



MÄK Data Logger

User's Guide





MÄK Data Logger

User's Guide

Copyright © 2006 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.

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-4.0-1-061004



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

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. Sample Logger Files	1-4
1.1.5. Sample Programs	1-4
1.2. Distributed Simulation Standards Supported	1-5
1.2.1. RTI Support	1-5
1.2.2. FOM Support	1-5

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	2-6
2.3. Setting Up and Using the FLEXlm License Manager	2-7
2.3.1. The FLEXlm Client-Server Architecture	2-7
2.3.2. Installing and Configuring the License Manager	2-8

2.3.3. Installing the License Manager Software	2-8
2.3.4. Requesting a License File	2-9
2.3.5. Putting the License File in the <i>flexlm</i> Directory	2-10
2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable	2-10
2.3.7. Adding Additional License Files	2-11
2.3.8. Running the License Server	2-12
2.3.9. Running FLEXlm as a Service on Windows	2-13
2.3.10. 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	2-19
2.5.2. Applying the Language Files	2-19

Chapter 3 Configuring the Logger

3.1. Configuring the Logger	3-3
3.2. Configuring the Exercise Connection through the GUI	3-3
3.2.1. Viewing Connection Properties	3-3
3.2.2. Disconnecting from an Exercise	3-4
3.2.3. Connecting to an Exercise	3-4
3.3. Setting Configuration Preferences in the GUI	3-5
3.3.1. Specifying the DIS Heartbeat	3-5
3.3.2. Specifying the DIS Timeout	3-5
3.3.3. Configuring Indexing	3-6
3.4. Setting the Notification Level	3-6
3.5. Configuring the Logger for HLA	3-7
3.5.1. Specifying the Federation Execution Name	3-7
3.5.2. Specifying A FED File	3-7
3.5.3. Specifying the RPR FOM Version	3-7
3.5.4. Specifying the Name of the Logger Federate	3-8
3.5.5. Specifying a FOM Mapper	3-8
3.5.6. Subscribing Passively to Objects and Interactions	3-9
3.5.7. Ignoring Advisory Messages	3-9
3.6. Configuring the Logger for DIS	3-9
3.6.1. Specifying A Destination Address for Outgoing and Incoming Data	3-9
3.6.2. Subscribing to Multicast Groups	3-10
3.6.3. Specifying A UDP Port	3-10
3.6.4. Setting the Buffer Size for Sending and Receiving PDUs	3-10
3.6.5. Using Asynchronous I/O	3-11
3.6.6. Setting the host:application:entity ID	3-11
3.6.7. Setting the Multicast Time to Live Value	3-11
3.6.8. Setting the Exercise ID	3-11

Chapter 4 **Logger Concepts**

4.1. What the Logger Does During Recording and Playback	4-2
4.1.1. The Exercise Connection	4-2
4.1.2. What The Logger Does During Recording	4-2
4.1.3. What The Logger Does During Playback	4-3
4.2. Timescale During Playback	4-3
4.2.1. Compensating for Timescale Effects During Playback	4-3
4.3. HLA Support	4-6
4.3.1. FOM Flexibility	4-6
4.3.2. Backward Compatibility	4-6

Chapter 5 **Using The Logger**

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
5.3. Recording Exercises	5-5
5.3.1. Enabling Immediate Recording	5-5
5.4. Playing Logger Recordings	5-6
5.4.1. Loading A Logger File for Playback	5-6
5.4.2. Playing A Logger File	5-6
5.4.3. Enabling Immediate Playback	5-6
5.4.4. Pausing Playback of A Logger File	5-6
5.4.5. Playing a Segment of a Recording	5-7
5.4.6. Specifying the End-of-File Action	5-7
5.4.7. Stopping Playback of A Logger File	5-7
5.4.8. Setting the Playback Speed	5-8
5.4.9. Closing a Logger File	5-8
5.5. Zooming the Timeline	5-9
5.5.1. Resetting the Zoom Level	5-9
5.6. Navigating to a Specific Time in a Recording	5-10
5.6.1. Jumping to the Beginning or End of A Recording	5-10
5.6.2. Moving to A Specific Time in A Recording	5-10
5.6.3. Setting the Time by Dragging the Timeline Cursor	5-10
5.7. Filtering Logger Input and Output	5-11
5.7.1. Filtering Options for DIS	5-11
5.7.2. Filtering HLA Exercises by Class	5-14
5.8. Annotating a Logger Recording	5-15
5.8.1. Adding an Annotation	5-15
5.8.2. Saving an Annotation File	5-16
5.8.3. Opening an Annotation File	5-16
5.8.4. Importing an Annotation File	5-17
5.8.5. Selecting an Annotation	5-17
5.8.6. Editing an Annotation	5-17
5.8.7. Deleting an Annotation	5-18

5.8.8. Jumping to the Start of an Annotation	5-18
5.8.9. Zooming the Timeline to an Annotation	5-19
5.8.10. Bounding Playback to an Annotation	5-20
5.9. Viewing Object Statistics	5-21
5.10. Summary of Command-line Options for the Logger	5-21

Chapter 6 Using The Text Interface

6.1. Using the Text Interface	6-2
6.1.1. Entering Commands	6-2
6.1.2. Displaying the List of Commands	6-3
6.2. Commands by Domains	6-4
6.2.1. System Commands	6-4
6.2.2. Playback Commands	6-4
6.2.3. Record Commands	6-5
6.2.4. Simulation Commands	6-6
6.3. Command List	6-7

Chapter 7 Using The Logger API

7.1. Introduction	7-3
7.2. The Logger Architecture	7-4
7.2.1. Drivers	7-4
7.2.2. Factories	7-5
7.2.3. Interfaces	7-5
7.2.4. The Logger Message Pipeline	7-6
7.2.5. The Simulation Manager	7-6
7.3. Embedding the Logger in an Application	7-7
7.3.1. The playFile Example	7-7
7.4. Creating a Stand-Alone Logger Application	7-10
7.5. The Logger Plug-in API	7-10
7.5.1. Writing a Plug-In	7-11
7.5.2. Loading Plug-Ins	7-12
7.6. Logger API Coding Conventions	7-13
7.6.1. Namespaces	7-13
7.6.2. Templates	7-13
7.6.3. Shared Pointers	7-14
7.6.4. Virtual Constructors	7-14
7.6.5. Runtime Values	7-15
7.6.6. Thread Safety	7-16
7.6.7. Interface Signal Slots	7-16
7.6.8. Observer Pattern	7-17
7.7. Logger Libraries	7-17
7.7.1. Other Libraries and Plug-ins	7-18
7.8. Compiling and Linking with the Logger API	7-18
7.8.1. Compiling and Linking for UNIX	7-18
7.8.2. Compiling and Linking for Microsoft Windows	7-19

Chapter 8 Remote Control

8.1. Remote Control Messages	8-2
8.2. Deleting Logger Files	8-3
8.3. Example of a Remote Control Application	8-3

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

Appendix B The Logger Configuration File

B.1. Editing lgrConfig.xml	B-2
B.1.1. Adding a Command to <i>lgrConfig.xml</i>	B-4
B.2. Logger Configuration Parameters	B-5

MÄK Products Glossary

Index

MÄK Data Logger Quick Reference Card



Preface

The MÄK Data Logger (Logger) records network activity of HLA or DIS exercises and plays it back for review. The Logger has a graphical user interface, a command-line interface, and several APIs.

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 the *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 the RTI.

Chapter 3, *Configuring the Logger*, explains how to configure the Logger for use with HLA or DIS.

Chapter 4, *Logger Concepts*, provides an overview of what the Logger does.

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

Chapter 6, *Using The Text Interface*, explains how to run the Logger from the command-line.

Chapter 7, *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 8, *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.

Appendix B, *The Logger Configuration File* lists and describes configuration parameters that you can use with the Logger.

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

The *MÄK Data Logger Quick Reference Card* lists command-line options, 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.
- ♦ 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.

Online Help

The Logger application online help uses the Qt Assistant help browser.

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) and the DIS protocol. VR-Link has built-in support for the RPR FOM and allows you to map to other FOMs. This library minimizes the time and effort required to build and maintain new HLA or DIS-compliant applications, and to integrate such compliance into existing applications.

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

- ♦ MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is optimized for high performance. It has an API, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface 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, or in reverse, and locate areas of interest quickly. The Logger has a graphical user interface and a command-line interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application.

The Logger comes with a set of utility programs that let you inspect, combine, and edit 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.
- ♦ **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.
- ♦ **MÄK HLA/DIS Toolbox for MATLAB and Simulink.** The MÄK HLA/DIS Toolbox is a wrapper around VR-Link that allows MATLAB and Simulink users to analyze data from live or recorded HLA or DIS exercises. It also allows Simulink users to build models that can participate in HLA or DIS exercises.
- ♦ **MÄK Game-Link** is a toolkit that allows games developed using Epic Games’ Unreal Engine to interoperate with DIS and HLA simulations and trainers.

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	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.



Introduction

This chapter describes the basic features of the Logger.

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

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 recording. 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 recording.
- ♦ Navigate to a specific time-point when playing a recording.
- ♦ Control playback speed and direction.
- ♦ Load a specific recording for playback.

You can use all the interfaces at the same time.

Localization

The Logger include translation files that allow you to localize the Logger's GUI. 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.

1.1.3. Annotations

You can annotate Logger recordings and jump directly to annotation points.

1.1.4. Sample Logger Files

The Logger includes several sample recordings, 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 recordings 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 recording uses the Makland database, which is distributed with the MÄK Stealth, MÄK Plan View Display, and VR-Forces.



The recordings 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.

1.1.5. Sample Programs

The Logger includes the following example programs:

- ♦ Simple examples of using the Logger API: *playFile* and *writeFile*
- ♦ An example of a plug-in: *stateObserver*
- ♦ 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. 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. (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

If you want to use the Logger 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. 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 a UNIX System.....	2-3
Silent Installation and Uninstallation on Windows.....	2-4
The Logger Directory Structure	2-6
Setting Up and Using the FLEXlm License Manager	2-7
The FLEXlm Client-Server Architecture	2-7
Installing and Configuring the License Manager.....	2-8
Installing the License Manager Software	2-8
Requesting a License File.....	2-9
Putting the License File in the flexlm Directory.....	2-10
Setting the MAKLMGRD_LICENSE_FILE Variable.....	2-10
Adding Additional License Files.....	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.....	2-19
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 2-7). 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 on a PC.

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

Installing the MÄK Data Logger from a CD

The Logger may be supplied to you on an “All Products” CD with a common installation interface or a CD that does not support common installation.

To install the Logger from a CD:

1. Insert the Logger CD into your CD-ROM drive.
 - If this is an All Products CD, and the autorun feature 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.
 - If this is not an All Products CD, and autorun is enabled, an installation program may start immediately. Follow the instructions of the installation program.

If the installation does not start automatically, open the Windows Explorer and select the CD drive. In the root directory for the Logger, double-click *setup.exe*. Go to step 3.
2. In the MÄK PC Products Installation window, click Logger.
3. Follow the instructions of the installation program.

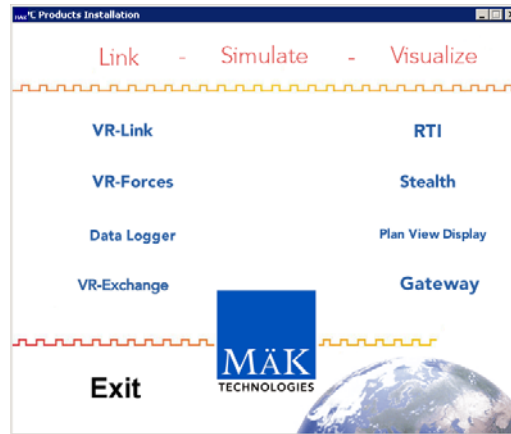


Figure 2-1. MÄK PC Products Installation window

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, *logger4.0-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, *logger4.0--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, *logger4.0--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
├── bin5
├── binHLA13
├── binHLA1516
├── data
├── doc
├── examples
├── flexlm10.8
├── include
├── lib
├── logger_tapes
└── 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>bin5</i>                              | Executable for DIS versions 2.0.4, 2.1.4, IEEE 1278, and IEEE 1278.1a.         |
| <i>binHLA13</i> and<br><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>flexlm10.8</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>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. (For general information about FLEXlm, see <http://www.macrovision.com/products/flexlm/index.shtml>.) 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.

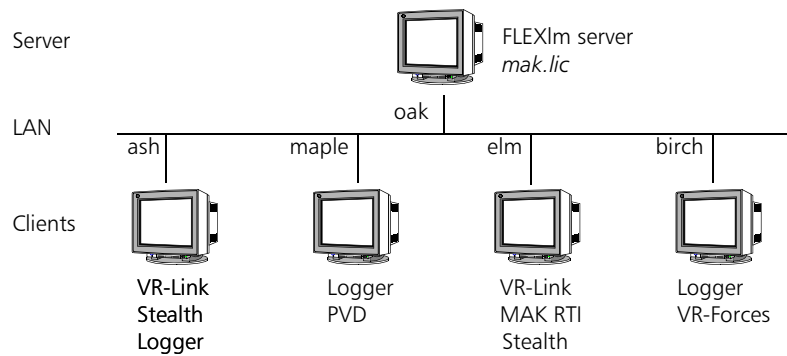
Licenses “float”, so you can install product software on as many computers as you want, but at any given time, you can run only as many instances of a product as you have licensed.



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, see [“Adding Additional License Files,”](#) on page 2-11.

### 2.3.3. Installing the License Manager Software

As part of the installation of MÄK products, the FLEXlm software is installed in *./flexlm10.8*.

1. Install your MÄK products on the machines on which you want to run them.
2. Copy the files in *flexlm10.8* 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. See the instructions in [“Adding Additional License Files,”](#) on page 2-11.

---

### 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 2000

To add the MAKLMGRD\_LICENSE\_FILE in Windows 2000:

1. Right-click the My Computer icon on the Desktop.
2. On the pop-up menu, select Properties. 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.

### **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. Follow steps 3 through 8 in the Windows 2000 procedure.

### **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. See [“Requesting a License File,”](#) on page 2-9.

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. 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`.

### **2.3.9. Running FLEXlm as a Service on Windows**

You can configure FLEXlm to run as a service on Windows 2000 or 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, specify 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.

### 2.3.10. Troubleshooting the FLEXlm License Manager

For general troubleshooting information, refer to the FLEXlm FAQ at <http://www.macrovision.com/products/flexlm/index.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. 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](mailto: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, 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

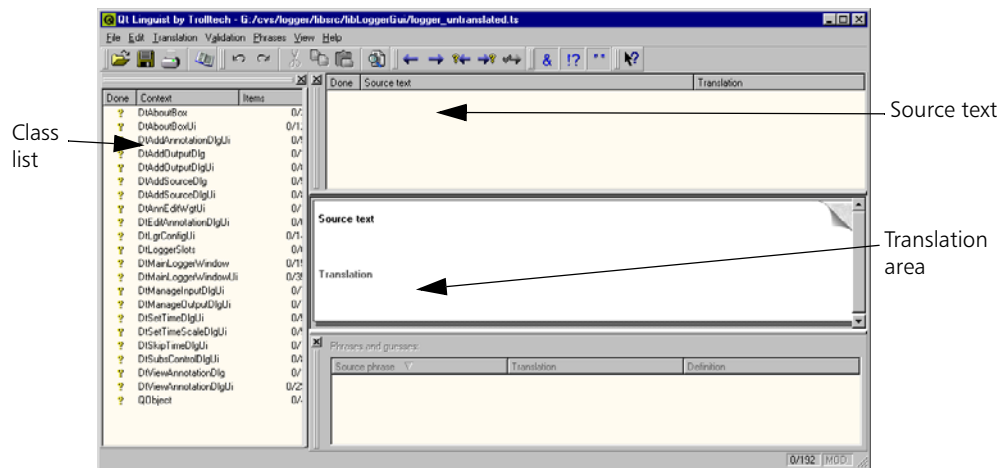
In many cases, you do not need an RTI server, such as the `rtiexec`, or a central application to use the MÄK RTI, however you can run the `rtiexec` if you want to. (It is required to use certain features of the MÄK RTI.) Be sure the FLEXlm license server is running, then start the applications, and they should be able to communicate. (For information about the license server, see “Setting Up and Using the FLEXlm License Manager”.) 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

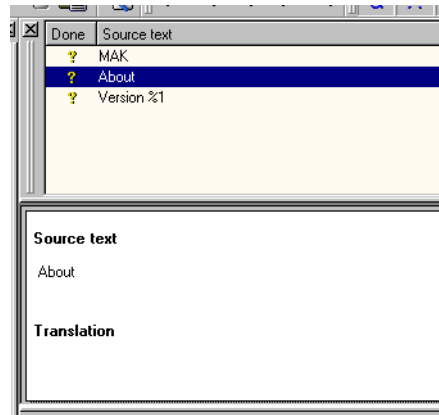
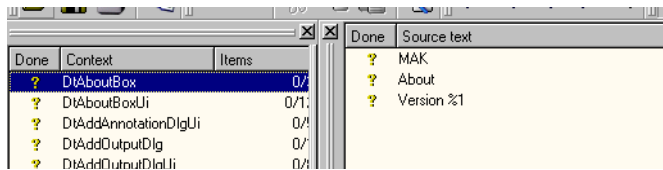
You can localize the Logger interface so that it uses languages other than English.

To localize the Logger, you edit a resource file using the following procedure:

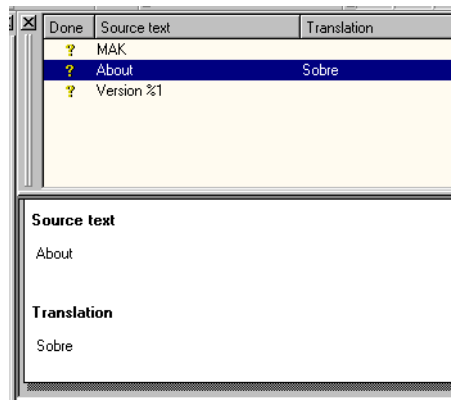
1. Start the Qt Linguist utility. It is `./translations/linguist.exe`
2. In the Qt Linguist window, choose File → Open. Select the file `./translations/logger_untranslated.ts`. The translation file is opened and a list of classes is displayed on the left side of the window, as illustrated.



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, the first class in the list, *DtAboutBox*, contains the text for the About Box.
4. In the Source Text list, select the word “About”. The text is displayed in the translation area (the window below the source text list.)
5. Select *DtAboutBox*. The text in the about box is displayed in the Source Text list. The yellow question marks indicate that the text has not been translated yet.



6. Place the cursor in the space below the word “Translation”.
7. Type the translation for the word “About”. As you type, the translation is displayed in the Translation column.



8. 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.
9. Repeat this process for each element of source text for each class.

10. 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.
11. 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** parameter in *lgrConfig.xml* to the two-letter country code for the local language.





## *Configuring the Logger*

---

This chapter explains how to configure connections settings for DIS and HLA in configuration files and from the command line.

|                                                                       |      |
|-----------------------------------------------------------------------|------|
| Configuring the Logger.....                                           | 3-3  |
| Configuring the Exercise Connection through the GUI.....              | 3-3  |
| Viewing Connection Properties .....                                   | 3-3  |
| Disconnecting from an Exercise.....                                   | 3-4  |
| Connecting to an Exercise .....                                       | 3-4  |
| Setting Configuration Preferences in the GUI.....                     | 3-5  |
| Specifying the DIS Heartbeat .....                                    | 3-5  |
| Specifying the DIS Timeout .....                                      | 3-5  |
| Configuring Indexing.....                                             | 3-6  |
| Setting the Notification Level.....                                   | 3-6  |
| Configuring the Logger for HLA .....                                  | 3-7  |
| Specifying the Federation Execution Name.....                         | 3-7  |
| Specifying A FED File .....                                           | 3-7  |
| Specifying the RPR FOM Version .....                                  | 3-7  |
| Specifying the Name of the Logger Federate .....                      | 3-8  |
| Specifying a FOM Mapper .....                                         | 3-8  |
| Subscribing Passively to Objects and Interactions .....               | 3-9  |
| Ignoring Advisory Messages.....                                       | 3-9  |
| Configuring the Logger for DIS.....                                   | 3-9  |
| Specifying A Destination Address for Outgoing and Incoming Data ..... | 3-9  |
| Subscribing to Multicast Groups .....                                 | 3-10 |
| Specifying A UDP Port .....                                           | 3-10 |
| Setting the Buffer Size for Sending and Receiving PDUs .....          | 3-10 |
| Using Asynchronous I/O .....                                          | 3-11 |

## **Configuring the Logger**

---

|                                               |      |
|-----------------------------------------------|------|
| Setting the host:application:entity ID.....   | 3-11 |
| Setting the Multicast Time to Live Value..... | 3-11 |
| Setting the Exercise ID .....                 | 3-11 |

### 3.1. Configuring the Logger

You can configure the Logger by editing *lgrConfig.xml*, with command-line options, and for some options, after you start the Logger. Command-line options override settings in *lgrConfig.xml*. For a complete list of command-line options, please see “[Summary of Command-line Options for the Logger](#),” on page 5-21 or the Quick Reference Card.

Logger configuration parameters and commands are stored in *lgrConfig.xml*. This file is essentially a way to send commands to the Logger through a file, rather than by typing command-line options. 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/config*. 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.

For a description of the default configuration parameters and how to edit *lgrConfig.xml*, please see “[Logger Configuration Parameters](#),” on page B-5.

### 3.2. Configuring the Exercise Connection through the GUI

You can configure the exercise connection in *lgrConfig.xml* and you can connect to exercises and disconnect from them through the GUI. When you connect to an exercise you can specify the parameters for the protocol you are using (DIS or HLA). The connection status is displayed in the Logger status bar ([Figure 3-1](#)).

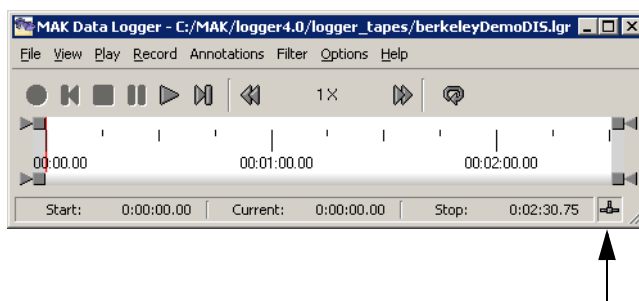


Figure 3-1. Connection status

#### 3.2.1. Viewing Connection Properties

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 3-2](#)).

### 3.2.2. Disconnecting from an Exercise

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



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

- To disconnect from an exercise, choose File → Disconnect, or double-click the connection status icon.

### 3.2.3. 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, see “Configuring the Logger for HLA,” on page 3-7 and “Configuring the Logger for DIS,” on page 3-9.

To connect to an exercise:

1. Choose File → Connection. The Connection Settings dialog box opens (Figure 3-2). It displays the settings configured by *lgrConfig.xml* and command-line options. The settings are specific to the protocol you are using (HLA or DIS.)

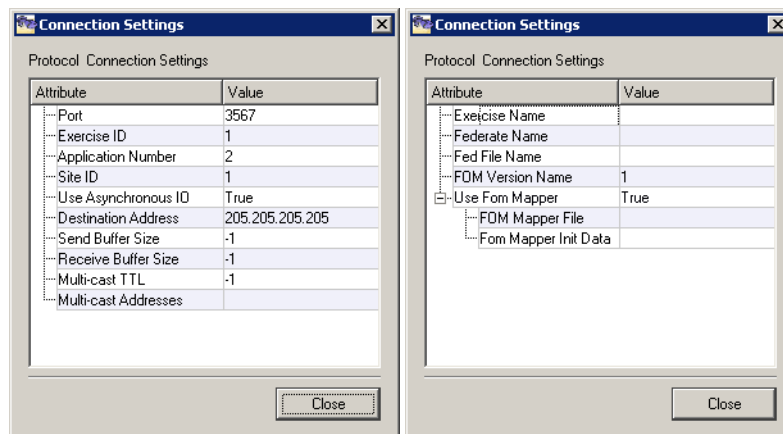


Figure 3-2. Connection Settings dialog box for DIS and HLA

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.

### 3.3. Setting Configuration Preferences in the GUI

You can set several protocol-specific configuration parameters in the Preferences dialog box.

#### 3.3.1. Specifying the DIS 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 4-4.

To set the DIS heartbeat:

1. Choose Options → Preferences. The Preferences dialog box opens ([Figure 3-3](#)).

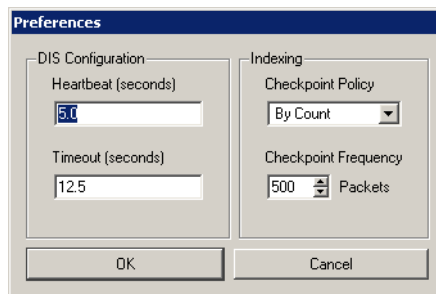


Figure 3-3. Preferences dialog box for DIS

2. Type a value in the Heartbeat text box.

#### 3.3.2. Specifying the DIS 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 recording. For more information, please see [“Generating Interim PDUs During Slow-Speed DIS Playback,”](#) on page 4-4.

To set the DIS timeout:

1. Choose Options → Preferences. The Preferences dialog box opens ([Figure 3-3](#)).
2. Type a value in the Timeout text box.

### 3.3.3. Configuring Indexing

Logger recordings contain index data that enable quick jumps from one time point to another. When you record, indexing is always enabled. You can set the frequency of index entries, in packets received.

To configure indexing:

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

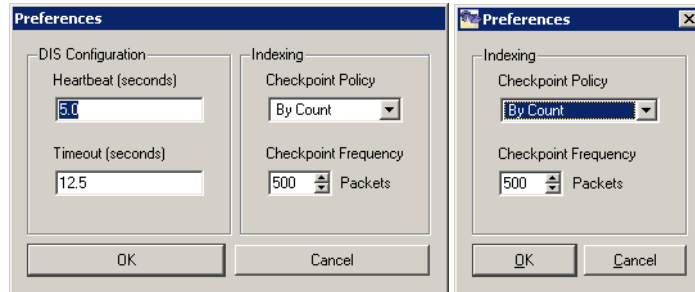


Figure 3-4. Preferences dialog boxes for DIS and HLA

2. Edit the checkpoint frequency value.
3. Click OK.

## 3.4. Setting the Notification Level

The notification level determines the verbosity of informational messages that the Logger generates. The logger sends messages to a log file. The default log file is *logger.log*. You can change the log file with the `-l` command-line option. The `-n` command-line option is the equivalent of the `messageLevel` parameter in *lgrConfig.xml*. The range is 0-4, with 0 being least verbose.

- To specify the notification level, start the Logger with the `-n` command-line option or set the `messageLevel` and `notifyLevel` parameters in *lgrConfig.xml*.

## 3.5. Configuring the Logger for HLA

This section describes configuration parameters and command-line options for HLA exercises. For HLA exercises, you must specify the federation execution and FOM.

### 3.5.1. Specifying the Federation Execution Name

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

- To specify a federation execution name, use the `-x` option at startup:

```
logger -x FedExName
```

where *FedExName* is the name of the federation execution. You can also specify the federation execution name with the `execName` parameter in *lgrConfig.xml*.



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, see “[Specifying A FED File](#),” on page 3-7.

---

### 3.5.2. Specifying A FED File

By default the Logger uses a FED file called *VR-Link.fed*, but you can specify an alternate FED file using the `-F` command-line option or the `fedFileName` parameter in *lgrConfig.xml*. The Logger also accepts federation description files with the *.fdd* and *.xml* extensions. See *MÄK RTI Reference Manual* for details.

MÄK products come with a FED file for each version of the RPR FOM that they support.

The following example instructs the Logger to use the *VR-Link2.0017-1.fed* FED file:

```
logger -F Link20017-1.fed
```

### 3.5.3. 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). You can specify the RPR FOM version with the `rprFomVersion` parameter or with the `--rprFomVersion` command-line option. If you specify a different FOM version, you must also specify the correct FED file, for example.

```
logger --rprFomVersion 2.0017 -F VR-Link20017-1.fed
```



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

---

### 3.5.4. Specifying the Name of the Logger Federate

By default, the Logger federate is named `Logger`. If you want to give it a different name, you can use the `-N` command-line option or the `federateName` parameter in *lgrConfig.xml*.

```
logger -N myLogger
```

### 3.5.5. Specifying a FOM Mapper

The Logger supports different FOMs using VR-Link FOM Mappers. You can specify which FOM Mapping code to use by editing the `rprFomVersion` or `fomMapperLib` parameters in *lgrConfig.xml*, or with the `-f` command-line option, as follows:

```
logger -f myFomMapper.dll
```



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

---

#### 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. If you specify a FOM Mapper that requires such data, specify the data with the `--fomMapperInitData` option or the `fomMapperInitData` parameter, for example:

```
logger -f myFomMapper --fomMapperInitData ABC
```

The string “ABC” is passed to the FOM Mapper creation function in the shared library as an initialization parameter.

### 3.5.6. 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.

- To have the Logger subscribe passively, issue the `SetPassiveSubscription` command from the text interface or add `SetPassiveSubscription` to *lgrConfig.xml*. Please see “[Simulation Commands](#),” on page 6-6 for details.

### 3.5.7. Ignoring Advisory Messages

- To have the Logger ignore advisory messages, set the `SetIgnoreAdvisories` command from the text interface, or add `SetIgnoreAdvisories` to *lgrConfig.xml*. Please see “[Simulation Commands](#),” on page 6-6 for details.

## 3.6. Configuring the Logger for DIS

This section describes configuration parameters and command-line options for DIS exercises.

### 3.6.1. 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.

- To specify a different destination address, change the `destAddrString` parameter in the *lgrConfig.xml* file, or use the `-A` command-line option, as in this example:

```
logger -A 123.234.123.234
```

The specified destination address can be a broadcast, multicast, or unicast address.

### 3.6.2. 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.

- To subscribe to multicast groups, use the `-S` command-line option or edit the `mcastAddresses` parameter in `lgrConfig.xml`. To subscribe to more than one multicast address, use the `-S` option several times in the same command:

```
logger newfile.lgr -S 225.0.0.118 -S 225.0.0.202
```

Subscription applies to record mode only.



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.

---

### 3.6.3. Specifying A UDP Port

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

- To specify the UDP port, specify the `disPort` parameter in the `lgrConfig.xml` file, or start the Logger with the `-P` option, for example:

```
logger -P 2727
```

### 3.6.4. 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 the size of the receive buffer, set the `recvBufferSize` parameter in the `lgrConfig.xml` file, or use the `--receiveBufferSize` command-line option, as follows:

```
logger --receiveBufferSize 64000
```

- To specify the size of the send buffer, set the `sendBufferSize` parameter in the `lgrConfig.xml` file, or use the `--sendBufferSize` command-line option, as follows:

```
logger --sendBufferSize 64000
```

### 3.6.5. 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 configure asynchronous I/O, set the `useAsyncIO` parameter to 1 in the *lgrConfig.xml* file.

### 3.6.6. Setting the host:application:entity ID

- To specify the host:application:entity ID, use the `-a` command-line option, for example:

```
logger -a 3000:2:34
```

### 3.6.7. Setting the Multicast Time to Live Value

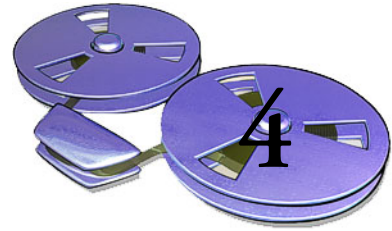
The multicast time to live value specifies the number of router jumps a message can take. To specify it, use the `--mcastTtl` command-line option or specify the `mcastTtl` parameter in *lgrConfig.xml*.

### 3.6.8. Setting the Exercise ID

To set the DIS exercise ID, use the `-e` option or set the `exerciseld` parameter in *lgrConfig.xml*, for example:

```
logger -e 2
```





## *Logger Concepts*

---

This chapter describes Logger concepts that are common to all interfaces. For step-by-step instructions about how to operate the Logger, see Chapter 5, *Using The Logger* and Chapter 6, *Using The Text Interface*.

|                                                          |     |
|----------------------------------------------------------|-----|
| What the Logger Does During Recording and Playback ..... | 4-2 |
| The Exercise Connection.....                             | 4-2 |
| What The Logger Does During Recording .....              | 4-2 |
| What The Logger Does During Playback .....               | 4-3 |
| Timescale During Playback.....                           | 4-3 |
| Compensating for Timescale Effects During Playback.....  | 4-3 |
| HLA Support.....                                         | 4-6 |
| FOM Flexibility .....                                    | 4-6 |
| Backward Compatibility.....                              | 4-6 |

## 4.1. What the Logger Does During Recording and Playback

This section describes, in general, what happens when you record and play back exercises. [Figure 4-1](#) illustrates the Logger window.

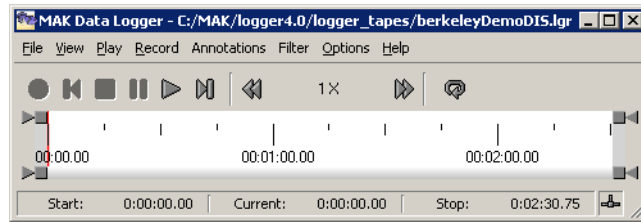


Figure 4-1. The Logger window

### 4.1.1. 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 through the GUI,”](#) on page 3-3.

### 4.1.2. What The Logger Does During Recording

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

#### **DIS only**

In record mode, the Logger records all DIS packets it receives, without regard to exercise ID. With each PDU it records, the Logger stores a timestamp indicating when the PDU occurred relative to the start of the recording.

#### **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.

### 4.1.3. 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 recording 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 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, see [“Timescale During Playback,”](#) on page 4-3.

The Logger lets you navigate to any timepoint in a recording. For procedures, please see [“Navigating to a Specific Time in a Recording,”](#) on page 5-10.

#### *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* file, the command line, or in the Connection Settings dialog box.)

#### *HLA only*

In HLA, the Logger acts as a federate that owns each of the objects represented in the recording. 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.

## 4.2. Timescale During Playback

The Logger lets you play a recording 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 forward, half-speed playback.

For procedures, see [“Setting the Playback Speed,”](#) on page 5-8.

### 4.2.1. 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, see [“Specifying the DIS Heartbeat,”](#) on page 3-5.

### ***Scaling the Timeout Period During Slow-Speed Playback***

When an entity's PDUs stop occurring in the recording, 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 recording 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, see [“Specifying the DIS Timeout,”](#) on page 3-5.

### **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.

To speed the update process, the Logger provides a feature called entity maintenance. 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 by entity maintenance, 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.

## 4.3. 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.

### 4.3.1. 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.

### 4.3.2. 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.



## *Using The Logger*

---

This chapter how to use the Logger's graphical user interface (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  |
| Recording Exercises.....                               | 5-5  |
| Enabling Immediate Recording .....                     | 5-5  |
| Playing Logger Recordings .....                        | 5-6  |
| Loading A Logger File for Playback .....               | 5-6  |
| Playing A Logger File.....                             | 5-6  |
| Enabling Immediate Playback .....                      | 5-6  |
| Pausing Playback of A Logger File .....                | 5-6  |
| Playing a Segment of a Recording.....                  | 5-7  |
| Specifying the End-of-File Action.....                 | 5-7  |
| Stopping Playback of A Logger File .....               | 5-7  |
| Setting the Playback Speed .....                       | 5-8  |
| Closing a Logger File.....                             | 5-8  |
| Zooming the Timeline .....                             | 5-9  |
| Resetting the Zoom Level.....                          | 5-9  |
| Navigating to a Specific Time in a Recording.....      | 5-10 |
| Jumping to the Beginning or End of A Recording .....   | 5-10 |
| Moving to A Specific Time in A Recording .....         | 5-10 |
| Setting the Time by Dragging the Timeline Cursor ..... | 5-10 |
| Filtering Logger Input and Output .....                | 5-11 |
| Filtering Options for DIS .....                        | 5-11 |

|                                                     |      |
|-----------------------------------------------------|------|
| Filtering HLA Exercises by Class .....              | 5-14 |
| Annotating a Logger Recording .....                 | 5-15 |
| Adding an Annotation.....                           | 5-15 |
| Saving an Annotation File .....                     | 5-16 |
| Opening an Annotation File.....                     | 5-16 |
| Importing an Annotation File .....                  | 5-17 |
| Selecting an Annotation .....                       | 5-17 |
| Editing an Annotation.....                          | 5-17 |
| Deleting an Annotation.....                         | 5-18 |
| Jumping to the Start of an Annotation .....         | 5-18 |
| Zooming the Timeline to an Annotation.....          | 5-19 |
| Bounding Playback to an Annotation.....             | 5-20 |
| Viewing Object Statistics .....                     | 5-21 |
| Summary of Command-line Options for the Logger..... | 5-21 |

## 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 5-21, 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.

## 5.2. The Logger Window

The Logger window has a VCR-style set of control buttons, an information display area, a timeline, and a status bar (Figure 5-1).

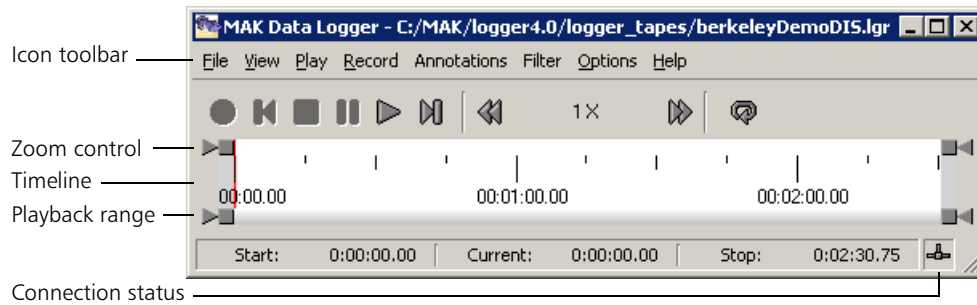


Figure 5-1. Logger window with recording loaded

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

- Record (1)
- Jump to the start of the recording (2)
- Stop (3)
- Pause (4)
- Play (5)
- Jump to the end of the recording (6)
- Decrease the playback speed (by 50%) (7)
- Displays current speed; open Set 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

## 5.3. Recording Exercises

To record an exercise:

1. Choose File → New. The New 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 recording.

---

3. Click Save. The Logger enters record mode. If Record On New is enabled, the Logger starts recording immediately.
- If Record On New is disabled, to begin recording, choose Record → Record, or click the Record button.
  - To stop recording, choose Record → Stop, or click the Stop button.

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. Enabling Immediate Recording

When Record On New is enabled, the Logger starts recording as soon as you specify a file name for recording. When Record On New is disabled, you must click the Record button to start recording.

- To enable or disable immediate recording, choose Options → Record On New.

## 5.4. Playing Logger Recordings

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

### 5.4.1. Loading A Logger File for Playback

To load a Logger file for playback:

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



If you load a recording 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.

---

### 5.4.2. Playing A Logger File

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

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

### 5.4.3. Enabling Immediate Playback

When Play On Load is enabled, the Logger starts playing as soon as you open a recording. If the recording 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.

### 5.4.4. Pausing Playback of A Logger File

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

### 5.4.5. Playing a Segment of a Recording

You can configure the Logger to play any arbitrary segment of a recording. If you have the Logger configured to wrap at the end of a recording, it loops within the segment you have specified.

To play a segment of a recording:

1. On the Logger timeline, drag the start time slider (Figure 5-3) 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.

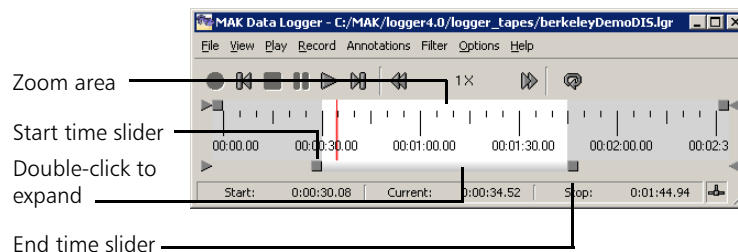


Figure 5-3. Timeline sliders

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.



The zoom display shows the selected timeline area.

3. Play the recording.

#### Expanding the Playback Segment to Full Length

- To expand the playback segment to the full length of the recording, drag the sliders back to the edge of the window or double-click the timeline between the start and end time sliders (Figure 5-3).

### 5.4.6. Specifying the End-of-File Action

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

- To set the end-of-file action, choose Play → Loop at End, or click the Loop At End icon in the Logger window (Figure 5-2).

### 5.4.7. Stopping Playback of A Logger File

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

### 5.4.8. 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 Speed dialog box lets you increase and decrease the speed by any integer value. If you set the speed using the Set 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 recording depends on the speed at which you recorded it.

To set the playback speed:

1. Choose Play → Playback Speed → Set Speed, or click the current speed setting on the Logger toolbar. The Set Speed dialog box opens.

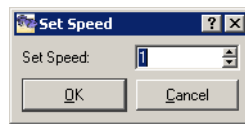


Figure 5-4. Set Speed dialog box

2. Type the speed or select 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](#)).

### 5.4.9. Closing a Logger File

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

## 5.5. 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.

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

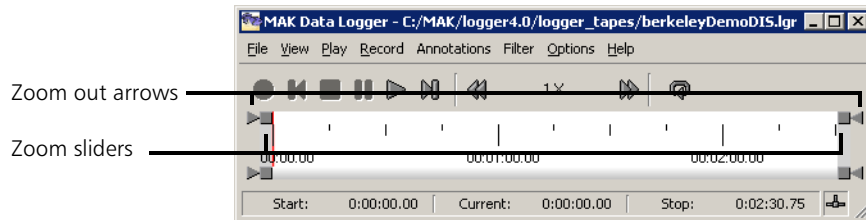


Figure 5-5. Zoom sliders

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.

### i

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 Recording,” on page 5-7 for details about the playback time.

### 5.5.1. Resetting the Zoom Level

- To quickly zoom the timeline to its full length, double-click on the zoom bar between the zoom sliders (Figure 5-6).

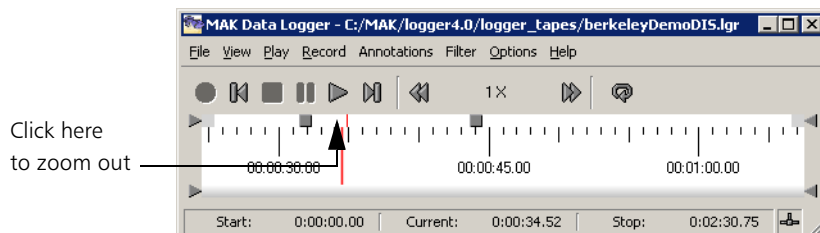


Figure 5-6. Zooming out

## 5.6. Navigating to a Specific Time in a Recording

The Logger provides the following ways to navigate to a specific time in the exercise:

- ♦ Jump to the beginning or end of the recording
- ♦ Jump to a specific time
- ♦ Jump to a point defined by an annotation
- ♦ Drag the time cursor.

### Notes

- ♦ 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 performs entity maintenance to speed the transmission of up-to-date messages. For more information, see [“Updating Entity Information Quickly After A Time Jump,”](#) on page 4-5.

### 5.6.1. Jumping to the Beginning or End of A Recording

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

### 5.6.2. Moving to A Specific Time in A Recording

The Set Time feature lets you move to a specific point in time measured from the start of the recording.

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.

### 5.6.3. 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.

- 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.

## 5.7. Filtering Logger Input and Output

When you record an exercise or play a recording, 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 recording and playback. Settings do not persist between sessions.

### 5.7.1. Filtering Options for DIS

This section describes the filtering options for DIS: filtering by entity type and PDU kind.

#### Filtering 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 5-7). The Common Entity Types window lists entity types that you have previously added to the list (either manually or by configuring the list in *lgrConfig.xml*.

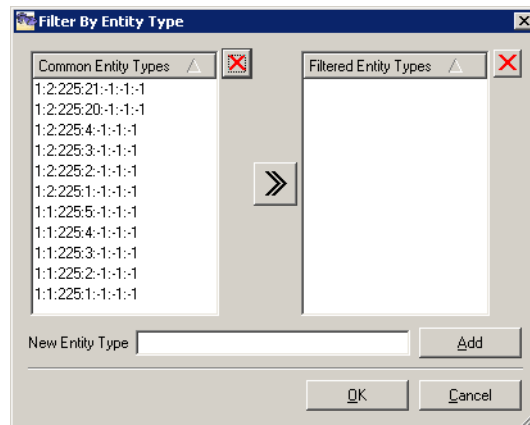


Figure 5-7. Filter By Entity Type dialog box

If the entity type that you want to filter is not listed in the Common Entity Types window:

- a. Enter the 7-digit enumeration for the entity in the New Entity Type box. Separate enumeration values with colons (:).
- b. Click Add. The entity type is added to the Common Entity Types list.



The New Entity Type box checks the entity type for validity. Invalid entity types get deleted when you click Add.

2. In the Common Entity Types list, select the entity type that you want to filter out.
3. Click the right-facing arrow. The entity type is added to the Filtered Entity Types list.

#### **Removing Entity Types from the Entity Types Lists**

- To remove an entity type from one of the Entity Types lists, select the entity type and click the Delete button for that list.



If you delete an entity type from the Filtered Entity Types list, its corresponding entry in the Common Entity Types list remains in that list. If you delete an entity type from the Common Entity Types list it is also deleted from the Filtered Entity Types list if it is being filtered.

---

#### **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, you 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>
```

## Filtering 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 5-8).

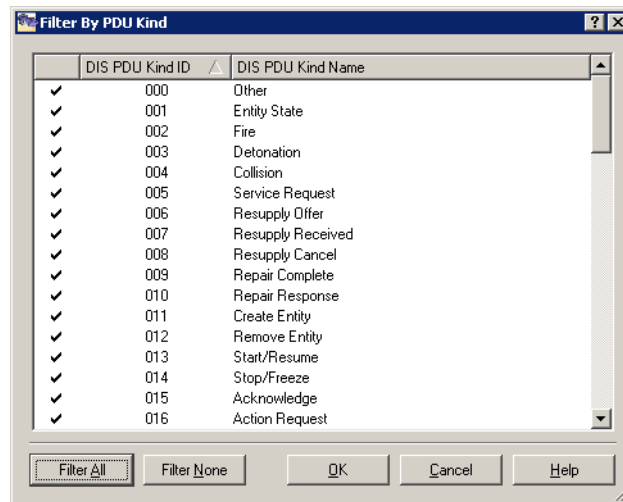


Figure 5-8. Filter By PDU Kind dialog box

2. To change the filtering status of an object, click the current filter status (check mark or X) for that object. It toggles to the alternative status.
3. To filter out all objects, click the Filter All button.
4. To filter out no objects, click the Filter None column.

## 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` parameter 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.

### 5.7.2. 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 5-9).

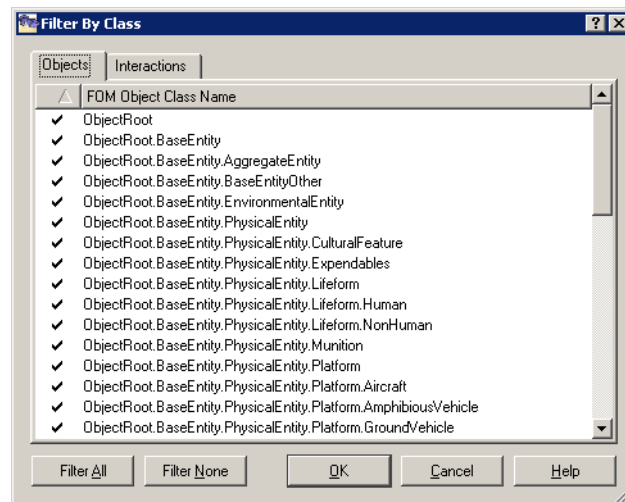


Figure 5-9. Filter By Class dialog box

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, click the current filter status (check mark or X) for that object. It toggles to the alternative status.
4. To filter out all classes, click the Filter All button.
5. To filter out no classes, click the Filter None column.

## 5.8. Annotating a Logger Recording

An annotation specifies a particular time or time range in a Logger recording to which you can jump. You can associate text with an annotation.

### 5.8.1. Adding an Annotation

To annotate a Logger recording:

1. Choose View → Annotations. Annotations, if any, are displayed in the Logger data display area (Figure 5-10).

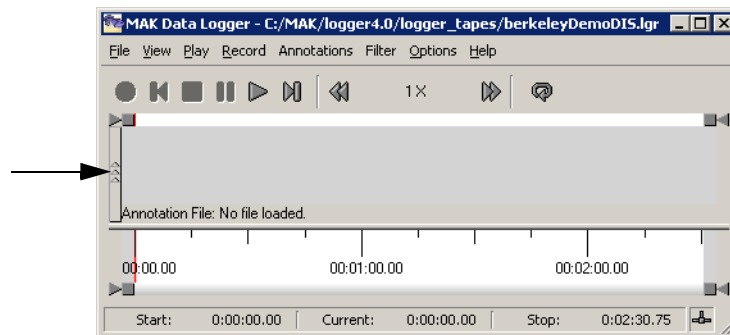


Figure 5-10. Annotations menu

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

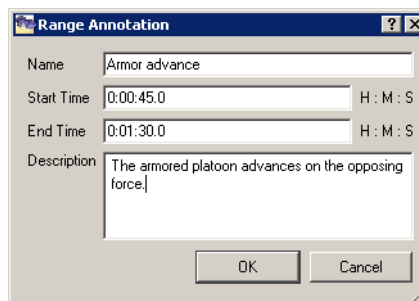


Figure 5-11. Range Annotation dialog box

3. In the Name text box, type a name for the annotation. The name is displayed in the annotation bar.
4. For a point annotation, enter the time at which the annotation should be placed. For a range annotation, enter a start time and an end time for the annotation's time range. 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 5-12).

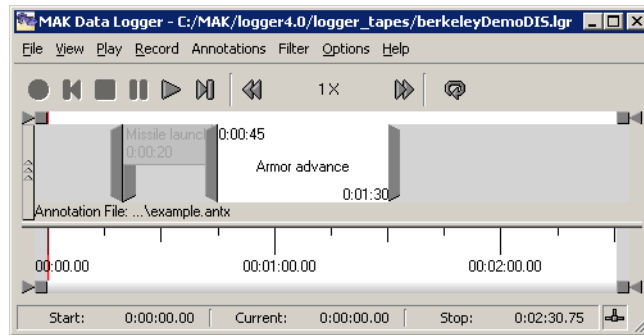


Figure 5-12. Annotations in display area

Annotations can be as long as you want.

### 5.8.2. 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 5-15.
2. Choose Annotations → Save. The Save Annotation File dialog box opens.
3. Type a name for the annotation file.
4. Click Save.

### 5.8.3. 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 recording with Play on Load enabled and the recording 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.

### **5.8.4. 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.

### **5.8.5. Selecting an Annotation**

- To select an annotation, click it.

### **5.8.6. Editing an Annotation**

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

To edit an annotation:

1. 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 anyplace inside an annotation's range and drag the entire annotation to a new location.

### **5.8.7. Deleting an Annotation**

You can delete an annotation from an annotation file or an unsaved list.

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.

### **5.8.8. Jumping to the Start of an Annotation**

You can jump to an annotation or the start time of a range annotation.

To jump to an annotation:

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

#### **Jumping to the End of an Annotation**

You can jump to the end time of a range annotation.

To jump to an annotation:

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

### 5.8.9. Zooming the Timeline to an Annotation

You can zoom the timeline view to an annotation. For example, in [Figure 5-13](#), in the first view, the time line shows the entire length of the recording 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.

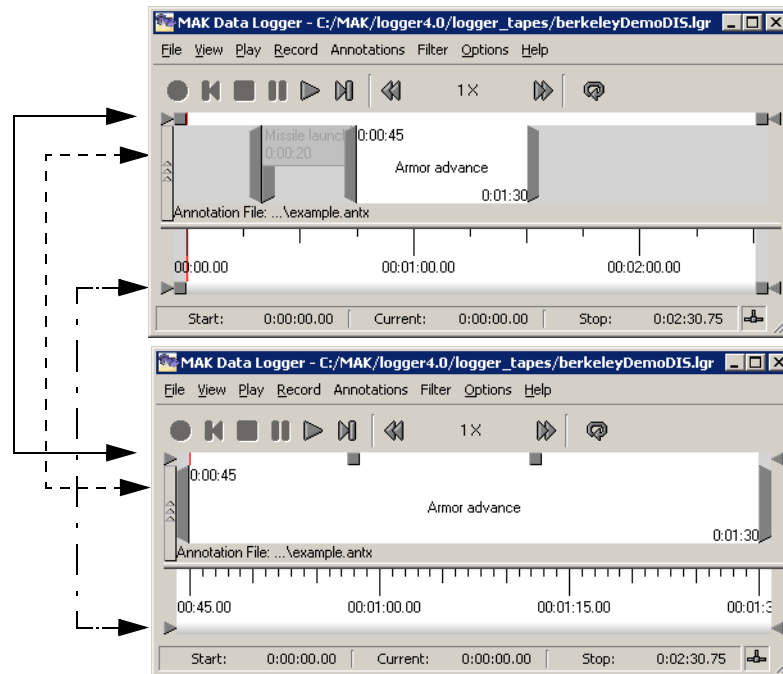


Figure 5-13. Zooming an annotation

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 5-13](#)).

### 5.8.10. Bounding Playback to an Annotation

You can set the playback bar to the range of an annotation. For example, in [Figure 5-14](#), in the first view, the playback handles are at the start and end of the recording. 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 5-13](#), 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 playback segment.)

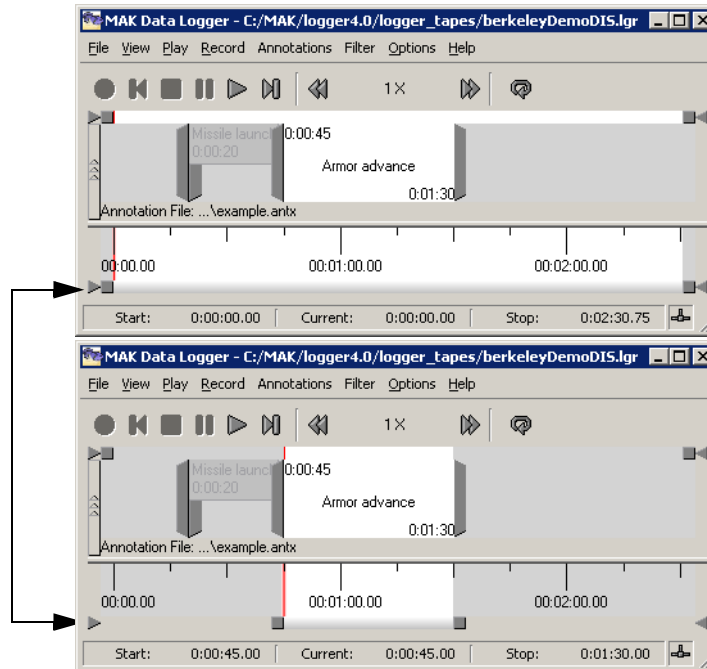


Figure 5-14. Bounding playback to an annotation

To set the playback bar 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 5-14](#)).

## 5.9. Viewing Object Statistics

The Logger can display statistics about the objects in an exercise. Statistics are displayed in the display area above the Logger timeline (Figure 5-15).

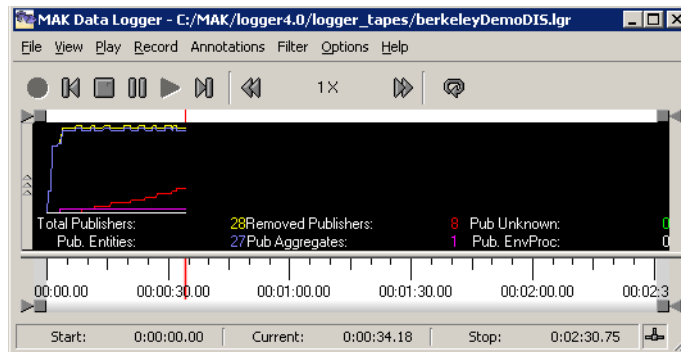


Figure 5-15. Object statistics

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

## 5.10. Summary of Command-line Options for the Logger

Table 1 lists command-line options. Command-line options override the comparable settings in the *lgrConfig.xml* file. They are grouped by mode and indicate if they are for HLA only or DIS only.

Table 5-1: Command-line options (Sheet 1 of 2)

Option	Description
<code>-c mode</code>	Enables (1) or disables (0) display of a console window.
<code>filename</code>	Specifies a Logger recording to load.
<code>{--   --ignore_rest}</code>	Ignore all arguments after this one.
<code>{-h   --help}</code>	Displays a brief summary of Logger command-line arguments.
<code>{-l   --logFileName} filename</code>	Specifies a file to send notification messages to. Default: <i>logger.log</i> .
<code>{-n   --notifyLevel} level</code>	Specifies the notification level for the log file. Range: 0-4.
<b>HLA Only</b>	
<code>{-f   --fomMapperLib} 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   --fedFileName} fedfile</code>	Specifies the FED file for the exercise.

Table 5-1: Command-line options (Sheet 2 of 2)

Option	Description
{-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>	Sets federation execution name. Default: VR-Link.
<b>DIS Only</b>	
{-A   --destAddrString} <i>address</i>	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. See <a href="#">"Specifying A Destination Address for Outgoing and Incoming Data,"</a> on page 3-9.
{-a   --appNumber} <i>appnumber</i>	Specifies the application number.
--disVersion <i>num</i>	Sets the DIS protocol version for PDUs sent onto the network. Set <i>num</i> to:
{-e   --exerciseId} <i>ID</i>	Sets the exercise ID.
--mcastTtl <i>tTl</i>	Specifies the multicast time-to-live.
{-P   --disPort} <i>port</i>	Specifies which DIS port (UDP port if -A not specified, otherwise, TCP) the Logger uses to send or receive data. Default: 3000.
--recvBufferSize <i>size</i>	Specifies the receive buffer size.
{-S   --mcastAddresses} <i>addresses</i>	Subscribes the Logger to the specified multicast address. To subscribe to several multicast addresses, use this option more than once on the command line.
--sendBufferSize <i>size</i>	Specifies the send buffer size.
{-s   --siteId} <i>ID</i>	Specifies the site ID.



## *Using The Text Interface*

---

The text (command-line) interface lets you issue Logger commands from a console window.

Using the Text Interface .....	6-2
Entering Commands .....	6-2
Displaying the List of Commands .....	6-3
Commands by Domains .....	6-4
System Commands .....	6-4
Playback Commands .....	6-4
Record Commands .....	6-5
Simulation Commands .....	6-6
Command List .....	6-7

## 6.1. Using the Text Interface

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

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

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

### 6.1.1. 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
```

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.

### 6.1.2. Displaying the 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) ...
```

Command arguments are order specific, so the order in which they are printed is the order to be entered.

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

#### Argument Types

[Table 6-1](#) describes the argument types used for commands.

Table 6-1: Argument types

Argument Type	Description
int	32 bit signed integer. For example: SetNotifyLevel 2
int64	64 bit signed integer. For example: SetStepSize 1000000
double	64 bit floating point number. For example: SetTimeoutRate 5.0
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).

## 6.2. Commands by Domains

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

- ♦ System
- ♦ Playback
- ♦ Record
- ♦ Simulation, including HLA and DIS-specific commands.

Command argument types are in parentheses following each argument.

### 6.2.1. System Commands

The commands in the system domain are as follows:

DeleteFile *filename* (string) – Delete the specified file.



This command is considered dangerous and should only be used with caution.

---

Exit – Shut down the Logger.

Help – Display a list of available commands.

LoadPlugin *pluginName* (string) – Load the specified plug-in.

SetNotifyLevel *vlLevel* (int) *lgrLevel* (int) – Set the notification level of VR-Link and the Logger.

### 6.2.2. Playback Commands

The commands in the playback domain are as follows:

ClosePlaybackFile – Close the current playback file.

OpenFile *filename* (string) – Open the specified file for playback.

Pause – Pause playback. This command has no effect if the Logger does not have a file open or is already paused.

Play – Start playback. If the Logger is already playing this command has no effect.

PlayPause – Toggle between the paused and play state.

Restart – Start playing from the beginning of the recording.

SetCurrentTime *time* (time) – Jumps playback to the specified time.

SetLoopAtEnd *value* (bool) – Enables or disables looping to start at the end of playback.

SetPlaybackSpeed *speed* (double) – Sets the playback speed of the Logger.

**SetPlayOnLoad** *value* (bool) – Enables starting playback immediately after opening a file.

**SetStartStopTimes** *startTime* (time) *stopTime* (time) – Set 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.

**StopPlayback** – Stops playback. This command has no effect if the Logger is not in the playback state.

**Unpause** – Unpause the Logger. This command has no effect if the Logger was not paused.

### 6.2.3. Record Commands

The commands in the recording domain are as follows:

**CloseRecordFile** – Closes the file currently recording.

**NewFile** *filename* (string) *overwrite* (bool) – Opens a new file for recording.

**Record** – Starts recording to a new file. This command has no effect if a new file is not opened.

**SetCheckpointParameters** *policy* (int) {*size* (int) | *count* (int) | *time* (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.

**SetRecordOnNew** *value* (bool) – Enables or disables starting recording immediately after opening a new file.

**SetStopCondition** *condition* (int) – Specifies the stop condition for recording as follows:

- ♦ 0 = no condition
- ♦ 1 = duration (enables SetStopDuration)
- ♦ 2 = filesize (enables SetStopSize).

**SetStopDuration** *duration* (time) – Stop recording at the specified time if SetStopCondition is set to 1.

**SetStopSize** *filesize* (int64) – Stop recording when the file reaches filesize size if SetStopCondition is set to 2.

**StopRecording** – Stops recording data to a file. The file remains open and recording may be restarted.

## 6.2.4. Simulation Commands

The protocol-independent commands in the simulation domain are as follows:

**Connect** – Connect to the network simulation using the previously set parameters.

**Disconnect** – Disconnect from the network simulation.

**SetRemoteControlSettings** *process* (bool) *record* (bool) *allowDelete* (bool) – Configure remove control set

### HLA-Specific Commands

This section describes the simulation domain commands that can only be used with the HLA Logger. Examples of using the **SetConnectionParameters** command are in *./data/config/lgrConfigHLA13.xml* and *./data/config/lgrConfigHLA1516.xml*.

**SetConnectionParameters** *federateName* (string) *federationName* (string) *fedFile* (string) *fomVersion* (double) *fomMapper* (string) *fomMapperData* (base64) – Specifies HLA connection parameters.

**SetIgnoreAdvisories** *mode* (bool) – Specifies the Logger's response to HLA advisory messages. You must specify this command before the **connect** command. Default: false.

**SetPassiveSubscription** *mode* (bool) – configure passive subscription for HLA. You must specify this command before the **connect** command. Default: false.

### DIS-Specific Commands

This section describes the simulation domain commands that can only be used with the DIS Logger. For an example of using the **SetConnectionParameters** command are in *./data/config/lgrConfigDIS.xml*.

**AddCustomPDU** *pduName* (string) *pduKind* (string) *entityIdOffset* (string) *entityTypeOffset* (string) – Specifies a PDU to add to the Filter by PDU Kind dialog box.

**SetCommonEntityTypes** *entityType1* ... *entityTypen* (string) – Specifies a list of entity types to list in the Filter by Entity Type dialog box.

**SetConnectionParameters** *port* (int) *exerciseId* (int) *siteId* (int) *appNumber* (int) *useAsynchIO* (bool) *destAddr* (int) *sendBufferSize* (int) *recvBufferSize* (int) *multicastTtl* (int) *mcastAddresses* (stringList) – Specifies parameters for the DIS exercise connection.

**SetHeartbeatRate** *rate* (double) – Specifies the DIS heartbeat rate.

**SetTimeoutRate** *rate* (double) – Specifies the DIS timeout rate.

## 6.3. Command List

Table 6-2 section lists all Logger commands and their domain:

Table 6-2: Logger commands

Command	Domain
AddCustomPDU (DIS only)	Simulation
ClosePlaybackFile	Playback
CloseRecordFile	Record
Connect	Simulation
DeleteFile	System
Disconnect	Simulation
Exit	System
Help	System
LoadPlugin	System
NewFile	Record
OpenFile	Playback
Pause	Playback
Play	Playback
PlayPause	Playback
Record	Record
Restart	Playback
SetCheckpointParameters	Record
SetCommonEntityTypes (DIS only)	Simulation
SetConnectionParameters (Different versions for HLA and DIS)	Simulation
SetCurrentTime	Playback
SetHeartbeatRate (DIS only)	Simulation
SetIgnoreAdvisories (HLA only)	Simulation
SetLoopAtEnd	Playback
SetNotifyLevel	System
SetPassiveSubscription (HLA only)	Simulation
SetPlaybackSpeed	Playback
SetPlayOnLoad	Playback
SetRecordOnNew	Record
SetRemoteControlSettings	Simulation

Table 6-2: Logger commands

Command	Domain
SetStartStopTimes	Playback
SetStopCondition	Record
SetStopDuration	Record
SetStopSize	Record
SetTimeoutRate (DIS only)	Simulation
StopPlayback	Playback
StopRecording	Record
Unpause.	Playback



The following commands are displayed by the Help command. However these commands are not supported for use by customers.

---

- ♦ IssueTextCommand
- ♦ RequestUpdateCommand
- ♦ SetDisDescription
- ♦ SetEntityTypeManager
- ♦ SetFilterSettings
- ♦ SetTimeManagement.



## *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 .....	7-3
The Logger Architecture .....	7-4
Drivers .....	7-4
Factories .....	7-5
Interfaces .....	7-5
The Logger Message Pipeline.....	7-6
The Simulation Manager.....	7-6
Embedding the Logger in an Application .....	7-7
The playFile Example .....	7-7
Creating a Stand-Alone Logger Application .....	7-10
The Logger Plug-in API .....	7-10
Writing a Plug-In .....	7-11
Loading Plug-Ins .....	7-12
Logger API Coding Conventions .....	7-13
Namespaces .....	7-13
Templates .....	7-13
Shared Pointers.....	7-14
Virtual Constructors.....	7-14
Runtime Values .....	7-15
Thread Safety .....	7-16
Interface Signal Slots .....	7-16
Observer Pattern.....	7-17
Logger Libraries .....	7-17
Other Libraries and Plug-ins .....	7-18

Compiling and Linking with the Logger API.....	7-18
Compiling and Linking for UNIX .....	7-18
Compiling and Linking for Microsoft Windows .....	7-19

## 7.1. 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 Qt 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.

## 7.2. The Logger Architecture

The Logger architecture is highly modular and designed to be independent of the simulation protocol, user interface, and execution (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.

### 7.2.1. 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.

### 7.2.2. 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*.

### 7.2.3. 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.

### 7.2.4. 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.

### 7.2.5. 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 call-backs. It also provides access to creating simulation-specific objects, such as the simulation input and output stages, and opens files for a specific format.

## 7.3. 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 two examples of using *DtLogger* in a custom application: *./examples/loggerAPI/playFile.cxx* and *./examples/loggerAPI/writeFile.cxx*.

### 7.3.1. 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;
```

## 7.4. 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.

## 7.5. 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 is an example of how to write a plug-in for the Data Logger application. It shows the use of many techniques employed by the Logger API. The source code for this plug-in example is in *./examples/stateObserver*.

### 7.5.1. 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;
} }
```

### 7.5.2. 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();
```

## 7.6. Logger API Coding Conventions

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

### 7.6.1. 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())
{
}
```

### 7.6.2. Templates

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

### 7.6.3. 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
}
```

### 7.6.4. 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*.

### 7.6.5. 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.

### 7.6.6. 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;
 }
}
```

### 7.6.7. 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);
```

### 7.6.8. 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.

## 7.7. Logger Libraries

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

Table 7-1: 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.

### 7.7.1. 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*.

## 7.8. 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.

---

### 7.8.1. 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$(MAK_VRLDIR)/lib -lvlHLA13 -lmatrix -lmtl -lvlutil
-L$(MAK_RTIDIR)/lib -lRTI-NG -lfedtime
```

For HLA 1516, link against:

```
-LloggerDir/lib -lliblgrUtil -lliblgrCore -lliblgrApp
-llibappEnv -lliblgrGui
loggerDir/lib/lgrSimGuiHLA1516.so -L$(MAK_VRLDIR)/lib
-lvlHLA1516 -lmatrix -lmtl -lvlutil -L$(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 -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.

## 7.8.2. 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 matrixRT.lib mtlRT.lib vlutilRT.lib (VR-
Link libraries)
librti1516.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)
v1HLA13RT.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)
v1DISRT.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.

<a href="#">Remote Control Messages .....</a>	<a href="#">8-2</a>
<a href="#">Deleting Logger Files .....</a>	<a href="#">8-3</a>
<a href="#">Example of a Remote Control Application .....</a>	<a href="#">8-3</a>

## 8.1. 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*.

## 8.2. 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
```

## 8.3. Example of a Remote Control Application

The following example demonstrates programmatic remote control of the Logger. For the complete example, 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
}
```

```
// 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;
 }
}
```

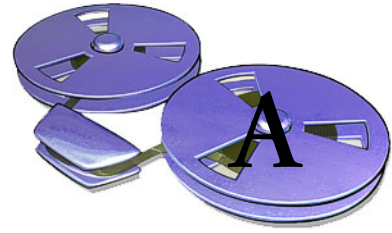
```
 }
 #if !DtHLA
 case DtLgrHeartbeat:
 {
 DtLgrHeartbeatPdu pdu;
 pdu.setHeartbeat(atof(optionalArg));
 sendPdu(exConn, &pdu, target, myId);
 break;
 }
 case DtLgrTimeout:
 {
 DtLgrTimeoutPdu pdu;
 pdu.setTimeout(atof(optionalArg));
 sendPdu(exConn, &pdu, target, myId);
 break;
 }
 #endif
 }
}
```



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.

<a href="#">Troubleshooting Logger Network Communications (DIS Only).....</a>	<a href="#">A-2</a>
<a href="#">Using the Correct Broadcast Address and UDP Port .....</a>	<a href="#">A-2</a>

## **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, 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
```



## ***The Logger Configuration File***

---

This describes the default settings in *lgrConfig.xml* and explains how to add commands to the file.

<a href="#">Editing lgrConfig.xml</a> .....	B-2
<a href="#">Adding a Command to lgrConfig.xml</a> .....	B-4
<a href="#">Logger Configuration Parameters</a> .....	B-5

## B.1. 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.

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 of the commands from the text interface in a configuration file.

The following is a heavily edited version of a configuration file 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"/>
 </commands>
</lgrConfig>
```

## Command Syntax

Commands that do not take arguments have the format:

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

where:

- ♦ *domain* is System, Playback, Record, or Simulation. Please see “[Commands by Domains](#),” on page 6-4 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 6, [Using The Text Interface](#).

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>
```

- ♦ *name* is the name of a command argument as listed in the command descriptions in “[Commands by Domains](#),” on page 6-4.
- ♦ *variableName* is the name of a variable 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

Variables entries take the format:

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

where:

- ♦ *variableName* is an user-defined name for this variable.
- ♦ *type* is the argument type for the variable, as defined for each command variable in “[Commands by Domains](#),” on page 6-4.
- ♦ *value* is the value you want to assign to the variable.

For example:

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

### B.1.1. 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="Record" protocol="None" command="NewFile">
 <param name="filename" var="newfile"></param>
 </command>
</commands>
```

In the command definition, the domain is Record, which you know because NewFile is listed with the commands in the Record domain in “[Commands by Domains](#),” on page 6-4. 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 param 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 you used (or plan to use) 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.

## B.2. Logger Configuration Parameters

Table B-1 describes the parameters in the default *lgrConfig.xml* files.

Table B-1: Logger configuration parameters (Sheet 1 of 2)

Parameter	Description
annotationPlugin	Specifies the library for the annotation plug-in.
locale	Specifies the two-character string for the language locale if you are using a translation file for the GUI.
logFileName	Specifies a file to send notification messages to. Default: <i>logger.log</i> .
messageLevel	Specifies the verbosity of VR-Link notification messages. Range: 0-4. Equivalent to the first value used in the <code>-n</code> command-line option.      ♦ 0 = Fatal errors only     ♦ 1 = Quiet     ♦ 2 = Normal     ♦ 3 = Verbose     ♦ 4 = Debug.      Default: 2.
notifyLevel	Specifies the notification level for Logger messages. Equivalent to the second value used in the <code>-n</code> startup option.
protocolGuiLibrary	Specifies the library that has the GUI libraries for the protocol used by this version of the Logger.
protocolLibrary	Specifies the library that has the libraries for the protocol used by this version of the Logger.
showConsole	Enables the console for the text interface.
stateObserverPlugin	Specifies the library for the state observer plug-in.
<b>HLA Only</b>	
execName	Specifies the name of the federation execution.
federateName	Specifies the name of the Logger federate.
fedFileName	Specifies the name of the FED file to use.
fomMapperInitData	Specifies initialization data to pass to a FOM Mapper library.
fomMapperLib	Specifies a FOM Mapper library to use.
rprFomVersion	Specifies a version of the RPR FOM to use.
<b>DIS Only</b>	
appNumber	Specifies the application number.
commonEntityType	Specifies a list of entity type enumerations to include in the common entity types list in the Filter by Entity Type dialog box.

Table B-1: Logger configuration parameters (Sheet 2 of 2)

Parameter	Description
customPdus	Specifies a list of custom PDUs to be added to the list of PDUs in the Filter by PDU Kind dialog box.
destAddrString	Specifies the IP address to which the Logger sends outgoing data packets. Default: Host machine's default broadcast address.
disPort	Specifies the UDP port over which the Logger sends and receives data. (If used with destAddr, specifies TCP port.) Default: 3000.
exerciseld	Specifies the exercise ID for the exercise.
mcastAddresses	Specifies one or more multicast addresses.
mcastTtl	Specifies the multicast time to live for messages.
recvBufferSize	Specifies the size of the buffer in which the Logger receives data packets. Default: 64000.
sendBufferSize	Specifies the size of the buffer in which the Logger stores data packets for sending. Default: 64000.
siteld	Specifies the DIS site ID.
useAsyncIO	Enables (1) or disables (0) use of asynchronous I/O. Default: 0.



# ***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.

**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](http://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.

**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 5-21
- appNumber command-line option 5-22
- destAddrString command-line option 5-22
- disPort command-line option 5-22
- disVersion command-line option 5-22
- execName command-line option 5-22
- exerciseId command-line option 5-22
- federateName command-line option 5-22
- fomMapperInitData command-line option 3-8, 5-21
- fomMapperLib command-line option 5-21
- ignore\_rest command-line option 5-21
- logFileName command-line option 5-21
- mcastAddresses command-line option 5-22
- mcastTtl command-line option 3-11, 5-22
- notifyLevel command-line option 5-21
- receiveBufferSize command-line option 3-10
- recvBufferSize command-line option 5-22
- rprFomVersion command-line option 3-7, 5-22
- sendBufferSize command-line option 3-10, 5-22
- siteId command-line option 5-22
- A command-line option 3-9, 5-22
- a command-line option 5-22
- e command-line option 3-11, 5-22
- F command-line option 3-7, 5-21
- f command-line option 3-8, 5-21
- h command-line option 5-21
- l command-line option 5-21
- N command-line option 3-8, 5-22
- n command-line option 3-6, 5-21, B-5
- P command-line option 3-10, 5-22
- S command-line option 3-10, 5-22
- s command-line option 5-22
- x command-line option 3-7, 5-22

## A

- acceleration scaling 4-5
- addCommandHandler() function 7-4
- AddCustomPDU command 6-6
- adding
  - annotation 5-15
  - command to *lgrConfig.xml* B-4
  - new license file 2-11
  - variable to *lgrConfig.xml* B-4
- address
  - multicast 5-22
- advisory message
  - ignoring 3-9
- annotating
  - recording 5-15
- annotation 5-15
  - bounding playback to 5-20
  - creating 5-15
  - deleting 5-18
  - editing 5-17
  - importing annotation file 5-17
  - loading automatically 5-6
  - opening annotation file 5-16
  - saving file 5-16
  - selecting 5-17
  - zooming timeline to 5-19
- annotationPlugin B-5
- API
  - Logger 7-3
  - plug-in 7-10
- appEnv library 7-18
- application
  - compiling and linking 7-18
  - embedding Logger in 7-7, 7-10

application number

specifying 5-22

**appNumber** parameter B-5

architecture

Logger API 7-4

argument type 6-3

asynchronous I/O B-6

configuring 3-11

attribute

annotation 5-17

connection 3-4

automatic

playback 5-6

recording 5-5

## B

beginning

moving to 5-10

broadcast address 3-9, A-2, B-6

buffer B-6

send and receive 3-10

button bar 5-4

## C

changing

connection settings 3-4

checkpoint frequency 3-6

class

*DtAnyValue* 7-15

*DtCommand* 7-5

*DtCommandDefinition* 7-5

*DtCommandSP* 7-14

*DtDisLogger* 7-7

*DtHlaLogger* 7-7

*DtLgrAppGuiInterface* 7-17

*DtLgrBaseSimManager* 7-6

*DtLgrCommandHandler* 7-4

*DtLgrControlInteraction* 8-2

*DtLgrControlPdu* 8-2

*DtLgrDelFilePdu* 8-3

*DtLgrDisSimulationManager* 7-6

*DtLgrFullInterface* 7-5

*DtLgrHlaSimulationManager* 7-6

*DtLgrInterface* 7-3

*DtLgrPlaybackCommand Factory* 7-5

*DtLgrPlaybackDriver* 7-4

*DtLgrPlaybackInterface* 7-5

*DtLgrProtocolRegistration* 7-6

*DtLgrSimInput* 7-6

*DtLgrSimOutput* 7-6

*DtLgrSystemDriver* 7-4

*DtLogger* 7-3, 7-4, 7-6, 7-7, 7-10, 7-11, 7-12

*DtLoggerApplication* 7-10, 7-11

*DtMessageInput* 7-6

*DtMessageOutput* 7-6

*DtMessageState* 7-6

*DtMutex* 7-16

*DtObserveConnection* 7-17

*DtObserveEvent* 7-17

*DtObserver* 7-17

*DtProtocolRegistration* 7-17

*DtResponse* 7-5, 7-14

*DtSimInput* 7-6

*DtString* 7-15

*DtThreadSafe* 7-16

*DtValue* 7-15

*DtValueSignal* 7-16

factory 7-5

filtering by (HLA) 5-14

interface 7-5

remote control 8-2

ClosePlaybackFile command 6-4

CloseRecordFile command 6-5

closing

recording 5-8

command

AddCustomPDU 6-6

adding to *lgrConfig.xml* B-4

ClosePlaybackFile 6-4

CloseRecordFile 6-5

Connect 6-6

defining, creating, and decoding 7-5

DeleteFile 6-4

Disconnect 6-6

Exit 6-4

Help 6-4

list of 6-7

LoadPlugin 6-4

NewFile 6-5

OpenFile 6-4

Pause 6-4

Play 6-4

PlayPause 6-4

Record 6-5

Restart 6-4

sending 7-5

SetCheckpointParameters 6-5

SetConnectionParameters 6-6

- SetCurrentTime 6-4
- SetHeartbeatRate 6-6
- SetIgnoreAdvisories 6-6
- SetLoopAtEnd 6-4
- SetNotifyLevel 6-4
- SetPassiveSubscription 6-6
- SetPlaybackSpeed 6-4
- SetPlayOnLoad 6-5
- SetRecordOnNew 6-5
- SetRemoteControlSettings 6-6
- SetStartStopTimes 6-5
- SetStopCondition 6-5
- SetStopDuration 6-5
- SetStopSize 6-5
- SetTimeoutRate 6-6
- StopPlayback 6-5
- StopRecording 6-5
- syntax in *lgrConfig.xml* B-3
- Unpause 6-5
- command line
  - installation 2-4
- command-line interface 6-2
  - domains 6-4
- command-line option 3-3
  - 5-21
  - A 3-9, 5-22
  - a 5-22
  - appNumber 5-22
  - destAddrString 5-22
  - disPort 5-22
  - disVersion 5-22
  - e 3-11, 5-22
  - execName 5-22
  - exerciseId 5-22
  - F 3-7, 5-21
  - f 3-8, 5-21
  - federateName 5-22
  - filename 5-21
  - fomMapperInitData 3-8, 5-21
  - fomMapperLib 5-21
  - h 5-21
  - ignore\_rest 5-21
  - l 5-21
  - logFileName 5-21
  - mcastAddresses 5-22
  - mcastTtl 3-11, 5-22
  - N 3-8, 5-22
  - n 3-6, 5-21, B-5
  - notifyLevel 5-21
  - P 3-9, 3-10, 5-22
  - receiveBufferSize 3-10
  - recvBufferSize 5-22
  - rprFomVersion 3-7, 5-22
  - S 3-10, 5-22
  - s 5-22
  - sendBufferSize 3-10, 5-22
  - siteId 5-22
  - table of 5-21
  - x 3-7, 5-22
- commonEntityTypes** parameter 5-12, B-5
- compiling
  - API code 7-18
  - application 7-18
- component files 2-6
- configuration file
  - RTI 2-16
- configuration parameter 3-3
  - recvBufferSize 3-10
  - rprFomVersion 3-8
  - useAsyncIO 3-11
- configuring
  - asynchronous I/O 3-11
  - DIS 3-9
  - FED file 5-21
  - indexing 3-6
  - list of common entity types 5-12
  - Logger 3-3
  - Logger for HLA 3-7
  - send and receive buffer size 3-10
  - system for MÄK RTI 2-16
- Connect command 6-6
- connecting
  - to exercise 3-4
- connection
  - changing settings 3-4
- Connection Settings 3-3
- connection status 3-3
- console 6-2
- control
  - Logger 5-4
- conventions
  - manual xiv
- create() function 7-5, 7-15
- creating
  - annotation 5-15
  - command 7-5
- cursor
  - timeline 5-10
- custom PDU B-6
- filtering 5-13

`customPdu` parameter 5-13, B-6

## D

dead reckoning

acceleration scaling 4-5

velocity scaling 4-5

`decode()` function 7-5

decoding

command 7-5

defining

command 7-5

DeleteFile command 6-4

deleting

annotation 5-18

Logger file 8-3

`destAddrString` parameter 3-9, B-6

destination address 3-9

directory

`./bin` 2-6

`./doc` 2-6

`./logger_tapes` 2-6

DIS 1-5

asynchronous I/O 3-11

configuring Logger for 3-9

exercise ID 3-11

heartbeat 3-5

protocol version 5-22

site ID B-6

timeout 3-5

DIS port 5-22

Logger default A-2

specifying 5-22, A-2

disabling

Play On Load 5-6

Record On New 5-5

Disconnect command 6-6

disconnecting

from exercise 3-4

displaying

object statistics 5-21

`disPort` parameter 3-10, B-6

domain

associated with command 6-7

command 6-4

dragging

timeline cursor 5-10

driver

in Logger API 7-4

*DtAnyValue* class 7-15

*DtCommand* class 7-5

*DtCommandDefinition* class 7-5

*DtCommandSP* class 7-14

*DtDisLogger* class 7-7

*DtHlaLogger* class 7-7

*DtLgrAppGuiInterface* class 7-17

*DtLgrBaseSimManager* class 7-6

*DtLgrCommandHandler* class 7-4

*DtLgrControllInteraction* class 8-2

*DtLgrControlPdu* class 8-2

*DtLgrDelFilePdu* class 8-3

*DtLgrDisSimulationManager* class 7-6

*DtLgrFullInterface* class 7-5

*DtLgrHlaSimulationManager* class 7-6

*DtLgrInterface* class 7-3

*DtLgrPlaybackCommand Factory* class 7-5

*DtLgrPlaybackDriver* class 7-4

*DtLgrPlaybackInterface* class 7-5

*DtLgrProtocolRegistration* class 7-6

*DtLgrSimInput* class 7-6

*DtLgrSimOutput* class 7-6

*DtLgrSystemDriver* class 7-4

*DtLogger* class 7-3, 7-4, 7-6, 7-7, 7-10, 7-11, 7-12

*DtLoggerApplication* class 7-10, 7-11

*DtMessageInput* class 7-6

*DtMessageOutput* class 7-6

*DtMessageState* class 7-6

*DtMutex* class 7-16

*DtObserveConnection* class 7-17

*DtObserveEvent* class 7-17

*DtObserver* class 7-17

*DtProtocolRegistration* class 7-17

*DtResponse* class 7-5, 7-14

*DtSimInput* class 7-6

*DtString* class 7-15

*DtThreadSafe* class 7-16

*DtValue* class 7-15

*DtValueSignal* class 7-16

## E

editing

annotation 5-17

`lgrConfig.xml` B-2

point annotation 5-17

range annotation 5-17

embedding

Logger in application 7-7, 7-10

`EnableLoggerAppPlugin()` function 7-11

`EnableLoggerPlugin()` function 7-11

- enabling
  - Play On Load 5-6
  - Record On New 5-5
- end
  - jumping to range annotation 5-18
  - moving to 5-10
- ending
  - recording 5-5
- end-of-file
  - setting 5-7
- entity
  - common types 5-12
  - filtering by type 6-6
  - maintenance 4-5, 5-10
- entity type
  - filtering by 5-11, B-5
  - removing from filtering list 5-12
- environment variable
  - MAKLMGRD\_LICENSE\_FILE 2-10
  - RTL\_CONFIG 2-16
- example
  - API 7-7
  - plug-in 7-10
  - remote control 8-3
- execName** parameter 3-7, B-5
- exercise
  - connecting to 3-4
  - connection 3-3
  - disconnecting from 3-4
  - ID 3-11, B-6
  - recording 5-5
- exerciseld** parameter 3-11, B-6
- Exit command 6-4
- expanding
  - playback segment 5-7

## F

- factory 7-5
- FDD file 2-16
- FED file 1-5, 2-16
  - changing 4-6
  - specifying 3-7, 3-8, 5-21, B-5
- federate
  - specifying Logger federate name 3-8
  - specifying name B-5
- federateName** parameter 3-8, B-5
- federation execution 1-5
  - name 3-7, 5-22
  - specifying B-5
- fedFileName** parameter 3-7, B-5
- file
  - annotating 5-15
  - closing 5-8
  - FED 2-16, 3-8
  - lgrConfig.xml* 3-3
  - license 2-10
  - loading 5-6
  - playing 5-6
  - RID 2-16
  - rti.mtl* 2-16
  - RTL.RID* 2-16
  - VR-Link.fed* 3-7
- filename command-line option 5-21
- Filter By Class dialog box 5-14
- Filter By Entity Type dialog box 5-11
- Filter By PDU Kind dialog box 5-13
- filtering
  - by entity type 6-6
  - by PDU kind 6-6
  - classes (HLA) 5-14
  - configuring common entity types 5-12
  - custom PDUs 5-13
  - entity type 5-11, B-5
  - input and output 5-11
  - PDU kind 5-13
- FLEXlm
  - directory location 2-8
  - installing 2-8
  - license file 2-10
  - license manager 2-7
  - license server name 2-9
  - multiple licenses 2-11
  - troubleshooting 2-14
- FOM 2-16
  - flexibility 4-6
  - supported 1-5
- FOM Mapper 1-5, 3-8, 5-21, B-5
  - specifying at startup 3-7
- FOM Mapper initialization data 5-21
- fomMapperInitData** parameter 3-8, B-5
- fomMapperLib** parameter 3-8, B-5
- function
  - `addCommandHandler()` 7-4
  - `create()` 7-5, 7-15
  - `decode()` 7-5
  - `EnableLoggerAppPlugin()` 7-11
  - `EnableLoggerPlugin()` 7-11
  - `LoadPlugin()` 7-11
  - `setCreator()` 7-15

tick() 7-4, 7-7  
UnloadPlugin() 7-11  
update() 7-5, 7-8

## **G**

GUI  
    localizing 2-17, B-5

## **H**

handle  
    start and end time 5-7  
header file  
    *lgrBaseSimManager.h* 7-6  
    *lgrCommand.h* 7-5  
    *lgrDisSimManager.h* 7-6  
    *lgrDriver.h* 7-4  
    *lgrHlaSimManager.h* 7-6  
    *lgrPlaybackCommandFactory.h* 7-5  
    *lgrPlaybackDriver.h* 7-4  
    *lgrPlaybackInterface.h* 7-5  
    *lgrProtocolRegistration.h* 7-6  
    *lgrRuntimeCommandFactory.h* 7-5  
    *lgrSimInput.h* 7-6  
    *lgrSimOutput.h* 7-6  
    *lgrSystemDriver.h* 7-4  
    *logger.h* 7-14  
    *messageInput.h* 7-6  
    *messageOutput.h* 7-6  
    *messageState.h* 7-6  
    *observeConnection.h* 7-17  
    *observeEvent.h* 7-17  
    *observe.h* 7-17  
    *runtimeValue.h* 7-15  
    *stateObController.h* 7-5  
    *stateObModel.h* 7-4  
    *virtualConstructor.h* 7-15  
heartbeat  
    defined 4-4  
    setting 3-5  
Help command 6-4  
HLA 1-5  
    advisory messages 3-9  
    configuring for 3-7  
    FOM flexibility 4-6  
    specifications supported 1-5  
    subscribing passively 3-9  
    support for 4-6  
host ID

getting 2-9  
host name 2-9

## **I**

IEEE 1516 1-5  
ignoring  
    advisory messages 3-9  
immediate  
    playback 5-6  
    recording 5-5  
importing  
    annotation file 5-17  
indexing  
    configuring 3-6  
informational message 3-6  
input  
    filtering 5-11  
input stage 7-6  
installation  
    over network 2-4  
    silent 2-4  
installing  
    FLEXlm software 2-8  
    RTI 2-16  
interaction  
    subscribing to 3-9  
interface  
    class 7-5  
    signal slot 7-16  
    text 6-2  
interim PDU 4-4, 4-5  
IP address 5-22  
    outgoing packets B-6

## **J**

jumping  
    range annotation end 5-18  
    range annotation start 5-18  
    to beginning of recording 5-10  
    to end of recording 5-10  
    to specific time 5-10

## **L**

language  
    specifying B-5  
lgrAnnotations library 7-18

- lgrApp library 7-17
  - lgrBaseSimManager.h* header file 7-6
  - lgrCommand.h* header file 7-5
  - lgrConfig.xml 6-2
  - lgrConfig.xml*
    - syntax and editing B-2
  - lgrConfig.xml* configuration file 3-3
  - lgrCore library 7-17
  - lgrDisSimManager.h* header file 7-6
  - lgrDriver.h* header file 7-4
  - lgrGui library 7-17
  - lgrHlaSimManager.h* header file 7-6
  - lgrPlaybackCommandFactory.h* header file 7-5
  - lgrPlaybackDriver.h* header file 7-4
  - lgrPlaybackInterface.h* header file 7-5
  - lgrProtocolRegistration.h* header file 7-6
  - lgrRuntimeCommandFactory.h* header file 7-5
  - lgrSim library 7-17
  - lgrSimGui library 7-17
  - lgrSimInput.h* header file 7-6
  - lgrSimOutput.h* header file 7-6
  - lgrSystemDriver.h* header file 7-4
  - lgrUtil library 7-17
  - library
    - API 7-17
    - appEnv 7-18
    - lbrUtil 7-17
    - lgrAnnotations 7-18
    - lgrApp 7-17
    - lgrCore 7-17
    - lgrGui 7-17
    - lgrSim 7-17
    - lgrSimGui 7-17
    - Logger API 7-18
    - plug-ins 7-11
  - license file 2-10
    - adding new 2-11
  - license server
    - name 2-9
    - running 2-12
    - status 2-12
    - stopping 2-12
    - troubleshooting 2-14
  - linking
    - API code 7-18
    - application 7-18
  - lmdown program 2-12
  - lmhostid 2-9
  - lmstat program 2-12
  - loading
    - annotation file automatically 5-6
    - plug-ins 7-12
    - recording 5-6
  - LoadPlugin command 6-4
  - LoadPlugin() function 7-11
  - locale parameter B-5
  - localizing
    - GUI 2-17, B-5
  - log file 3-6, 5-21, B-5
  - logFileName parameter B-5
  - Logger
    - API 7-3
    - embedding in application 7-7, 7-10
    - file
      - deleting 8-3
    - recording
      - samples 1-4
      - viewing with Stealth 1-4
    - window 5-4
  - Logger API
    - architecture 7-4
    - coding conventions 7-13
    - compiling and linking 7-18
    - compiling and linking applications 7-18
    - drivers 7-4
    - embedding Logger 7-7, 7-10
    - examples 7-7
    - libraries 7-17
    - runtime values 7-15
  - Logger Control PDU 8-2
  - Logger State Observer 7-10
  - logger.h* header file 7-14
  - logger.log 5-21, B-5
  - logger.log* 3-6
- ## M
- MÄK RTI
    - configuring 2-16
  - MAKLMGRD\_LICENSE\_FILE environment
    - variable 2-10
  - manual
    - conventions xiv
  - mcastAddresses parameter 3-10, B-6
  - mcastTtl parameter 3-11, B-6
  - message
    - advisory 3-9
    - informational 3-6
    - pipeline 7-6
  - messageInput.h* header file 7-6

- `messageLevel` parameter 3-6, B-5
- `messageOutput.h` header file 7-6
- messages
  - log file 5-21, B-5
- `messageState.h` header file 7-6
- moving
  - to beginning of recording 5-10
  - to end of recording 5-10
- multicast
  - address 5-22
- multicast address 3-10
- multicast time-to-live 3-11, B-6
  - specifying 5-22

## N

- name
  - Logger federate 3-8
- namespace 7-13
- network
  - parameter
    - HLA 3-7
  - requirements for Logger A-2
- networked installation 2-4
- NewFile command 6-5
- notification level B-5
  - setting 3-6
- `notifyLevel` parameter 3-6, B-5

## O

- object
  - displaying statistics 5-21
  - subscribing to 3-9
- `observeConnection.h` header file 7-17
- `observeEvent.h` header file 7-17
- `observe.h` header file 7-17
- observer pattern 7-17
- OpenFile command 6-4
- opening
  - annotation file 5-16
  - recording 5-6
- output
  - filtering 5-11
- output stage 7-6

## P

- P command-line option 3-9, 3-10

- packet
  - preventing dropped 3-11
  - sending and receiving 3-10
- parameter
  - `appNumber` B-5
  - `commonEntityTypes` 5-12, B-5
  - `customPdus` 5-13, B-6
  - `destAddrString` 3-9, B-6
  - `disPort` 3-10, B-6
  - `execName` 3-7, B-5
  - `exerciseld` 3-11, B-6
  - `federateName` 3-8, B-5
  - `fedFileName` 3-7, B-5
  - `fomMapperInitData` 3-8, B-5
  - `fomMapperLib` 3-8, B-5
  - HLA 3-7
  - `locale` B-5
  - `logFileName` B-5
  - `mcastAddresses` 3-10, B-6
  - `mcastTtl` 3-11, B-6
  - `messageLevel` 3-6, B-5
  - `notifyLevel` 3-6, B-5
  - `recvBufferSize` 3-10, B-6
  - `rprFomVersion` 3-7, 3-8, B-5
  - `sendBufferSize` B-6
  - `showConsole` 6-2, B-5
  - `siteld` B-6
  - `useAsyncIO` 3-11, B-6
- passive subscription 3-9
- path
  - specifying for RTI 2-16
- Pause command 6-4
- pausing playback 5-6
- PDU
  - filtering B-6
  - filtering by 5-13, 6-6
  - filtering custom 5-13
  - remote control 8-2
- pipeline 7-6
- Play button 5-6
- Play command 6-4
- Play On Load 5-6
- playback 4-3, 5-6
  - automatic 5-6
  - bounding to annotation 5-20
  - described 4-3
  - loading file 5-6
  - mode 4-3
  - moving to beginning 5-10
  - moving to end 5-10

- pausing 5-6
- segment 5-7
  - expanding 5-7
- setting speed 5-8
- stopping 5-7
- playFile example
  - example
    - playFile 7-7
- playing
  - part of a recording 5-7
  - recording 5-6
- PlayPause command 6-4
- plug-in
  - example 7-10
  - loading 7-12
  - object statistics 5-21
- plug-in API 7-10
- plug-in library 7-11
- point annotation
  - editing 5-17
- Point Annotation dialog box 5-17
- pointer 7-14
- port B-6
  - specifying UDP 3-10
- Portable Document Format (PDF) x
- preferences
  - setting 3-6
- Preferences dialog box 3-6
- products
  - technical support xiii
  - upgrades xiii
- property
  - annotation 5-17
  - connection 3-3
- protocol
  - supported 1-5
  - version
    - DIS 5-22
- protocolGuiLibrary B-5
- protocolLibrary B-5

## Q

- Qt Linguist 2-17

## R

- range annotation
  - bounding playback to 5-20
  - editing 5-17

- jumping to end 5-18
  - jumping to start 5-18
- Range Annotation dialog box 5-17
- receive buffer B-6
  - size
    - specifying 5-22
- receiving
  - packets 3-10
- receiving application responses 7-5
- record
  - mode 4-2
- Record command 6-5
- Record On New 5-5
- recording 4-2
  - annotating 5-15
  - automatically 5-5
  - closing 5-8
  - described 4-2
  - exercise 5-5
  - jumping to beginning 5-10
  - jumping to end 5-10
  - loading 5-6
  - pausing 5-6
  - playing 5-6
  - starting 5-5
  - stopping 5-5
- recvBufferSize parameter 3-10, B-6
- remote control 8-2
  - example 8-3
- response 7-5
- Restart command 6-4
- RID file 2-16
- rid.mtl file 2-16
- RPR FOM 3-7, 4-6, B-5
  - specifying version to use 3-8
- rprFomVersion parameter 3-7, 3-8, B-5
- RTI 2-16
  - definition of 2-16
  - specifications supported 1-5
- RTI\_CONFIG environment variable 2-16
- RTI.rid file 2-16
- runLm program 2-12
- running
  - license server 2-12
- runtime value 7-15
- runtimeValue.h header file 7-15

## S

- sample recording 1-4

- saving
  - annotation file 5-16
  - annotations file 5-16
- scaling
  - time 4-5
- segment
  - playing 5-7
  - setting playback to annotation 5-20
- selecting
  - annotation 5-17
- send buffer B-6
  - size
    - specifying 5-22
- sendBufferSize** parameter B-6
- sending
  - commands 7-5
  - packets 3-10
- Set Time 4-5
  - entity maintenance 5-10
- SetCheckpointParameters command 6-5
- SetCommonEntityTypes command
  - command
    - SetCommonEntityTypes 6-6
- SetConnectionParameters command 6-6
- setCreator() function 7-15
- SetCurrentTime command 6-4
- SetHeartbeatRate command 6-6
- SetIgnoreAdvisories 3-9
- SetIgnoreAdvisories command 6-6
- SetLoopAtEnd command 6-4
- SetNotifyLevel command 6-4
- SetPassiveSubscription command 6-6
- SetPlaybackSpeed command 6-4
- SetPlayOnLoad command 6-5
- SetRecordOnNew command 6-5
- SetRemoteControlSettings command 6-6
- SetStartStopTimes command 6-5
- SetStopCondition command 6-5
- SetStopDuration command 6-5
- SetStopSize command 6-5
- SetTimeoutRate command 6-6
- setting
  - DIS heartbeat 3-5
  - DIS timeout 3-5
  - end-of-file action 5-7
  - notification level 3-6
  - playback segment to annotation 5-20
  - playback speed 5-8
  - preferences 3-6
- shared pointer 7-14
- showConsole** parameter 6-2, B-5
- shutting down
  - license server 2-12
- signal 7-16, 7-17
- signal slot 7-16
- silent installation 2-4
- simulation
  - recording 5-5
  - standard
    - supported 1-5
- simulation manager 7-6
- SISO 1516 API 1-5
- site ID
  - specifying 5-22, B-6
- siteId** parameter B-6
- skip command
  - entity maintenance 4-5
- Skip Time 4-5
- slider
  - start and end time 5-7
  - zoom 5-7
- slot 7-17
  - signal 7-16
- slow-speed playback 4-3
- smart pointer 7-14
- specifying
  - application number 5-22
  - FED file 3-7, 3-8, 5-21
  - federation execution name 3-7
  - language B-5
  - multicast time-to-live 5-22
  - receive buffer size 5-22
  - RPR FOM version 3-8
  - send buffer size 5-22
  - site ID 5-22
  - siteID B-6
  - UDP port 3-10
- speed
  - setting for playback 5-8
- speed control 5-4
- stage
  - input, output, state 7-6
- start
  - jumping to range annotation 5-18
  - moving to 5-10
- starting
  - recording 5-5
- starting Logger 5-3
- startup
  - playback mode 4-3

- record mode 4-2
- UNIX 5-3
- Windows 5-3
- state stage 7-6
- stateObController.h* header file 7-5
- stateObModel.h* header file 7-4
- stateObserverPlugin B-5
- status
  - field 5-5
  - license server 2-12
- stopping
  - license server 2-12
  - playback 5-7
  - recording 5-5
- StopPlayback command 6-5
- StopRecording command 6-5
- subscription
  - passive 3-9
- support
  - technical xiii
- syntax
  - lgrConfig.xml* B-2

## T

- tape
  - annotating 5-15
  - loading 5-6
  - playing 5-6
  - starting 5-5
- TCP 5-22
  - port B-6
- technical support xiii
- template 7-13
- template specialization 7-13
- text interface 6-2
- thread safety
  - mutex
    - use of 7-16
- tick() function 7-4, 7-7
- time
  - jumping to 5-10
  - navigating during playback 5-10
  - navigation 5-10
  - setting 5-10
  - sliders 5-7
- time command
  - entity maintenance 4-5
- time jump, updating entities after 4-5
- time scale

- described 4-3
- timeline
  - cursor 5-10
  - zooming 5-9
  - zooming out 5-9
  - zooming to annotation 5-19
- timeout
  - defined 4-4
  - setting 3-5
- timepoint
  - setting 5-10
- timescale
  - during recording 5-5
  - effects 4-3
- toolbar 5-4
- translating
  - GUI text 2-17
- translation file
  - untranslated.ts 2-19
- troubleshooting
  - FLEXlm 2-14

## U

- UDP port 3-10, 5-22, B-6
  - specifying 3-10, 5-22
- UNIX
  - compiling and linking API 7-18
  - starting
    - Logger 5-3
- UnloadPlugin() function 7-11
- Unpause command 6-5
- untranslated.ts* file 2-19
- update() function 7-5, 7-8
- upgrades xiii
- useAsyncIO* parameter 3-11, B-6

## V

- variable
  - adding to *lgrConfig.xml* B-4
  - environment 2-10
  - syntax in *lgrConfig.xml* B-3
- VCR controls 5-4
- velocity
  - scaling 4-5
- View Annotations window 5-15
- view control PDUs 1-4
- viewing
  - connection properties 3-3

## ***Index***

---

- object statistics 5-21
- virtual constructor 7-14
- virtualConstructor.h* header file 7-15
- VR-Link.fed* file 1-5, 3-7

## **W**

- window
  - Logger 5-4
- Windows
  - compiling and linking API 7-18
  - starting Logger 5-3

## **X-Y-Z**

- XML
  - RTI configuration file 2-16
- zoom area 5-7
- zoom bar 5-9
- zoom slider 5-9
- zooming
  - timeline 5-9
  - timeline out 5-9
  - timeline to annotation 5-19

# MÄK Data Logger Quick Reference Card

## Command-line Options

Option	Description
<b>Record and Playback</b>	
<code>-c mode</code>	If <i>mode</i> = 1, show the Logger console.
<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>-n level</code>	Specifies the notification level for the log file. Range 0-4.
<b>HLA Only</b>	
<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.
<b>DIS Only</b>	
<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. Set <i>num</i> to:
<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 <code>-A</code> 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
New	Ctrl + N
Open	Ctrl + O
Close	Ctrl + W
Connection	Ctrl + Shift + C
Disconnect	Ctrl + D
Exit	Ctrl + Q
Play	Ctrl + P
Pause	Ctrl + P
Jump to Start	Ctrl + R
Jump to Time	Ctrl + T
Jump to End	Ctrl + E
Toggle Loop at End	Ctrl + L
Stop	Ctrl + S
Record	Ctrl + R
Preferences	Ctrl + Shift + P

## Text Interface and Configuration Commands

Command	Description
<b>System Commands</b>	
DeleteFile <i>filename</i>	Deletes the specified file.
Exit	Shut down the Logger.
Help	Display a list of available commands.
LoadPlugin <i>pluginName</i>	Load a plug-in of the given name.
SetNotifyLevel <i>vlLevel lgrLevel</i>	Set the notification level of VR-Link and the Logger.
<b>Playback Commands</b>	
ClosePlaybackFile	Close the current playback file.
OpenFile <i>filename</i>	Open the provided file for playback.
Pause	Pause playback. This function will have no effect if the Logger does not have a file open or is already paused.
Play	Start playback. If the Logger is already playing this function has no effect.
PlayPause	Toggle pause/unpaused state.
Restart	Restart playback. Equivalent to Stop, Jump to Start and Play.
SetCurrentTime <i>time</i>	Jumps playback to the specified time.
SetLoopAtEnd <i>enable</i>	Enable or disable looping the start and the end of playback.
SetPlaybackSpeed <i>speed</i>	Sets the playback speed of the Logger.
SetPlayOnLoad <i>play</i>	Enable/display starting playback immediately after opening a file.
SetStartStopTimes <i>startTime stopTime</i>	Set the start and stop time of the Logger. The both times are bound to the first and last times in the tape. Also the <i>stopTime</i> must be after the <i>startTime</i> .
StopPlayback	Stops playback. This command has no effect if the Logger is not in the playback state.
Unpause	Unpause the Logger. This command has no effect if the Logger was not paused.
<b>Recording Commands</b>	
CloseRecordFile	Closes the file currently recording.
NewFile <i>filename overwrite</i>	Opens a new file for recording.
Record	Starts recording to a new file. This command has no effect if a new file is not opened.
SetCheckpointParameters <i>policy {size   count   time}</i>	Sets the checkpoint policy parameters on when checkpoints are generated. The Policy is either (count = 0, size = 1, time = 2). Size is in size of packets in bytes. Count is in number of packets. Time is in seconds between checkpoints.
SetRecordOnNew <i>record</i>	Enable or disable starting recording immediately after opening a new file.
SetStopCondition <i>condition</i>	Set the stop condition for recording. Conditions are: 0: no condition, 1: duration, 2: file size.
SetStopDuration <i>duration</i>	Stop recording at the specified time.
SetStopSize <i>filesize</i>	Stop recording when the file reaches <i>filesize</i> size.
StopRecording	Stops recording data to a file. The file remains open and recording may be restarted.
<b>Simulation Commands</b>	
AddCustomPDU <i>pduName pdfuKind entityIdOffset entityTypeOffset</i>	Specifies a custom PDU to add to the Filter by PDU Kind dialog box. (DIS only)
Connect	Connect to the network simulation using the previous parameters.
Disconnect	Disconnect from the network simulation.
SetCommonEntityTypes <i>entityType</i>	Specifies one or more DIS entity type enumerations to list in the Filter by Entity Type dialog box.
SetConnectionParameters <i>federateName federationName fedFile fomVersion fomMapper fomMapperData</i>	Specifies connection parameters for HLA.
SetConnectionParameters <i>port exerciseId siteId appNumber useAsynchIO destAddr sendBufferSize recvBufferSize multicastTtl mcastAddresses</i>	Specifies connection parameters for DIS.
SetHeartbeatRate <i>rate</i>	Specifies the DIS heartbeat, in seconds.
SetIgnoreAdvisories <i>mode</i>	Specifies the Logger's response to HLA advisory messages.
SetPassiveSubscription <i>mode</i>	Specifies whether the Logger subscribes to all objects and interactions or just records the updates that other federates subscribe to.
SetRemoteControlSettings <i>process record allowDelete</i>	Configure remote control settings, whether remote control messages are processed, recorded to the file, and if remote file deletion is allowed.
SetTimeoutRate <i>rate</i>	Specifies the DIS timeout value, in seconds.





Link - Simulate - Visualize

LOG-4.0-1-061004