalArg[128];
int cmd = 0;
while (cmd >= 0)
{
    // Print usage instructions and prompt for input
    printf("%s\n> ",helpstr);
    gets (cmdline);

    // Scan for input
    cmd = 0 ; optionalArg[0] = '\0';
    sscanf(cmdline, "%d%s",&cmd,optionalArg);

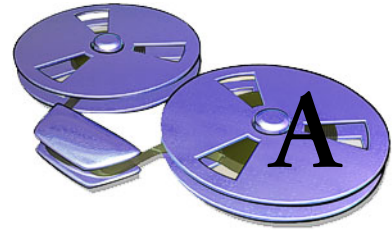
    // Construct and send desired PDU
    switch(cmd)
    {
    case DtLgrPbFile:
    {
        DtLgrPbFilePdu pdu;
        pdu.setFilename(optionalArg);
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    case DtLgrRecFile:
    {
        DtLgrRecFilePdu pdu;
        pdu.setFilename(optionalArg);
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    case DtLgrRecord:
    {
        // Start recorded traffic from network (during record mode)
        DtLgrRecordPdu pdu;
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    case DtLgrStop:
    {
        // Stop sending recorded traffic onto network
        // (during playback)
        DtLgrStopPdu pdu;
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    case DtLgrQuit:
    {
        DtLgrQuitPdu pdu;
        sendPdu(exConn, &pdu, target, myId);
    }
    }
```

```
        break;
    }
    #if !DtHLA
    case DtLgrHeartbeat:
    {
        DtLgrHeartbeatPdu pdu;
        pdu.setHeartbeat(atof(optionalArg));
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    case DtLgrTimeout:
    {
        DtLgrTimeoutPdu pdu;
        pdu.setTimeout(atof(optionalArg));
        sendPdu(exConn, &pdu, target, myId);
        break;
    }
    #endif
    }
}
```



A program can use wildcards to identify the Loggers it sends commands to.
For example:

```
DtObjectId target("255:255:7");
```



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 UNIX `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

[Application Management Object \(AMO\)](#).

Application Management Object (AMO)

An 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.

In the MÄK Stealth, when the eyepoint moves with respect to an entity, we describe it as movement in a body coordinate frame.

channel

A channel is a particular view when you are displaying multiple views of a simulation.

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 eyepoint.

clipping planes

The range of distances from the eyepoint (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.

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.

Defense Modeling and Simulation Office

The executive secretariat for the Executive Council on Modeling and Simulation (EXCIMS), which provides a full-time focal point for information concerning Department of Defense modeling and simulation (M&S) activities. Currently the DMSO 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](#).

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

[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

Federation Execution Data file. The Federation Execution Data is the subset of [FOM](#) data needed and read by the [RTI](#). VR-Link applications also read the 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

Field of view controls the perspective of the eyepoint.

A wide field of view creates an effect like that of a wide-angle camera lens. Objects appear smaller and farther away from the eyepoint, since the eyepoint 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 eyepoint, 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

[Federation Object Model](#).

frame rate

The rate at which the application displays updated images.

ground clamping

A process by which the MÄK Stealth 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.

heads-up display

A set of indicators and readouts superimposed onto a graphics display. In the MÄK Stealth, it displays the state of the eyepoint and the locations of other entities. Also called the 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

[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

A set of DIS PDUs or HLA interactions that let you control the MÄK Logger remotely.

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.

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.

lookahead

In HLA time-managed federations, the lookahead value is the smallest amount by which a time-regulating federate can advance time. Per the HLA specification, it is "the lowest value of time stamps that can be sent in TSO messages by the joined federate. Specifically, a time-regulating joined federate shall not send a TSO message that contains a time stamp less than its current logical time plus its current lookahead." (IEEE 1516)

LROM

[Logical Range Object Model](#).

MÄK Technologies Lisp

An adaptation of the Lisp language used in configuration files for MÄK Technologies 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.

MTL

[MÄK Technologies Lisp](#).

Network Naming Service

Acts as a registry for TENA Execution Managers.

NNS

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.

PDU

[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.

RTI Initialization Data (RID)

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

[Realtime Platform Reference FOM](#).

RTI

[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

[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

[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

[Test and Training Enabling Architecture.](#)

TENA Middleware

“The software that links range resources together into a logical range. The “TENA Middleware” is the software component that is in between (that is, in the middle of) range resources during a logical range execution.” –www.tena-sda.org

terrain following

Causes the Stealth eyepoint to maintain a constant distance above the terrain surface. The eyepoint'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.

Time Management

Time management is used in federations to coordinate the advancement of individual federates along a unified time axis. Time management is beneficial for federations that must maintain a causal ordering (that is, timestamp order) to the messages exchanged between federates. Because of the overhead involved in synchronization, time management is not usually used for real-time simulations, which can tolerate out of order messages, but have strict latency and throughput requirements.

time-constrained federate

An HLA federate whose ability time advance time is controlled by one or more time-regulating federates.

time-regulating federate

An HLA federate that controls the ability of time-constrained federates to advance time.

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. When the eyepoint moves with respect to the terrain, we describe it as movement in a topographic coordinate frame.

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. When in UTM mode (the default), the Stealth uses an offset UTM coordinate system, one in which coordinates are expressed with respect to an origin located at the latitude and longitude specified in a configuration file. Typically, this will be the location of the origin of your terrain database.

UTM coordinates

[Universal Transverse Mercator](#) coordinates.

view control messages

A set of programmatic messages that let you control the MÄK Stealth or Logger remotely.

view floor

In the Stealth, a minimum height above the terrain in Absolute or Track mode, below which the eyepoint is not allowed to go. The view floor is measured relative to the terrain surface.



Index

- command-line option 4-29
- appNumber command-line option 4-30
- configFile command-line option 4-25, 4-29
- destAddrString command-line option 4-30
- disPort command-line option 4-30
- disVersion command-line option 4-30
- execName command-line option 4-29
- exerciseId command-line option 4-30
- federateName command-line option 4-29
- fomMapperInitData command-line option 4-13, 4-29
- fomMapperLib command-line option 4-29
- ignore_rest command-line option 4-29
- logFileName command-line option 4-29
- mcastAddresses command-line option 4-30
- mcastTtl command-line option 4-22, 4-30
- messageLevel command-line option 4-29
- notifyLevel command-line option 4-29
- receiveBufferSize command-line option 4-21
- recvBufferSize command-line option 4-30
- rprFomVersion command-line option 4-11, 4-29
- runCommand command-line option 4-29
- sendBufferSize command-line option 4-21, 4-30
- showConsole command-line option 4-29
- siteId command-line option 4-30
- version command-line option 4-29

A

- A command-line option 4-21, 4-30
- a command-line option 4-20, 4-30
- acceleration scaling 3-7
- Access, configuring data source for 8-7
- addCommandHandler() function 10-4

AddCustomPDU command 9-10

adding

- annotation 6-2
- command to *lgrConfig.xml* 4-27
- instant annotation 6-3
- new license file 2-10
- variable to *lgrConfig.xml* 4-27

address, multicast 4-30

advisory message

- configuring 9-9
- ignoring 4-13

annotating, file 6-2

annotation 6-2

- bounding playback to 6-8
- converting type 6-6
- creating 6-2
- deleting 6-6
- editing 6-5
- importing annotation file 6-5
- instant, adding 6-3
- loading automatically 5-11
- opening annotation file 6-4
- saving file 6-4
- selecting 6-5
- updating 6-4
- zooming timeline to 6-7

API

- Logger 10-3
- plug-in 10-10

AppendFileToEnd command 9-11

appending

- file 9-12
- files 9-11

appending file 7-4

appEnv library 10-18

- application
 - compiling and linking 10-18
 - embedding Logger in 10-7, 10-10
 - embedding plug-ins in 10-7
- application number 4-20, 9-10
 - specifying 4-30
- architecture, Logger API 10-4
- argument type 9-4
- asynchronous I/O 9-10
 - configuring 4-20
- attribute
 - annotation 6-5
 - connection 4-5
- attribute scaling, enabling or disabling 9-8
- automatic
 - playback 5-11
 - recording 5-7

B

- backup, enabling or disabling 9-6
- backup file 5-15, 9-6
- bak file 5-15, 9-6
- beginning, moving to 5-18
- berkeleyDemo 1-4
- broadcast address 4-21, A-2
- buffer, send and receive 4-21
- button bar 5-4

C

- c command-line option 4-29
- canceling, recording 9-7
- CancelRecording command 9-7
- changing, connection settings 4-5
- checkpoint policy 4-6, 9-7
- class
 - DtAnyValue* 10-15
 - DtCommand* 10-5
 - DtCommandDefinition* 10-5
 - DtCommandSP* 10-14
 - DtDisLogger* 10-7
 - DtHlaLogger* 10-7
 - DtLgrAppGuiInterface* 10-17
 - DtLgrBaseSimManager* 10-6
 - DtLgrCommandHandler* 10-4
 - DtLgrControlInteraction* 11-2
 - DtLgrControlPdu* 11-2
 - DtLgrDelFilePdu* 11-3
 - DtLgrDisSimulationManager* 10-6

- DtLgrFullInterface* 10-5
- DtLgrHlaSimulationManager* 10-6
- DtLgrInterface* 10-3
- DtLgrPlaybackCommand Factory* 10-5
- DtLgrPlaybackDriver* 10-4
- DtLgrPlaybackInterface* 10-5
- DtLgrProtocolRegistration* 10-6
- DtLgrSimInput* 10-6
- DtLgrSimOutput* 10-6
- DtLgrSystemDriver* 10-4
- DtLogger* 10-3, 10-4, 10-6, 10-7, 10-10, 10-11, 10-12
- DtLoggerApplication* 10-10, 10-11
- DtMessageInput* 10-6
- DtMessageOutput* 10-6
- DtMessageState* 10-6
- DtMutex* 10-16
- DtObserveConnection* 10-17
- DtObserveEvent* 10-17
- DtObserver* 10-17
- DtProtocolRegistration* 10-17
- DtResponse* 10-5, 10-14
- DtSimInput* 10-6
- DtString* 10-15
- DtThreadSafe* 10-16
- DtValue* 10-15
- DtValueSignal* 10-16
- factory 10-5
- filtering by (HLA) 5-21
- interface 10-5
- remote control 11-2
- classes, subscribing to 4-15
- clearing
 - file segment 7-3
 - packets from file 9-11
- ClearSection command 9-11
- ClosePlaybackFile command 9-5
- CloseRecordFile command 9-7
- closing
 - file 5-10, 9-5
 - recording 9-7
- command 4-3
 - AddCustomPDU 9-10
 - adding to *lgrConfig.xml* 4-27
 - AppendFileToEnd 9-11
 - CancelRecording 9-7
 - ClearSection 9-11
 - ClosePlaybackFile 9-5
 - CloseRecordFile 9-7
 - Connect 4-5, 9-8

- ConvertFile 9-11
- CutSection 9-11
- defining, creating, and decoding 10-5
- DeleteFile 9-5
- Disconnect 4-4, 9-8
- Editor domain 9-11
- Exit 9-5
- ExportFile 9-12
- ExportSection 9-12
- Help 9-5
- HLA simulation domain 9-9
- in configuration file 4-26
- list of 9-13
- LoadPlugin 9-5
- LoadSubscriptionListFile 4-17, 9-9
- MergeInFile 9-12
- NewFile 9-7
- OffsetTime 9-12
- OpenFile 9-5
- Pause 9-5
- PauseRecording 9-7
- Play 9-5
- Player domain 9-5
- PlayPause 9-5
- PrefixFileToStart 9-12
- Record 9-7
- Recorder domain 9-7
- Restart 9-5
- RevertFile 9-6
- running at startup 4-29
- SaveFile 9-6
- SaveFileAs 9-6
- SaveSubscriptionListFile 4-16, 9-9
- sending 10-5
- SetBackupOnSave 5-15, 9-6
- SetCheckpointParameters 4-7, 9-7
- SetConnectionParameters 4-9, 4-18, 9-8, 9-9, 9-10
- SetCurrentTime 9-6
- SetDisableTimeManagementTest 9-9
- SetDynamicNamePrefix 3-3, 9-9
- SetHeartbeatRate 4-23, 9-10
- SetIgnoreAdvisories 4-13, 9-9
- SetLoopAtEnd 9-6
- SetNotifyLevel 4-7, 9-5
- SetPassiveSubscription 4-17, 9-9
- SetPauseOnRecordAs 9-7
- SetPlaybackSpeed 9-6
- SetPlayOnLoad 9-6
- SetRemoteControlSettings 9-8
- SetScaleAttributes 9-8
- SetStartStopTimes 9-6
- SetStopCondition 9-7
- SetStopDuration 9-7
- SetStopSize 9-8
- SetSubscriptionLists 4-16, 4-28, 9-10
- SetTempDir 9-5
- SetTimeManagement 4-14, 9-10
- SetTimeoutRate 4-22, 9-10
- SetVRFFiltering 9-8
- Simulation domain 9-8
- StopPlayback 9-6
- StopRecording 9-8
- syntax in *lgrConfig.xml* 4-26
- System domain 9-5
- TrimStart 9-12
- Unpause 9-6
- command line, installation 2-4
- command-line interface 9-2
 - domains 9-4
- command-line option 4-3
 - 4-29
 - A 4-21, 4-30
 - a 4-20, 4-30
 - appNumber 4-30
 - c 4-29
 - configFile 4-25, 4-29
 - destAddrString 4-30
 - disPort 4-30
 - disVersion 4-30
 - e 4-19, 4-30
 - execName 4-29
 - exerciseId 4-30
 - F 4-11, 4-29
 - f 4-12, 4-29
 - federateName 4-29
 - filename 4-29
 - fomMapperInitData 4-13, 4-29
 - fomMapperLib 4-29
 - h 4-29
 - ignore_rest 4-29
 - l 4-8, 4-29
 - logFileName 4-29
 - m 4-7, 4-29
 - mcastAddresses 4-30
 - mcastTtl 4-22, 4-30
 - messageLevel 4-29
 - N 4-10, 4-29
 - n 4-7, 4-29
 - notifyLevel 4-29

- P 4-19, 4-30
- r 4-29
- receiveBufferSize 4-21
- recvBufferSize 4-30
- rprFomVersion 4-11, 4-29
- runCommand 4-29
- S 4-24, 4-30
- s 4-19, 4-30
- sendBufferSize 4-21, 4-30
- showConsole 4-29
- siteId 4-30
- table of 4-29
- v 4-29
- version 4-29
- x 4-10, 4-29
- commonEntityTypes** parameter 5-20
- compiling
 - API code 10-18
 - application 10-18
- component files 2-5
- concatenating, files 7-4, 9-12
- configuration file 4-25
 - adding string list 4-28
 - RTI 2-16
 - specifying 4-29
- configuration parameter 4-3
 - useAsyncIO 4-20
- configuring
 - advisory messages 9-9
 - application number 4-20
 - asynchronous I/O 4-20
 - data source for Microsoft Access 8-7
 - data source for MySQL 8-10
 - DIS heartbeat 9-10
 - DIS timeout 9-10
 - FED file 4-29
 - filtering 9-8
 - HLA connection 9-9
 - indexing 4-6
 - list of common entity types 5-20
 - Logger 4-3
 - Logger for HLA 4-9
 - passive subscription 9-9
 - Record As 9-7
 - recording 9-7
 - response to Logger Control messages 9-8
 - site ID 4-19
 - system for MÄK RTI 2-16
 - Time Management 9-10
- Connect command 4-5, 9-8

- connecting 9-8
 - to exercise 4-5
 - to network 4-4
- connection
 - changing settings 4-5
 - configuring DIS 9-10
 - configuring HLA 9-9
 - settings 4-4
 - status 4-4
- console window 9-2
 - displaying 4-29
- control, Logger 5-4
- ConvertFile command 9-11
- converting, annotation type 6-6
- create() function 10-5, 10-15
- creating
 - annotation 6-2
 - command 10-5
 - subscription control list 4-15
- cursor
 - setting to time 9-6
 - timeline 5-18
- custom PDU, filtering 5-21
- customPdus** parameter 5-21
- CutSection command 9-11
- cutting
 - file segment 7-3
 - packets 9-11

D

- daemon, maklmgrd 2-11
- data
 - exporting 8-2
 - exporting to database 8-4
- data source
 - Access 8-7
 - configuring for ODBC 8-6
 - MySQL 8-10
- data type, HLA 8-6
- Data View 5-23
- database
 - configuring ODBC data source 8-6
 - exporting data to 8-4
 - exporting HLA data to 8-6
 - schema 8-4
- dead reckoning
 - acceleration scaling 3-7
 - velocity scaling 3-7
- dead space, removing 9-12

- decode() function 10-5
 - decoding, command 10-5
 - defining, command 10-5
 - DeleteFile command 9-5
 - deleting
 - annotation 6-6
 - file segment 7-3
 - Logger file 11-3
 - packets 9-11
 - destination address 4-21, 9-10
 - dialog box
 - Set Start and End Time 5-13
 - Set Zoom 5-17
 - directory
 - .bin* 2-5
 - .doc* 2-5
 - .logger_tapes* 2-5
 - DIS 1-6
 - application number 4-20
 - asynchronous I/O 4-20
 - configuring connection 9-10
 - exercise ID 3-3, 4-19
 - export to database 8-5
 - filtering entity types 9-10
 - filtering PDUs 9-10
 - heartbeat 4-23, 9-10
 - port, specifying 4-30, A-2
 - protocol version 4-30
 - site ID 4-19
 - timeout 4-22, 9-10
 - DIS port, Logger default A-2
 - disabling
 - attribute scaling 9-8
 - backup 9-6
 - looping playback 9-6
 - Pause on Record As 5-7
 - Play On Load 5-11
 - Time Management 9-9
 - Disconnect command 4-4, 9-8
 - disconnecting 9-8
 - from exercise 4-4
 - displaying
 - console window 4-29
 - object statistics 5-24
 - plug-in view 5-5
 - toolbar 5-5
 - version information 4-29
 - docking, toolbars and plug-in views 5-5
 - domain 4-27
 - associated with command 9-13
 - command 9-4
 - dragging, timeline cursor 5-18
 - driver, in Logger API 10-4
 - DtAnyValue* class 10-15
 - DtCommand* class 10-5
 - DtCommandDefinition* class 10-5
 - DtCommandSP* class 10-14
 - DtDisLogger* class 10-7
 - DtHlaLogger* class 10-7
 - DtLgrAppGuiInterface* class 10-17
 - DtLgrBaseSimManager* class 10-6
 - DtLgrCommandHandler* class 10-4
 - DtLgrControlInteraction* class 11-2
 - DtLgrControlPdu* class 11-2
 - DtLgrDelFilePdu* class 11-3
 - DtLgrDisSimulationManager* class 10-6
 - DtLgrFullInterface* class 10-5
 - DtLgrHlaSimulationManager* class 10-6
 - DtLgrInterface* class 10-3
 - DtLgrPlaybackCommand Factory* class 10-5
 - DtLgrPlaybackDriver* class 10-4
 - DtLgrPlaybackInterface* class 10-5
 - DtLgrProtocolRegistration* class 10-6
 - DtLgrSimInput* class 10-6
 - DtLgrSimOutput* class 10-6
 - DtLgrSystemDriver* class 10-4
 - DtLogger* class 10-3, 10-4, 10-6, 10-7, 10-10, 10-11, 10-12
 - DtLoggerApplication* class 10-10, 10-11
 - DtMessageInput* class 10-6
 - DtMessageOutput* class 10-6
 - DtMessageState* class 10-6
 - DtMutex* class 10-16
 - DtObserveConnection* class 10-17
 - DtObserveEvent* class 10-17
 - DtObserver* class 10-17
 - DtProtocolRegistration* class 10-17
 - DtResponse* class 10-5, 10-14
 - DtSimInput* class 10-6
 - DtString* class 10-15
 - DtThreadSafe* class 10-16
 - DtValue* class 10-15
 - DtValueSignal* class 10-16
- E**
- e command-line option 4-19, 4-30
 - editing
 - annotation 6-5
 - file 7-2

- lgrConfig.xml* 4-25
- point annotation 6-5
- range annotation 6-5
- undoing changes 7-2
- Editor domain, commands 9-11
- embarkdemo 1-4
- embedding
 - Logger in application 10-7, 10-10
 - plug-ins in application 10-7
- empty space, trimming 7-5
- EnableLoggerAppPlugin() function 10-11
- EnableLoggerPlugin() function 10-11
- enabling
 - attribute scaling 9-8
 - backup 9-6
 - looping playback 9-6
 - Pause on Record As 5-7
 - Play On Load 5-11
 - Time Management 9-9
- end
 - jumping to range annotation 6-6
 - moving to 5-18
- end time
 - setting dynamically 6-4
 - setting for instant range annotation 6-3
- ending, recording 5-9
- end-of-file, setting 5-13
- entity
 - common types 5-20
 - filtering by type 9-10
 - maintenance 3-7
- entity type
 - filtering 9-10
 - filtering by 5-19
 - removing from filtering list 5-19
- environment variable
 - MAKLMGRD_LICENSE_FILE 2-9, 2-11
 - RTI_CONFIG 2-16
- example
 - API 10-7
 - plug-in 10-10
 - programs 1-5
 - remote control 11-3
- exercise
 - connecting to 4-5
 - connection 4-4
 - configuring DIS 9-10
 - disconnecting from 4-4
 - ID 3-3, 4-19, 9-10
 - playback 3-3

- recording 5-6
- Exit command 9-5
- expanding, playback segment 5-13
- export, log file 8-6
- ExportFile command 9-12
- exporting
 - data, naming convention 5-8
 - data to database 8-4
 - file 8-2, 9-12
 - multiple files 8-3
- ExportSection command 9-12

F

- F command-line option 4-11, 4-29
- f command-line option 4-12, 4-29
- factory 10-5
- FDD file 2-16
- features 1-2
- FED file 1-6, 2-16
 - changing 3-8
 - specifying 4-11, 4-12, 4-29
- federate, specifying Logger federate name 4-10
- federation execution 1-6
 - name 4-10, 4-29
- file
 - annotating 6-2
 - appending 7-4, 9-11, 9-12
 - backup 5-15, 9-6
 - clearing packets from 9-11
 - clearing segment 7-3
 - closing 5-10, 9-5
 - configuration 4-25
 - deleting packet from 9-11
 - deleting segment 7-3
 - editing 7-2
 - exporting 8-2, 9-12
 - exporting to database 8-4
 - exporting to multiple 8-3
 - FED 2-16, 4-12
 - jumping to beginning 5-18
 - jumping to end 5-18
 - lgrConfig.xml* 4-3
 - license 2-9
 - loading 5-10
 - merging 7-4, 9-12
 - offsetting 7-5
 - opening 9-5
 - pausing 5-11
 - playing 5-11

- recording to multiple 5-7
- reverting to saved version 7-2, 9-6
- RID 2-16
- rti.mtl* 2-16
- saving 5-15, 9-6
 - segment to new file 5-15
 - to new name 5-15
- trimming 9-12
- trimming empty space 7-5
- upgrading format 5-16
- VR-Link.fed* 4-11
- filename, *logger.log* 4-8
- filename command-line option 4-29
- filter, message 1-3
- Filter By Class dialog box 5-21
- Filter By Entity Type dialog box 5-19
- Filter By PDU Kind dialog box 5-20
- filtering
 - by entity type 9-10
 - by PDU kind 9-10
 - classes (HLA) 5-21
 - configuring 9-8
 - configuring common entity types 5-20
 - custom PDUs 5-21
 - entity type 5-19
 - entity types 9-10
 - input and output 5-19
 - PDU kind 5-20
 - PDUs 9-10
 - VR-Forces messages 5-22
- FLEXlm
 - directory location 2-7
 - installing 2-7
 - license file 2-9
 - license manager 2-6
 - license server name 2-8
 - multiple licenses 2-10
 - running as service 2-13
 - specifying port 2-11
 - troubleshooting 2-14
- floating, license 2-6
- FOM 2-16
 - database export 8-6
 - flexibility 3-8
 - supported 1-6
- FOM Mapper 1-6, 4-12, 4-29
 - initialization data 4-29
 - specifying 4-12
 - specifying at startup 4-11
- format, upgrading 5-16

- frequency, packets for indexing 4-6
- function
 - `addCommandHandler()` 10-4
 - `create()` 10-5, 10-15
 - `decode()` 10-5
 - `EnableLoggerAppPlugin()` 10-11
 - `EnableLoggerPlugin()` 10-11
 - `LoadPlugin()` 10-11
 - `setCreator()` 10-15
 - `tick()` 10-4, 10-8
 - `UnloadPlugin()` 10-11
 - `update()` 10-5, 10-8

G

- GUI, localizing 2-17

H

- h command-line option 4-29
- handle, start and end time 5-12
- header file
 - lgrBaseSimManager.h* 10-6
 - lgrCommand.h* 10-5
 - lgrDisSimManager.h* 10-6
 - lgrDriver.h* 10-4
 - lgrHlaSimManager.h* 10-6
 - lgrPlaybackCommandFactory.h* 10-5
 - lgrPlaybackDriver.h* 10-4
 - lgrPlaybackInterface.h* 10-5
 - lgrProtocolRegistration.h* 10-6
 - lgrRuntimeCommandFactory.h* 10-5
 - lgrSimInput.h* 10-6
 - lgrSimOutput.h* 10-6
 - lgrSystemDriver.h* 10-4
 - logger.h* 10-14
 - messageInput.h* 10-6
 - messageOutput.h* 10-6
 - messageState.h* 10-6
 - observeConnection.h* 10-17
 - observeEvent.h* 10-17
 - observe.h* 10-17
 - runtimeValue.h* 10-15
 - stateObController.h* 10-5
 - stateObModel.h* 10-4
 - virtualConstructor.h* 10-15
- heartbeat 9-10
 - defined 3-6
 - setting 4-23
- Help command 9-5

- hiding
 - plug-in view 5-5
 - toolbar 5-5
- histogram 5-4, 5-23
- HLA 1-6
 - 1516
 - playback 3-4
 - advisory messages 4-13, 9-9
 - commands 9-9
 - configuring connection 9-9
 - configuring for 4-9
 - configuring Time Management 4-14
 - database schema 8-6
 - exporting to database 8-6
 - FOM flexibility 3-8
 - passive subscription 4-17, 9-9
 - specifications supported 1-6
 - subscription control list 4-15
 - subscription list 9-10
 - support for 3-8
 - Time Management 4-8
- host ID
 - FLEXIm 2-8
 - getting 2-8
- host name 2-8

I

- IEEE 1516 1-6
- ignoring, advisory messages 4-13
- immediate
 - playback 5-11
 - recording 5-6, 5-7
- importing, annotation file 6-5
- indexing, configuring 4-6
- informational message 4-7
- input
 - filtering 5-19
 - stage 10-6
- inserting, file 9-12
- installation 2-2
 - over network 2-4
 - silent 2-4
- installing
 - FLEXIm software 2-7
 - RTI 2-16
- instant annotation 6-3
- interaction
 - passive subscription 4-17
 - subscribing to 4-15

- interface 1-3
 - class 10-5
 - signal slot 10-16
 - text 9-2
- interim PDU 3-6, 3-7
- IP address 4-30

J

- jumping
 - range annotation end 6-6
 - range annotation start 6-6
 - to beginning of file 5-18
 - to end of file 5-18
 - to specific time 5-18

K

- keyboard, setting annotation times from 6-4

L

- l command-line option 4-8, 4-29
- language parameter 2-18
- lgrAnnotations library 10-18
- lgrApp library 10-17
- lgrBaseSimManager.h* header file 10-6
- lgrCommand.h* header file 10-5
- lgrConfig.xml 9-2
- lgrConfig.xml*, syntax and editing 4-25
- lgrConfig.xml* configuration file 4-3
- lgrCore library 10-17
- lgrDisSimManager.h* header file 10-6
- lgrDriver.h* header file 10-4
- lgrDump 1-5
- lgrGui library 10-17
- lgrHlaSimManager.h* header file 10-6
- lgrPlaybackCommandFactory.h* header file 10-5
- lgrPlaybackDriver.h* header file 10-4
- lgrPlaybackInterface.h* header file 10-5
- lgrProtocolRegistration.h* header file 10-6
- lgrRuntimeCommandFactory.h* header file 10-5
- lgrSim library 10-17
- lgrSimGui library 10-17
- lgrSimInput.h* header file 10-6
- lgrSimOutput.h* header file 10-6
- lgrSystemDriver.h* header file 10-4
- lgrUtil library 10-17
- library

- API 10-17
- appEnv 10-18
- lbrUtil 10-17
- lgrAnnotations 10-18
- lgrApp 10-17
- lgrCore 10-17
- lgrGui 10-17
- lgrSim 10-17
- lgrSimGui 10-17
- Logger API 10-18
- plug-in 10-7
- plug-ins 10-11
- license, floating 2-6
- license file 2-9
 - adding new 2-10
 - requesting 2-8
- license server
 - name 2-8
 - running 2-12
 - running as service 2-13
 - specifying port for 2-11
 - status 2-12
 - stopping 2-12
 - troubleshooting 2-14
- linking
 - API code 10-18
 - application 10-18
- Linux, installing on 2-3
- lmdown program 2-12
- lmgrd 2-6
- lmhostid 2-8
- lmstat program 2-12
- loading
 - annotation file automatically 5-11
 - configuration file 4-29
 - file 5-10, 9-5
 - plug-ins 10-12
 - subscription list 4-17, 9-9
- LoadPlugin command 9-5
- LoadPlugin() function 10-11
- LoadSubscriptionListFile command 4-17, 9-9
- localization 1-3
- localizing, GUI 2-17
- log file 4-8, 4-29
 - sql_export.txt* 8-6
- logFileName parameter 4-8
- Logger
 - embedding in application 10-7, 10-10
 - file, deleting 11-3

- recording
 - samples 1-4
 - viewing with Stealth 1-4
- window 5-4
- Logger API 10-3
 - architecture 10-4
 - coding conventions 10-13
 - compiling and linking 10-18
 - compiling and linking applications 10-18
 - drivers 10-4
 - embedding Logger 10-7, 10-10
 - examples 10-7
 - libraries 10-17
 - runtime values 10-15
- Logger command 4-3
- Logger Control
 - configuring 9-8
 - message, configuring response to 9-8
 - PDU 11-2
- Logger State Observer 10-10
- logger.h* header file 10-14
- logger.log* 4-29
- loop, enabling or disabling 9-6
- looping, playback 5-13

M

- m command-line option 4-7, 4-29
- macro, file naming convention 5-8
- MÄK RTI, configuring 2-16
- maklanddemo 1-4
- mak.lic* 2-9
- maklmgrd 2-6
- maklmgrd daemon 2-11
- MAKLMGRD_LICENSE_FILE environment
 - variable 2-9, 2-11
- MergeInFile command 9-12
- merging, files 7-4, 9-12
- message 9-8
 - advisory 4-13
 - filtering 1-3
 - filtering VR-Forces 5-22
 - informational 4-7
 - log file 4-29
 - pipeline 10-6
 - VR-Link 4-7
- messageInput.h* header file 10-6
- messageOutput.h* header file 10-6
- messageState.h* header file 10-6
- Microsoft Access, configuring data source for 8-7

- moving
 - to beginning of file 5-18
 - to beginning or end of playback range 5-18
 - to end of file 5-18
 - to specific time 5-18
- multicast address 4-24, 4-30, 9-10
- multicast time-to-live 4-22, 9-10
 - specifying 4-30
- multiple files
 - exporting to 8-3
 - recording to 5-7
- MySQL, configuring data source for 8-10

N

- N command-line option 4-10, 4-29
- n command-line option 4-7, 4-29
- name, Logger federate 4-10
- name reservation 3-4
- namespace 10-13
- naming, partial files 5-8
- naming convention, for database tables and rows 8-4
- network
 - connecting to 4-5
 - connection 4-4
 - requirements for Logger A-2
- networked installation 2-4
- new file 9-7
- NewFile command 9-7
- notification level, setting 4-7

O

- object
 - displaying statistics 5-24
 - name 9-9
 - passive subscription 4-17
 - subscribing to 4-15
- observeConnection.h* header file 10-17
- observeEvent.h* header file 10-17
- observe.h* header file 10-17
- observer pattern 10-17
- ODBC, configuring data source 8-6
- OffsetTime command 9-12
- offsetting
 - file 7-5
 - packets 9-12
- OMT file 8-6
- OpenFile command 9-5

- opening
 - annotation file 6-4
 - file 5-10, 9-5
- output
 - filtering 5-19
 - stage 10-6

P

- P command-line option 4-19, 4-30
- packet
 - clearing 7-3
 - count 5-23
 - deleting 7-3
 - from file 9-11
 - dumping to text 8-2
 - offsetting 7-5
 - time 9-12
 - preventing dropped 4-20
 - removing
 - empty 7-5
 - from file 9-11
- parameter
 - `commonEntityTypes` 5-20
 - `customPdu` 5-21
 - `languagefilename > untranslated.ts` file 2-18
 - `logFileName` 4-8
 - `showConsole` 9-2
 - `useAsyncIO` 4-20
- partial file, naming convention 5-8
- passive subscription 4-17
 - configuring 9-9
- path, specifying for RTI 2-16
- Pause command 9-5
- Pause on Record As 5-6, 5-7
- PauseRecording command 9-7
- pausing
 - playback 5-11, 9-5
 - recording 5-9, 9-7
- PDU
 - filtering 9-10
 - filtering by 5-20, 9-10
 - filtering custom 5-21
 - remote control 11-2
 - supported for database export 8-5
- pipeline 10-6
- Play button 5-11
- Play command 9-5
- Play On Load 5-10, 5-11
- playback 3-3, 5-10

- automatic 5-11
- bounding to annotation 6-8
- described 3-3
- exercise ID 3-3
- HLA 1516 3-4
- loading file 5-10
- looping 5-13, 9-6
- mode 3-3
- moving to beginning 5-18
- moving to end 5-18
- pausing 5-11, 9-5
- range, moving to beginning or end 5-18
- segment 5-12, 5-13
 - expanding 5-13
- setting speed 5-14, 9-6
- starting 9-5
- stopping 5-13, 9-6
- time-managed 4-8
- unpausing 9-6
- Player domain, commands 9-5
- playFile example
 - example, playFile 10-8
- playing
 - file 5-11
 - part of a file 5-12, 5-13
- PlayPause command 9-5
- plug-in 5-4
 - API 10-10
 - docking and undocking views 5-5
 - embedding in application 10-7
 - example 10-10
 - hiding or displaying 5-5
 - loading 10-12
 - object statistics 5-24
- plug-in library 10-11
- point annotation
 - adding instantly 6-3
 - converting to range 6-6
 - editing 6-5
- Point Annotation dialog box 6-5
- pointer 10-14
- port 9-10
 - specifying for license server 2-11
 - specifying UDP 4-19
- preferences, setting 4-6
- Preferences dialog box 4-6
- PrefixFileToStart command 9-12
- prefixing, file 9-12
- property
 - annotation 6-5

- connection 4-4
- protocol
 - supported 1-6
 - version, DIS 4-30

Q

- Qt Linguist 2-17

R

- r command-line option 4-29
- range annotation
 - adding instantly 6-3
 - bounding playback to 6-8
 - converting to point 6-6
 - editing 6-5
 - jumping to end 6-6
 - jumping to start 6-6
 - setting end time 6-4
- Range Annotation dialog box 6-5
- receive buffer 4-21, 9-10
 - size, specifying 4-30
- receiving application responses 10-5
- record, mode 3-3
- Record As, configuring 9-7
- Record command 9-7
- Recorder domain, commands 9-7
- recording 3-3
 - automatically 5-7
 - canceling 9-7
 - closing 9-7
 - configuring 9-7
 - described 3-3
 - exercise 5-6
 - exercise ID 3-3
 - immediately 5-6
 - multiple files 5-7
 - pausing 5-9, 9-7
 - resuming 5-9
 - saving 5-15
 - starting 5-6
 - stopping 5-9
 - time-managed 4-8
- remote control 11-2
 - example 11-3
- removing
 - dead space 9-12
 - empty packets 7-5
 - packets from file 9-11

- requesting, license file 2-8
- resetting, annotation time point 6-4
- response 10-5
- Restart command 9-5
- resuming, recording 5-9
- RevertFile command 9-6
- reverting
 - file 9-6
 - to saved file 7-2
- RID file 2-16
- rid.mtl* file 2-16
- row, naming conventions 8-4
- RPR FOM 3-8, 4-11
- RTI
 - definition of 2-16
 - specifications supported 1-6
- RTI_CONFIG environment variable 2-16
- runLm program 2-12
- running, license server 2-12
- runtime value 10-15
- runtimeValue.h* header file 10-15

S

- S command-line option 4-24, 4-30
- s command-line option 4-19, 4-30
- sample recording 1-4
- Save 5-15
- Save As 5-15
- SaveFile command 9-6
- SaveFileAs command 9-6
- SaveSubscriptionListFile command 4-16, 9-9
- saving
 - annotation file 6-4
 - annotations file 6-4
 - file 9-6
 - to new name 5-15
 - files 5-15
 - segment as new file 5-15
 - subscription list 4-16, 9-9
- scaling, time 3-7
- schema, database 8-4
- segment
 - playing 5-12, 5-13
 - saving as new file 5-15
 - setting playback to annotation 6-8
- selecting, annotation 6-5
- send buffer 4-21, 9-10
 - size, specifying 4-30
- sending, commands 10-5

- server name, FLEXlm 2-8
- Set Start and End Time dialog box 5-13
- Set Time 3-7, 5-18
- Set Zoom dialog box 5-17
- SetBackupOnSave command 5-15, 9-6
- SetCheckpointParameters command 4-7, 9-7
- SetCommonEntityTypes command
 - command, SetCommonEntityTypes 9-10
- SetConnectionParameters command 4-9, 4-18, 9-8, 9-9, 9-10
- setCreator() function 10-15
- SetCurrentTime command 9-6
- SetDisableTimeManagementTest command 9-9
- SetDynamicNamePrefix command 3-3, 9-9
- SetHeartbeatRate command 4-23, 9-10
- SetIgnoreAdvisories command 4-13, 9-9
- SetLoopAtEnd command 9-6
- SetNotifyLevel command 4-7, 9-5
- SetPassiveSubscription command 4-17, 9-9
- SetPauseOnRecordAs command 9-7
- SetPlaybackSpeed command 9-6
- SetPlayOnLoad command 9-6
- SetRemoteControlSettings command 9-8
- SetScaleAttributes command 9-8
- SetStartStopTimes command 9-6
- SetStopCondition command 9-7
- SetStopDuration command 9-7
- SetStopSize command 9-8
- SetSubscriptionLists command 4-16, 4-28, 9-10
- SetTempDir command 9-5
- SetTimeManagement command 4-14, 9-10
- SetTimeoutRate command 4-22, 9-10
- setting
 - checkpoint policy 9-7
 - cursor to time 9-6
 - DIS heartbeat 4-23
 - DIS timeout 4-22
 - end-of-file action 5-13
 - notification level 4-7
 - playback segment to annotation 6-8
 - playback speed 5-14, 9-6
 - preferences 4-6
- SetVRFFiltering command 9-8
- shared pointer 10-14
- showConsole** parameter 9-2
- shutting down, license server 2-12
- signal 10-16, 10-17
 - slot 10-16
- silent installation 2-4
- simulation

- manager 10-6
 - recording 5-6
 - standard, supported 1-6
 - Simulation domain, commands 9-8
 - SISO 1516 API 1-6
 - site ID 4-19, 9-10
 - specifying 4-30
 - size, indexing 4-6
 - skip command, entity maintenance 3-7
 - Skip Time 3-7
 - slider
 - start and end time 5-12
 - zoom 5-12
 - slot 10-17
 - signal 10-16
 - slow-speed playback 3-6
 - smart pointer 10-14
 - specialization, template 10-13
 - specifying
 - application number 4-30
 - configuration file 4-29
 - FED file 4-11, 4-12, 4-29
 - federation execution name 4-10
 - FOM Mapper 4-12
 - multicast time-to-live 4-30
 - receive buffer size 4-30
 - send buffer size 4-30
 - site ID 4-30
 - subscription list 9-10
 - Time Management 4-8
 - UDP port 4-19
 - speed
 - control 5-4
 - setting for playback 5-14
 - setting playback 9-6
 - SQL, exporting data to 8-4
 - sql_export.txt* log file 8-6
 - stage, input, output, state 10-6
 - start
 - jumping to range annotation 6-6
 - moving to 5-18
 - start time, setting dynamically 6-4
 - starting
 - Logger 5-3
 - playback 9-5
 - recording 5-6
 - starting Logger 5-3
 - startup
 - playback mode 3-3
 - record mode 3-3
 - running commands 4-29
 - UNIX 5-3
 - Windows 5-3
 - state stage 10-6
 - stateObController.h* header file 10-5
 - stateObModel.h* header file 10-4
 - status
 - field 5-6
 - license server 2-12
 - stopping
 - license server 2-12
 - playback 5-13, 9-6
 - recording 5-9
 - StopPlayback command 9-6
 - StopRecording command 9-8
 - string list 4-16
 - adding to configuration file 4-28
 - subscribing, passively 4-17
 - subscription control list, creating 4-15
 - subscription list
 - loading 4-17, 9-9
 - saving 4-16, 9-9
 - specifying 9-10
 - syntax, *lgrConfig.xml* 4-25
 - System domain, commands 9-5
- ## T
- table, naming conventions 8-4
 - tape
 - annotating 6-2
 - editing 7-2
 - loading 5-10
 - playing 5-11
 - starting 5-6
 - TCP 4-30
 - template 10-13
 - specialization 10-13
 - temporary file, recording to 5-6
 - text, dumping file to 8-2
 - text interface 9-2
 - thread safety
 - mutex, use of 10-16
 - tick() function 10-4, 10-8
 - time
 - command, entity maintenance 3-7
 - indexing 4-6
 - jumping to 5-18
 - navigating during playback 5-18
 - navigation 5-18

- offsetting packet 9-12
- setting 5-18
- setting cursor to 9-6
- sliders 5-12
- time jump, updating entities after 3-7
- Time Management
 - configuring 9-10
 - configuring in GUI 4-14
 - enabling or disabling 9-9
 - specifying use of 4-8
- time scale
 - described 3-4, 3-5, 3-6
 - during recording 5-6
 - effects 3-6
- time-constrained, setting 4-14
- timeline
 - cursor 5-18
 - moving 5-18
 - zooming 5-16
 - zooming out 5-17
 - zooming to annotation 6-7
- timeout 9-10
 - defined 3-6
 - setting 4-22
- timepoint, setting 5-18
- time-regulating, setting 4-14
- timestamp, offsetting 7-5
- toolbar 5-4
 - docking and undocking 5-5
 - hiding or displaying 5-5
- translating, GUI text 2-17
- translation 1-3
 - file, *untranslated.ts* 2-18
- trimming, file 7-5, 9-12
- TrimStart command 9-12
- troubleshooting, FLEXIm 2-14

U

- UDP port 4-30
 - specifying 4-19, 4-30
- undo, edits 7-2
- undocking, toolbars and plug-in views 5-5
- uninstalling application 2-4
- UNIX
 - compiling and linking API 10-18
 - installing on 2-3
 - starting, Logger 5-3
- UnloadPlugin() function 10-11
- Unpause command 9-6

- unpausing 9-6
- Update Annotation 6-4
- update() function 10-5, 10-8
- updating, annotation 6-4
- Upgrade File Format 5-16
- upgrading, file format 5-16
- useAsyncIO parameter 4-20

V

- v command-line option 4-29
- variable
 - adding to *lgrConfig.xml* 4-27
 - environment 2-9
 - syntax in *lgrConfig.xml* 4-26
- VCR controls 5-4
- velocity, scaling 3-7
- version, displaying 4-29
- View Annotations window 6-2
- view control PDUs 1-4
- View menu 5-5
- viewing
 - connection properties 4-4
 - object statistics 5-24
- virtual constructor 10-14
- virtualConstructor.h* header file 10-15
- VR-Forces message 5-22
- VR-Link, messages 4-7
- VR-Link.fed* file 1-6, 4-11

W

- window, Logger 5-4
- Windows
 - compiling and linking API 10-18
 - FLEXIm service 2-13
 - installing on 2-2
 - starting Logger 5-3

X-Y-Z

- x command-line option 4-10, 4-29
- XML, RTI configuration file 2-16
- XML file 8-6
- zoom area 5-12
- zoom bar 5-17
- zoom slider 5-16
- zooming
 - timeline 5-16

timeline out 5-17

timeline to annotation 6-7

MÄK Data Logger Commands Quick Reference Card

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.
SetNotifyLevel <i>vlLevel</i> (int) <i>lgrLevel</i> (int)	Sets the notification level of VR-Link and the Logger.
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)	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)	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.
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.

Command	Description
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>record</i> (bool) <i>allowDelete</i> (bool)	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>federationName</i> (string) <i>fedFile</i> (string) <i>fomVersion</i> (double) <i>fomMapper</i> (string) <i>fomMapperData</i> (base64)	Specifies HLA connection parameters indicated by the argument names.
SetDisableTimeManagementTest <i>disableTesting</i> (bool)	Disables testing for time management if that test causes the Logger to behave badly.
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.
SetIgnoreAdvisories <i>mode</i> (bool)	Specifies the Logger's response to HLA advisory messages. Default: false.
SetPassiveSubscription <i>mode</i> (bool)	Configures passive subscription for HLA. Default: false.
SetSubscriptionLists <i>objectClasses</i> (stringList) <i>interactionClasses</i> (stringList)	Sets the class subscription list.
SetTimeManagement <i>timeConstrainedRecording</i> (bool) <i>timeConstrainedPlayback</i> (bool) <i>timeRegulatingRecording</i> (bool) <i>timeRegulatingPlayback</i> (bool) <i>lookahead</i> (double)	Configures Time Management options.
DIS-specific commands	
AddCustomPDU <i>pduName</i> (string) <i>pduKind</i> (string) <i>entityIdOffset</i> (string) <i>entityTypeOffset</i> (string)	Specifies a PDU to add to the Filter by PDU Kind dialog box.
SetCommonEntityTypes <i>entityType1</i> ... <i>entityTypeN</i> (string)	Specifies a list of entity types to list in the Filter by Entity Type dialog box.
SetConnectionParameters <i>port</i> (int) <i>exerciseId</i> (int) <i>siteId</i> (int) <i>appNumber</i> (int) <i>useAsynchIO</i> (bool) <i>destAddr</i> (int) <i>sendBufferSize</i> (int) <i>recvBufferSize</i> (int) <i>multicastTtl</i> (int) <i>mcastAddresses</i> (stringList)	Specifies parameters for the DIS exercise connection.
SetHeartbeatRate <i>rate</i> (double)	Specifies the DIS heartbeat rate.
SetTimeoutRate <i>rate</i> (double)	Specifies the DIS timeout rate.
Editor domain commands	
AppendFileToEnd <i>filename</i> (string)	Appends the file <i>filename</i> to the end of the open file.
ClearSection <i>start</i> (time) <i>end</i> (time)	Removes all packets between the <i>start</i> and <i>end</i> times in a file.
ConvertFile	Converts the current file to the latest file format.
CutSection <i>start</i> (time) <i>end</i> (time)	Removes all packets between the <i>start</i> and <i>end</i> times in a file.
ExportFile <i>exportFormat</i> (string) <i>destination</i> (string) <i>checkpoint</i> (bool) <i>openAfterExport</i> (bool)	Exports a file using a given <i>exportFormat</i> to a <i>destination</i> ..
ExportSection <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> .
MergeInFile <i>filename</i> (string) <i>offset</i> (double)	Merges the Logger file <i>filename</i> into the open file.
OffsetTime <i>offset</i> (double)	Offsets the time of the packets in the open file by the <i>offset</i> value.
PrefixFileToStart <i>filename</i> (string)	Inserts the file named <i>filename</i> at the start of the open file.
TrimStart	Removes any dead time from the start of the file and offset all packets so that the first packet time stamp is 0.0.

MÄK Data Logger Quick Reference Card

Command-line Options

Option	Description
<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>-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>--fomMapperInitData data</code>	Specifies FOM Mapper initialization data, if required.
<code>-F fedfile</code>	Specifies the FED file for the exercise.
<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>-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>--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.
<code>-P 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.
<code>--sendBufferSize size</code>	Specifies the send buffer size.
<code>-s ID</code>	Specifies the site ID.

Keyboard Shortcuts

Action	Key	Action	Key	Action	Key
Annotations	Shift + A	Increase Speed	Ctrl + Up	Record	Ctrl + R
Append to End	Ctrl + Shift + E	Jump to End	End	Record As	Ctrl + N
Clear Section	Ctrl + Shift + Delete	Jump to Start	Home	Record To Multiple Files	Ctrl + M
Close	Ctrl + W	Jump to Time	Ctrl + T	Revert	Ctrl + Shift + R
Connection	Ctrl + Shift + C	Merge In File	Ctrl + Shift + I	Save	Ctrl + Shift + S
Create Instant Point Annotation	\	Modify Annotation	Return	Save As	Ctrl + Shift + A
Create Instant Range Annotation	[	Object Statistics	Shift + O	Save Recorded Data	Ctrl + Shift + S
Create Point Annotation	Ctrl + \	Offset Time	Ctrl + Shift + O	Set Start and End Times	Ctrl + B
Create Range Annotation	Ctrl + [	Open	Ctrl + O	Set Zoom	Ctrl + I
Data View	Shift + D	Pause	Ctrl + P	Stop	Ctrl + S
Decrease Speed	Ctrl + Down	Play	Ctrl + P	Toggle Loop at End	Ctrl + L
Delete Annotation	Delete	Preferences	Ctrl + Shift + P	Trim Start	Ctrl + Shift + T
Delete Section	Ctrl + X	Prefix to Start	Ctrl + Shift + W	Update Annotation	]
Disconnect	Ctrl + D	Properties	Return	Upgrade File Format	Ctrl + Shift + U
Exit	Ctrl + Q				



Link - Simulate - Visualize