

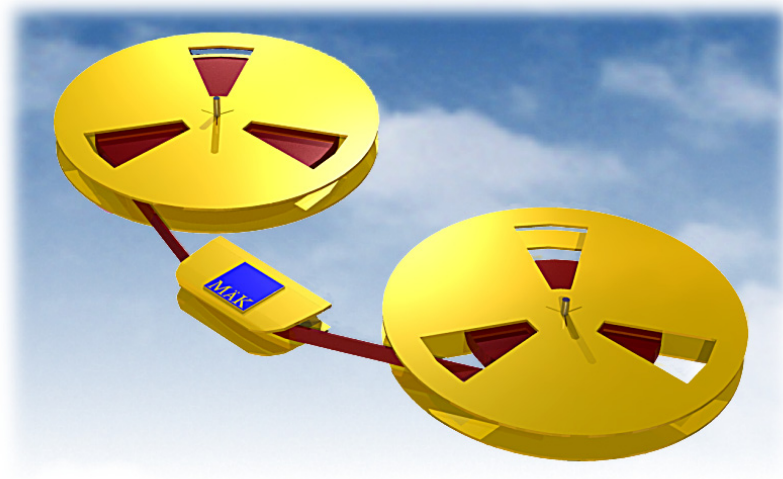


MÄK Data Logger



User's Guide





MÄK Data Logger



User's Guide

Copyright © 1997 - 2004 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.

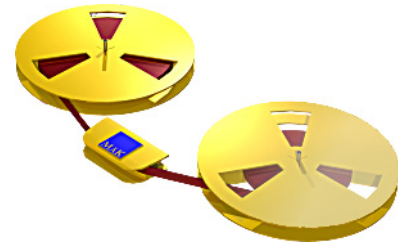
MÄK Technologies®, RTIspy®, VR-Forces®, and VR-Link® are registered trade-
marks of MÄK Technologies, Inc. All other trademarks are owned by their respective
companies.

MÄK Technologies
10 Fawcett St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revision LOG-3.10-1-040917



Contents

Preface

How the Manual is Organized	ix
Logger Documentation.....	x
Online Help	x
Other MÄK Products	xi
How to Contact Us	xii
Document Conventions	xiii
Hardware and Operating System Naming Conventions.....	xiii
Mouse Button Naming Conventions	xiv

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. Export to Database Format	1-3
1.1.5. Differences Between UNIX and Windows Versions	1-4
1.1.6. Sample Logger Files	1-4
1.1.7. Sample Programs	1-5
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 and Configuring the Logger

2.1. Installing the Logger	2-2
2.1.1. Installing the Logger on Windows	2-2
2.1.2. Installing the Logger On A UNIX System	2-3
2.2. The Logger Directory Structure	2-5

2.3. Setting Up and Using the FLEXlm License Manager	2-6
2.3.1. The FLEXlm Client-Server Architecture	2-6
2.3.2. Installing and Configuring the License Manager	2-7
2.3.3. Installing the License Manager Software	2-7
2.3.4. Requesting A License File	2-8
2.3.5. Putting the License File in the <i>flexlm</i> Directory	2-8
2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable	2-9
2.3.7. Adding Additional License Files	2-10
2.3.8. Running the License Server	2-11
2.3.9. Running FLEXlm as A Service on Windows	2-11
2.3.10. Troubleshooting the FLEXlm License Manager	2-12
2.4. Installing An RTI	2-14
2.4.1. Installing the MÄK RTI	2-14
2.4.2. Installing the DMSO RTI	2-15
2.5. Localizing the Logger Interface	2-16
2.5.1. Translating Other Interface Files	2-18
2.5.2. Applying the Language Files	2-18
2.6. Configuring the Logger	2-19
2.6.1. Protocol Independent Configuration Options	2-19
2.6.2. Configuring the Logger for HLA	2-21
2.6.3. Configuring the Logger for DIS	2-27

Chapter 3 Logger Concepts

3.1. What the Logger Does During Recording and Playback	3-2
3.1.1. What The Logger Does During Recording	3-2
3.1.2. What The Logger Does During Playback	3-3
3.1.3. Recording and Playback Using HLA Time Management	3-3
3.2. Timescale and Direction During Playback	3-5
3.2.1. Using Time Scaling with Time Management	3-5
3.3. Compensating for Timescale Effects During Playback	3-5
3.3.1. Generating Interim PDUs During Slow-Speed DIS Playback ..	3-6
3.3.2. Updating Entity Information Quickly After A Time Jump	3-7
3.3.3. Velocity and Acceleration Scaling	3-7
3.4. Specifying the End-of-File Action	3-8
3.5. Exporting Logger Data to Databases	3-8
3.6. HLA Support	3-8
3.6.1. FOM Flexibility	3-8
3.6.2. Backward Compatibility	3-9

Chapter 4 Using The Logger

4.1. The Logger Windows	4-3
4.1.1. Starting the Logger	4-4
4.2. Recording Exercises	4-5

4.3. Playing Logger Recordings	4-6
4.3.1. Loading A Logger File for Playback	4-6
4.3.2. Playing A Logger File	4-6
4.3.3. Pausing Playback of A Logger File	4-6
4.3.4. Stopping Playback of A Logger File	4-6
4.4. Navigating to A Specific Time in A Recording	4-7
4.4.1. Jumping to the Beginning or End of A Recording	4-7
4.4.2. Moving to A Specific Time in A Recording	4-8
4.4.3. Skipping Forward or Backward in A Recording	4-8
4.5. Filtering Logger Input and Output	4-9
4.5.1. Filtering Options for DIS	4-9
4.5.2. Filtering Options for HLA	4-13
4.5.3. Saving and Loading Filtering Settings	4-14
4.6. Annotating A Logger Recording	4-16
4.6.1. Adding An Annotation	4-16
4.6.2. Saving An Annotation File	4-17
4.6.3. Opening An Annotation File	4-17
4.6.4. Importing An Annotation File	4-17
4.6.5. Moving to An Annotation Point	4-18
4.6.6. Editing An Annotation File	4-18
4.6.7. Deleting An Annotation	4-18
4.6.8. Clearing the Annotation List	4-19
4.7. Setting the Timescale and Direction	4-19
4.7.1. Using the Timescale Slider	4-19
4.7.2. Entering A Timescale Value Numerically	4-20
4.8. Setting Configuration Options in the GUI	4-21
4.9. Summary of Startup Options for the Logger	4-23

Chapter 5 Using The Text Interface

5.1. Starting the Text Interface	5-2
5.2. Recording Exercises	5-2
5.2.1. Beginning A Recording Session in Record Mode	5-2
5.2.2. Logger Run-time Commands for Recording	5-3
5.2.3. Displaying the Status of A Recording	5-3
5.3. Playing Back Recordings	5-3
5.3.1. Basic Logger Playback Commands	5-4
5.3.2. Displaying Playback Status	5-4
5.3.3. Beginning A Playback Session in Play Mode	5-4
5.4. Navigating Time	5-5
5.4.1. Controlling End-of-File Behavior	5-5
5.5. Setting Playback Timescale and Direction	5-6
5.5.1. Setting the Timescale at Startup	5-6
5.6. Controlling the DIS Heartbeat and Timeout	5-6
5.6.1. Setting the Heartbeat Rate	5-7
5.6.2. Setting the Timeout	5-7

5.7. Using A Script to Control Recording and Playback	5-7
5.7.1. Controlling Processing with the Wait Command	5-8
5.7.2. Stopping A Script	5-8
5.8. Summary of Logger Run-Time Commands	5-9

Chapter 6 Exporting Logger Data to A Database

6.1. Connecting to SQL Database Servers	6-2
6.2. Using the sqlSender Plug-in	6-2
6.2.1. Loading the sqlSender Plug-in	6-2
6.2.2. Configuring and Enabling the sqlSender Plug-in in the Logger GUI	6-3
6.2.3. Configuring and Enabling the sqlSender Plug-in in <i>logger.mtl</i> ..	6-5
6.2.4. Disabling the sqlSender Plug-in	6-6
6.2.5. Troubleshooting Data Export	6-7
6.3. Using the sqlSender Application	6-7
6.4. Database Schema Used by the Logger SQL API	6-9
6.4.1. The entities Table	6-9
6.4.2. The aggregates Table	6-11
6.4.3. The aggregate_subordinates Table	6-13
6.4.4. The detonations Table	6-13
6.4.5. The fires Table	6-15
6.5. Importing Logger Data into A Spreadsheet	6-17

Chapter 7 Remote Control

7.1. Remote Control Messages	7-2
7.2. Using Logger Commands for Remote Control	7-3
7.2.1. Capturing User Actions as Logger Control Messages	7-3
7.2.2. Deleting Logger Files	7-4
7.2.3. Getting the Logger's State	7-4
7.3. Example of Remote Control	7-5

Chapter 8 Using The Logger API

8.1. Introduction	8-2
8.2. The <i>DtLogger</i> Class	8-4
8.2.1. Including the Logger in An Application	8-4
8.2.2. Controlling the Logger Time Clock	8-6
8.2.3. Controlling Logger Files	8-8
8.2.4. Controlling the Logger's Configuration and Status	8-9
8.2.5. Controlling Logger Annotations	8-10
8.2.6. Setting Protocol-Specific Parameters	8-11
8.2.7. Managing Pipeline Objects	8-12
8.2.8. Using Large Files	8-12
8.3. The Logger Pipeline Architecture	8-13

8.4. Logger Pipeline Stages	8-14
8.4.1. Pipeline Stage Member Functions	8-14
8.4.2. Default Pipeline Stages	8-18
8.5. Creating Plug-ins	8-18
8.5.1. Initializing A Plug-in	8-19
8.5.2. Modifying the Pipeline Using A Plug-in	8-20
8.5.3. Overriding <i>DtLogger</i> from A Plug-in	8-22
8.5.4. Loading Plug-Ins	8-23
8.6. Managing Pipeline Stages	8-24
8.6.1. Adding Pipeline Stages	8-25
8.6.2. Removing Pipeline Stages	8-26
8.7. Compiling and Linking with the Logger API	8-26

Chapter 9 Using the Logger File API

9.1. Using the Logger File API	9-2
9.1.1. Reading A Logger File	9-2
9.1.2. Creating An <i>LtLoggerPlaybackFile</i>	9-3
9.1.3. Reading Data	9-4
9.1.4. Writing A Logger File	9-5
9.1.5. Creating An <i>LtLoggerRecordFile</i>	9-5
9.1.6. Writing Data	9-6
9.1.7. Interpreting <i>LtPacketInfo</i>	9-6
9.1.8. Using Large Files	9-9
9.2. Editing Logger Files	9-9
9.3. Compiling and Linking with the Logger Library	9-10

Chapter 10 Using Logger Tools

10.1. Viewing Logger Files with <i>lgrDump</i>	10-2
10.1.1. <i>lgrDump</i> for HLA	10-3
10.1.2. How <i>lgrDump</i> Displays Unexpected Message Types	10-5
10.1.3. Displaying Raw Data	10-5
10.2. Concatenating Files with <i>lgrCat</i>	10-6
10.2.1. Inserting Time Between Concatenated Files	10-7
10.2.2. Processing Input from Other Programs	10-7
10.3. Merging Files with <i>lgrMerge</i>	10-8
10.4. Using <i>lgrOffset</i> to Shift Messages in Time	10-9
10.5. Using <i>lgrTrim</i> to Extract Part of A File	10-10

Appendix A Troubleshooting

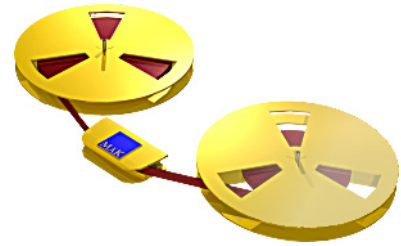
A.1. Troubleshooting Logger Network Communications (DIS Only)	A-2
A.1.1. Using the Correct Broadcast Address and UDP Port	A-2

Appendix B Logger MTL Parameters

B.1. MÄK Technologies Lisp (MTL) Files	B-2
B.1.1. Using Environment Variables in An MTL File	B-2
B.1.2. Specifying Lists of Lists	B-3
B.1.3. Using Conditional Statements	B-3
B.2. Logger MTL Parameters	B-4

MÄK Products Glossary

Index



Preface

The MÄK Data Logger (Logger) records network activity of HLA or DIS exercises and plays it back for review. The Logger has a graphical user interface, a 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 *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 and Configuring the Logger*, explains how to install the Logger and related software such as the FLEXlm license manager and the RTI. It also explains how to configure the Logger for use with HLA or DIS.

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

Chapter 4, *Using The Logger*, shows how to record and play back HLA/DIS network activity using the Logger's graphical user interface.

Chapter 5, *Using The Text Interface*, explains how to run the Logger from the command-line.

Chapter 6, *Exporting Logger Data to A Database* explains how to use the sqlSender plug-in and *sqlSender* application to export Logger data to a SQL database.

Chapter 7, *Remote Control*, explains how to control the Logger using view control messages sent from other applications.

Chapter 8, *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 9, *Using the Logger File API* describe the Logger File API, which allows you to read and write Logger recordings from another application.

Chapter 10, *Using Logger Tools*, describes the utility programs that combine, modify, and inspect Logger recordings.

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

Appendix B, *Logger MTL Parameters* lists and describes MTL parameters that you can use with the Logger.

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

There is a *Quick Reference Card* at the end of the manual that you can remove and keep near your computer for convenience. It lists startup 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.

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

Online Help

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

Other MÄK Products

The Logger is a member of the MÄK Technologies® line of software products designed to streamline the process of developing and using networked simulated environments.

In addition to Logger, the MÄK Technologies product line includes:

- **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.

VR-Link is the basis of the other MÄK products.

- **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, RTIsby®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface for the MÄK RTI rtiexec.
- **MÄK 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.** 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.
- **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.

- ♦ MÄK Gateway. The Gateway is a translator application that allows DIS exercises to operate as members of HLA federation executions that use the RPR FOM, thereby preserving the value of simulation code written to support DIS. The MÄK Gateway is a standalone application and does not require you to write additional code for its use.

How to Contact Us

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

Telephone

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

E-mail

Sales and upgrade information:	info@mak.com
Technical support:	support@mak.com

Internet

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

Post

Send postal correspondence to:	MÄK Technologies 10 Fawcett 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.
<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.

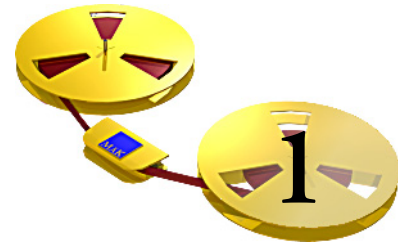
Directory names are preceded with dot and slash characters that show their position with respect to the Logger home directory. For example, the directory *logger/doc* appears in the text as *./doc*.

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
Export to Database Format.....	1-3
Differences Between UNIX and Windows Versions.....	1-4
Sample Logger Files	1-4
Sample Programs	1-5
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 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.

The Logger software package includes the following:

- ♦ The Logger, a data-recorder/player that you can control through a VCR-style graphical user interface, a text interface, a remote control interface, or programmatically.
- ♦ A set of Logger Tools that let you combine, modify and inspect recorded Logger files. You can merge the activity from several exercises into a single Logger file, extract a portion of a recording to a new file, and perform other file operations.
- ♦ Libraries and header files that support APIs that allow you to write plug-ins for the Logger and allow you to embed Logger functionality in other applications.
- ♦ A set of sample Logger recordings (also called Logger tapes).
- ♦ Example projects for building applications and plug-ins that use the Logger API.
- ♦ A User's Guide, online help, and class documentation.

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 provides the following interfaces:

- ♦ A graphical user interface (GUI) that uses VCR-style buttons to control the recording and playback of exercises
- ♦ 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 includes translation files that allow you to localize the Logger's GUI. See [“Localizing the Logger Interface,”](#) on page 2-16 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 objects and interactions you want to subscribe to.

1.1.3. Annotations

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

1.1.4. Export to Database Format

You can export Logger recordings to SQL databases for storage and retrieval.

1.1.5. Differences Between UNIX and Windows Versions

There are a few minor differences between the UNIX and Windows versions of the Logger.

Under UNIX, Logger commands are case-sensitive, while under Windows, only the startup option letters are case-sensitive.

For consistency across platforms, the hyphen is always used to precede command-line arguments, rather than the slash character typically used with Microsoft Windows command-line arguments.

1.1.6. Sample Logger Files

The Logger includes several sample recordings:

- ♦ *montyDemo.lgr* and *montyDemoRPRxx.lgr* — DIS 2.0.4 and HLA (RPR FOM 1.0) versions of an exercise in which two A-10s attempt to destroy seven ground targets near an airfield. It uses a portion of the Monterey database that is distributed with the MÄK Stealth, Plan View Display, and VR-Forces.
- ♦ *aggDemDIS.lgr* — demonstrates aggregate entities.
- ♦ *berkeleyDemo.lgr* and *berkeleyDemoRPR10.lgr* — 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, Plan View Display, and VR-Forces.



The *montyDemo* and *berkeleyDemo* 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, see your Stealth documentation.

Sample Annotation Files

The *montyDemo* (HLA only) and *berkeleyDemo* (DIS only) recordings have annotation files. (You must load the *montyDemoRPR.ant* file manually, because its title does not match that of the HLA versions of the *montyDemo* Logger recordings.)

1.1.7. Sample Programs

The Logger includes the following example programs:

- ♦ A plug-in that filters out air entities
- ♦ A simple example of using the Logger API
- ♦ An example of using the Logger File API
- ♦ An example of a remote control commander application
- ♦ An example of a remote control monitoring application.

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 SISO Dynamic Link Compatibility HLA API Product Development Group (HLA API PDG, <http://www.sisostds.org/index.cfm>) draft 1516 API. 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 a FED file called *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. 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 and Configuring the Logger

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

Installing the Logger	2-2
Installing the Logger on Windows	2-2
Installing the Logger On A UNIX System	2-3
The Logger Directory Structure	2-5
Setting Up and Using the FLEXlm License Manager	2-6
The FLEXlm Client-Server Architecture	2-6
Installing and Configuring the License Manager.....	2-7
Installing the License Manager Software	2-7
Requesting A License File	2-8
Putting the License File in the flexlm Directory.....	2-8
Setting the MAKLMGRD_LICENSE_FILE Variable.....	2-9
Adding Additional License Files.....	2-10
Running the License Server	2-11
Running FLEXlm as A Service on Windows.....	2-11
Troubleshooting the FLEXlm License Manager	2-12
Installing An RTI.....	2-14
Installing the MÄK RTI	2-14
Installing the DMSO RTI	2-15
Localizing the Logger Interface	2-16
Translating Other Interface Files.....	2-18
Applying the Language Files.....	2-18
Configuring the Logger.....	2-19
Protocol Independent Configuration Options	2-19
Configuring the Logger for HLA.....	2-21
Configuring the Logger for DIS	2-27

2.1. Installing the 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 (see [“Setting Up and Using the FLEXlm License Manager,”](#) on page 2-6) and if you want to use the Logger with HLA, an RTI (see [“Installing An RTI,”](#) on page 2-14).

2.1.1. Installing the 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 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 Logger from A Downloaded File

The Logger is distributed via ftp as 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 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 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 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
```

2.2. The Logger Directory Structure

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

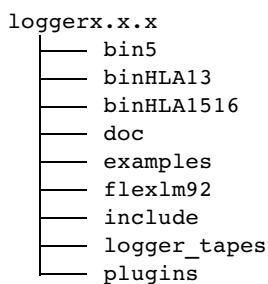


Figure 2-2. Logger directory structure

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

Table 2-1: Logger directory structure

Directory	Contents
<i>bin5</i>	Executable for DIS versions 2.0.4, 2.1.4, IEEE 1278, and IEEE 1278.1a.
<i>binHLA13</i> and <i>binHLA1516</i>	Executables for use with the HLA RTI 1.3 and 1516.
<i>doc</i>	Documentation.
<i>examples</i>	Examples of the Logger API and the Logger File API.
<i>flexlm92</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>plugins</i>	This directory and its subdirectories contains plug-ins and example code.

2.3. Setting Up and Using the FLEXlm License Manager

The Logger, like other MÄK Technologies products, uses the FLEXlm license manager. Before you can use the Logger, 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.

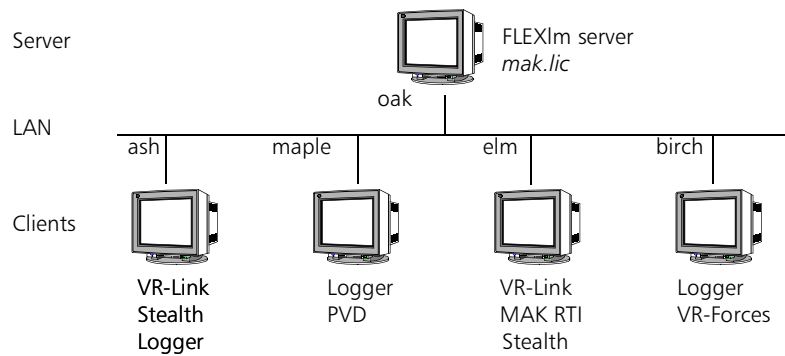
The license server executes vendor daemons (for MÄK Technologies, *maklmgrd*) and uses one or more license files, provided by individual vendors. The license file identifies the products for which you have licenses. On each client machine, the `MAKLMGRD_LICENSE_FILE` environment variable (which you must 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 2-10.

2.3.3. Installing the License Manager Software

As part of the installation of all MÄK products, the FLEXlm software is installed in the *flexlm92* directory under the product's root directory.

1. Install your MÄK products on the machines on which you want to run them.
2. Copy the files in *./flexlm92* 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/licenses.htm> 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 2-10.

2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable

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



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

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

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

Windows 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 2-11.

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 2-8.

To add a new license file to your license server:

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

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

2.3.8. Running the License Server

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

- To start the license server, 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, 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 the `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 path of the directory where the `lmgrd.exe` file is located (for example, `C:/flexlm`), or use the Browse button to select the directory.
7. In the Path To The License File box, enter the path of the directory where the license file is located (for example, `C:/flexlm`), or use the Browse button to select the directory.
8. In the Path To The Debug Log File box, enter the path of the directory where you want FLEXlm to create the debug file, or use the Browse button to select a directory.
9. Check the User Services check box. The Start Server At Power Up check box becomes enabled.
10. Check 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 putting 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.

If you are trying to run your application with the DMSO 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 DMSO RTI. To fix this problem do the following:

- ♦ Make sure that the DMSO 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.

All MÄK products have been tested against the MÄK RTI and the DMSO RTI. For the most recent information about the RTI versions supported by the Logger, see the Product Versions page on the MÄK Technologies web site at:
http://www.mak.com/product_version.htm.



Some versions of MÄK products are available for operating systems that the DMSO RTI does not support, for example, later releases of Linux. MÄK does not test these product versions with the DMSO RTI and does claim compatibility with the DMSO RTI on these operating systems.

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. This 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 most 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.

2.4.2. Installing the DMSO RTI

DMSO does not distribute the DMSO RTI any more, but you may be able to obtain it from other sources.

Consult the DMSO RTI Release Notes to determine version information and whether or not any patches are required. Install the RTI according to the instructions provided.

Configuring Your System to Use the DMSO RTI

Follow the configuration instructions provided by your version of the DMSO RTI.

Running Applications Using the DMSO RTI

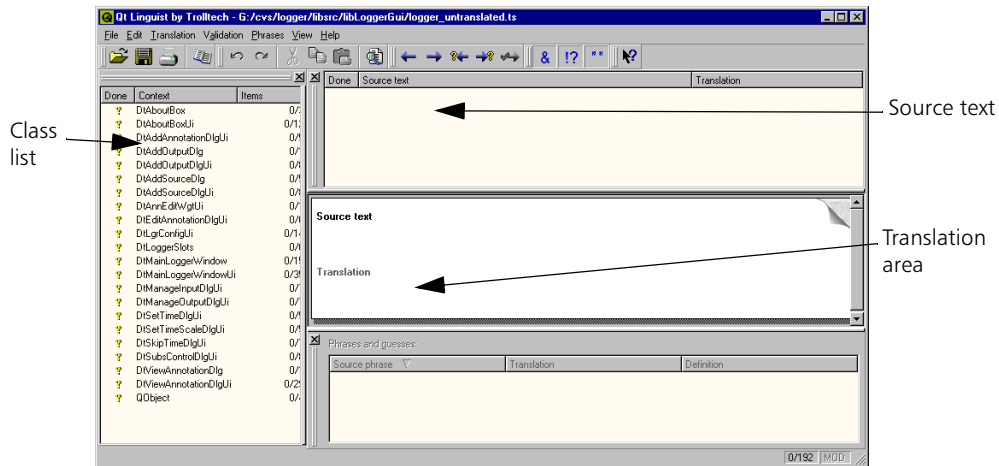
To use the DMSO RTI, you need to run the RTI executive (`rtiexec`). This executable is in the *bin* directory or a subdirectory. Run the `rtiexec` only on one machine, regardless of how many HLA applications you are running.

Once the `rtiexec` is running, you can run HLA applications.

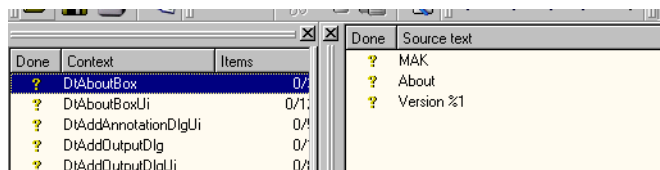
2.5. Localizing the Logger Interface

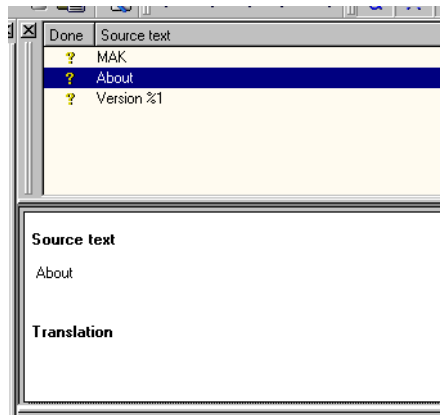
You can localize the Logger interface so that it uses languages other than English. To localize the Logger, you edit a resource file using the following procedure:

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.

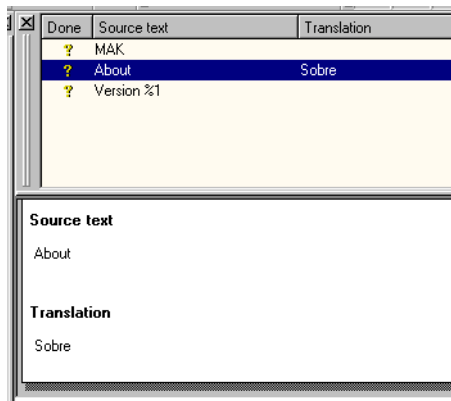


3. To translate text, you select each class in turn and translate the text associated with that class. You do not have to understand anything about what the classes mean, you just have to translate the text. For example, the first class in the list, *DtAboutBox*, contains the text for the About Box.
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, check 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 locale to the proper language.
- ♦ Set the **language** parameter in *logger.mtl* to the correct language.
- ♦ Start the Logger with the **-G** startup option.

2.6. Configuring the Logger

You can configure Logger parameters in the *logger.mtl* file or with startup options. Startup options override settings in the *logger.mtl* file. You can also set the heartbeat and timeout at runtime in the Configuration dialog box. For a complete list of MTL parameters, see “[Logger MTL Parameters](#),” on page B-4. For a complete list of startup options, see “[Summary of Startup Options for the Logger](#),” on page 4-23 or the Quick Reference Card.

2.6.1. Protocol Independent Configuration Options

This section describes or references configuration options that apply to both HLA and DIS. The following configuration options are described elsewhere in this manual:

- ♦ Allowing file deletion by Logger Control messages is described in “[Deleting Logger Files](#),” on page 7-4
- ♦ Specifying which Logger Control messages the Logger responds to is described in “[Using Logger Commands for Remote Control](#),” on page 7-3
- ♦ Recording Logger Control messages is described in “[Using Logger Commands for Remote Control](#),” on page 7-3
- ♦ Sending Logger Control messages when the state of the Logger changes is described in “[Capturing User Actions as Logger Control Messages](#),” on page 7-3
- ♦ Setting the initial time scale is described in “[Setting Playback Timescale and Direction](#),” on page 5-6
- ♦ Specifying the end-of-file action is described in “[Setting Configuration Options in the GUI](#),” on page 4-21 and “[Controlling End-of-File Behavior](#),” on page 5-5
- ♦ Specifying an alternative plug-in directory is described in “[Loading Plug-Ins](#),” on page 8-23.

Specifying the User Interface

The Logger has a graphical user interface (GUI) and a text-based interface. You can also hide the interface.

- To specify the type of interface to use, specify the `interfaceType` parameter in the *logger.mtl* file, or start the Logger with the `-g` startup option (see “[Starting the Text Interface](#),” on page 5-2).

Specifying A Configuration File

The default Logger configuration file is *logger.mtl*.

- To specify an alternative configuration file, start the Logger with the `-c` startup option, for example:

```
Logger -c myconfigfile.mtl
```

Using Large Files

By default, the Logger can accept recordings that are up to 2 GB. In 64 bit mode, the Logger can accept larger files.

- To use large files, start the Logger with the **-Q 1** startup option, or set the **useLargeFiles** parameter to 1 in *logger.mtl*.

When using the Logger API, you can direct the Logger to use 64-bit file I/O by calling `setUseLargeFiles()` on the *DtLogger* object. If you are using the Logger File API, pass a value of 1 for the **useFD64** parameter when invoking either the constructors or the `init()` functions for *LtLoggerRecordFile* or *LtLoggerPlaybackFile*.

Setting the Notification Level

The notification level determines the verbosity of informational messages that the Logger generates. You can specify a notification level for all messages and one for Logger-specific messages. The range is 0-4, with 0 being least verbose.

- To specify the notification level, start the Logger with the **-n** startup option or set the **DtNotifyLevel** and **DtLoggerNotifyLevel** parameters in *logger.mtl*.

The syntax is:

```
-n generalNotify:loggerNotify
```

For example:

```
-n 2:3
```

Enabling or Disabling Scaling of Articulated Parts

When the Logger is in paused mode, it stops sending update messages for articulated parts so that they do not continue to move while the host entity is paused. Although it is not recommended, you can disable articulated parts scaling.

To disable articulated parts scaling, start the Logger with the **-Y 1** startup option or set the **noArtPartScaling** parameter to 1 in *logger.mtl*.

2.6.2. Configuring the Logger for HLA

This section describes configuration parameters and startup options that are for HLA exercises.

Specifying the Federation Execution Name

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

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

```
Logger -p my_exercise.lgr -x FedExName
```

where *FedExName* is the name of the federation execution. You can also specify the federation execution name with the **federationName** parameter in *logger.mtl*.



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 2-22.

Selecting A FOM Mapper

The Logger has built-in support for the RPR FOM. (See the *Release Notes* for your version of the Logger for a list of supported versions of the RPR FOM.) It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper.

You can specify which FOM Mapping code to use by specifying the **rprFomVersion** or **fomMapperLib** and **fomMapperInitData** parameters in *logger.mtl*, or with the **-f** startup option, in one of two ways:

```
Logger -f v1RPRxx
Logger -f library:init_data
```

The keywords for specifying different versions of the RPR FOM with the **-f** startup option are:

- ♦ RPR FOM 0.5: **v1RPR05**
- ♦ RPR FOM 0.7: **v1RPR07**
- ♦ RPR FOM 0.8: **v1RPR08**
- ♦ RPR FOM 1.0: **v1RPR10**
- ♦ RPR FOM 2.0 draft 6: **v1RPR20006**
- ♦ RPR FOM 2.0 draft 14: **v1RPR20014**
- ♦ RPR FOM 2.0 draft 17: **v1RPR20017**.

If the argument to the **-f** option is one of the **v1RPRxx** keywords, the Logger uses VR-Link's built-in RPR FOM mapper, configured for the RPR FOM version specified by the keyword, for example, **v1RPR08** specifies RPR FOM 0.8.

If the argument to the `-f` option is not one of the keywords, it is interpreted as the name of a shared library (DSO or DLL) containing code for an alternate FOM Mapper that you would like the Logger to use, optionally followed by any initialization data the library requires. The library name and initialization data must be separated by a colon.

Examples

```
Logger -f v1RPR08
```

This command instructs the Logger to use the built-in RPR FOM Mapper, configured for RPR FOM version 0.8.

```
Logger -f myFomMapLib:ABC
```

This command instructs the Logger to use the FOM Mapper contained in a shared library called *myFomMapLib.so* (or *myFomMapLib.dll*). The string “ABC” is passed to the FOM Mapper creation function in the shared library as an initialization parameter.

Specifying A FED File



The `-f` option only tells the application which FOM Mapping code to use. It does not ensure that you are using the right 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 or the `fedFileName` parameter in *logger.mtl*. 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. (See the *Release Notes* for your version of the Logger for a list of FED files supplied.)

The following example instructs the Logger to use the RPR FOM 0.8 FOM Mapping code (`-f v1RPR08`), and also instructs it to use the *VR-Link08.fed* file (`-F VR-Link08`).

```
Logger -f v1RPR08 -F VR-Link08
```

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` startup option or the `federateName` parameter in *logger.mtl*.

Specifying Use of Time Management

The Logger supports recording and playback in a time-managed federation executions. See “Recording and Playback Using HLA Time Management,” on page 3-3 for details.

- To enable and start use of Time Management for recording at startup, use the `-u` startup option or set the `timeConstrainedRecord` parameter to 1 in `logger.mtl`.
- To enable and start use of Time Management for playback at startup, use the `-U` startup option or set the `timeRegulatingPlayback` parameter to 1 in `logger.mtl`.
- To enable use of Time Management for playback or recording, but not start its use, use the `-w` startup option or set the `useTimeManagement` parameter to 1 in `logger.mtl`.

Notes:

- ♦ The `-u` and `-U` options override the `-w` option.
- ♦ To use Time Management in the Logger, you must configure your RTI for Time Management services.

Specifying the Lookahead Value

To specify the lookahead value, in seconds, for Time Management, use `-l` startup option or the `fedLookahead` parameter in `logger.mtl`.

Specifying A Global Object Designator

- To specify the global object designator used by the Logger, specify the `entityIDStr` parameter in the `logger.mtl` file, or use the `-a` startup option.



This designator is used extensively when running with the `-L 1` option and the remote control interface.

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, start it with the `-P 1` startup option.

Ignoring Advisory Messages

- To have the Logger ignore advisory messages, start it with the `-i 1` startup option or set the `ignoreAdvisories` parameter to 1 in `logger.mtl`.

Specifying Class and Interaction Subscriptions

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



On PCs, you can specify subscriptions through the GUI. See “[Specifying the Subscription List from the Logger GUI](#),” on page 2-26.

To create a subscription control file manually:

1. Start the Logger with the `—S` startup option and specify the name of a file to write the subscription list to, for example:

```
Logger -S sublist.mtl
```

The Logger creates the file based on the information in the FOM that you are using. The *sublist.mtl* file looks as follows for the VR-Link FED file:

```
;subscription lists
(setq subscribedClasses
  (list
    "ObjectRoot.EnvironmentProcess"
    "ObjectRoot.GriddedData"
    "ObjectRoot.BaseEntity"
    "ObjectRoot.BaseEntity.EnvironmentalEntity"
    "ObjectRoot.BaseEntity.BaseEntityOther"
    "ObjectRoot.BaseEntity.AggregateEntity"
    "ObjectRoot.BaseEntity.PhysicalEntity"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.AmphibiousVehicle"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.GroundVehicle"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.MultiDomainPlatform"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.Spacecraft"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.SubmersibleVessel"
    "ObjectRoot.BaseEntity.PhysicalEntity.Platform.SurfaceVessel"
    "ObjectRoot.BaseEntity.PhysicalEntity.Lifeform"
    "ObjectRoot.BaseEntity.PhysicalEntity.Lifeform.Human"
    "ObjectRoot.BaseEntity.PhysicalEntity.Lifeform.NonHuman"
    "ObjectRoot.BaseEntity.PhysicalEntity.CulturalFeature"
    "ObjectRoot.BaseEntity.PhysicalEntity.Expendables"
    "ObjectRoot.BaseEntity.PhysicalEntity.Munition"
    "ObjectRoot.BaseEntity.PhysicalEntity.Radio"
    "ObjectRoot.BaseEntity.PhysicalEntity.Sensor"
    "ObjectRoot.BaseEntity.PhysicalEntity.Supplies"
    "ObjectRoot.EmbeddedSystem"
    "ObjectRoot.EmbeddedSystem.Designator"
    "ObjectRoot.EmbeddedSystem.EmitterSystem"
    "ObjectRoot.EmbeddedSystem.RadioReceiver"
    "ObjectRoot.EmbeddedSystem.RadioTransmitter"
    "ObjectRoot.EmitterBeam"
    "ObjectRoot.EmitterBeam.RadarBeam"
    "ObjectRoot.EmitterBeam.JammerBeam"
  )
)
```

```
)  
  
(setq subscribedInteractions  
  (list  
    "InteractionRoot.Acknowledge"  
    "InteractionRoot.ActionRequest"  
    "InteractionRoot.ActionResponse"  
    "InteractionRoot.ActionRequestToObject"  
    "InteractionRoot.ActionResponseFromObject"  
    "InteractionRoot.AttributeChangeRequest"  
    "InteractionRoot.AttributeChangeResult"  
    "InteractionRoot.Collision"  
    "InteractionRoot.Comment"  
    "InteractionRoot.CreateObjectRequest"  
    "InteractionRoot.CreateObjectResult"  
    "InteractionRoot.CreateEntity"  
    "InteractionRoot.Data"  
    "InteractionRoot.DataQuery"  
    "InteractionRoot.EventReport"  
    "InteractionRoot.MunitionDetonation"  
    "InteractionRoot.RadioSignal"  
    "InteractionRoot.RadioSignal.ApplicationSpecificRadioSignal"  
    "InteractionRoot.RadioSignal.DatabaseIndexRadioSignal"  
    "InteractionRoot.RadioSignal.EncodedAudioRadioSignal"  
    "InteractionRoot.RadioSignal.RawBinaryRadioSignal"  
    "InteractionRoot.RemoveObjectRequest"  
    "InteractionRoot.RemoveObjectResult"  
    "InteractionRoot.RemoveEntity"  
    "InteractionRoot.RepairComplete"  
    "InteractionRoot.RepairResponse"  
    "InteractionRoot.ResupplyCancel"  
    "InteractionRoot.ResupplyOffer"  
    "InteractionRoot.ResupplyReceived"  
    "InteractionRoot.ServiceRequest"  
    "InteractionRoot.SetData"  
    "InteractionRoot.StartResume"  
    "InteractionRoot.StopFreeze"  
    "InteractionRoot.WeaponFire"  
    "InteractionRoot.ViewControl"  
  )  
)
```

2. Edit the file by adding or removing the classes and interactions that you do not want the Logger to subscribe to.

Specifying the Subscription List from the Logger GUI

You can control the subscription list from the graphical user interface. This option is available only in Record mode.

To create a subscription control list:

1. In the Logger window (in Record mode), choose Options → Subscription Control. The Subscription Control dialog box opens (Figure 2-4). The Subscription Control window lists all object classes and interactions in the FOM. The objects and interactions that the Logger subscribes to are highlighted.

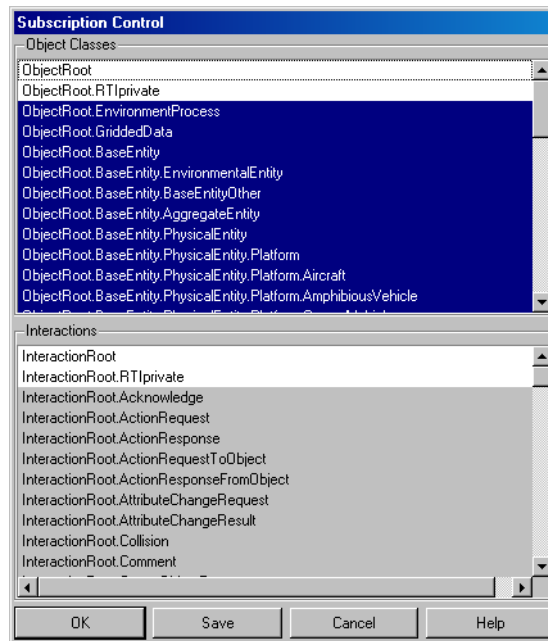


Figure 2-4. Subscription Control dialog box

2. Edit the subscription list by clicking on the object classes or interactions that you want to subscribe to or remove from the list.
3. Click OK.

Saving A Subscription List

To save a subscription list:

1. Edit the subscription list as described in the previous section.
2. Click Save. The Save As dialog box opens.
3. Enter a name for the subscription list and click Save.

Loading A Subscription List at Startup

To load a subscription list at startup, include the file in the *logger.mtl* configuration file by adding the following line.

```
(load "pathToSublistFile" "sublistfile")
```

where:

- ♦ *pathToSublistFile* is the relative or complete pathname of the directory in which the subscription list file is located.
- ♦ *sublistfile* is the name of the subscription list file.

Specifying the Content of Logger-Generated Message Tags

From time-to-time the Logger has to generate object update messages that are not part of a recording. For example, a federate may enter an exercise and ask for an update for objects that the Logger is responsible for. The **-q** option specifies how the Logger should fill the message tag in the object update message. The syntax is:

```
Logger -q mode
```

where *mode* is one of the following:

- ♦ 0 = Use the same tag as the last message for this object
- ♦ 1 = The tag is an `asciiTime` timestamp computed by adding the time in the tag from the last message for the object and the time interval since that tag, using tape-time.
- ♦ 2 = The tag is an `asciiTime` timestamp computed by adding the time in the tag from the last message for the object and the time interval since that tag, using simulation time.

You can also specify this setting with the `messageTagType` MTL parameter.

2.6.3. Configuring the Logger for DIS

This section describes configuration parameters and startup options for DIS exercises.

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 `destAddr` parameter in the *logger.mtl* file, or use the **-A** command-line option, as in this example:

```
Logger -p my_exercise.lgr -A 123.234.123.234
```

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

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** startup option. To subscribe to more than one multicast address, use the **-S** option several times in the same command:

```
Logger -r 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. Therefore, 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.

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 *logger.mtl* file, or start the Logger with the **-P** option, for example:

```
Logger -P 2727
```

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.

- To specify the size of the receive buffer, set the **recvBufferSize** parameter in the *logger.mtl* file, or use the **-B** startup option, as follows:

```
logger -B 64000:64000
```

where the values provided with **-B** represent *receiveBufferSize:sendBufferSize*.

- To specify the size of the send buffer, set the **sendBufferSize** parameter in the *logger.mtl* file, or use the **-B** startup option, as follows:

```
logger -B 64000:64000
```

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 *logger.mtl* file, or start the Logger with the `-I 1` startup option.

Setting the DIS Heartbeat

The heartbeat specifies how frequently, in seconds, the Logger sends out interim PDUs during slow-speed playback.

- To configure the heartbeat, specify the `heartbeat` parameter in the *logger.mtl* file.

Setting the Timeout

The timeout value specifies how long, in seconds, the Logger should continue to generate PDUs for an entity during slow-speed playback without timing it out.

- To configure the timeout, specify the `timeout` parameter in the *logger.mtl* file.

Setting the host:application:entity ID

- To specify the host:application:entity ID, specify the `entityIDStr` parameter in the *logger.mtl* file, or use the `-a` startup option, for example:

```
Logger -a 3000:2:34
```



The entity ID string (designator) is used extensively when running with the `-L 1` option and the remote control interface.

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 `-l` startup option or specify the `multicastTtl` parameter in *logger.mtl*.

Setting the Exercise ID

To set the DIS exercise ID, use the `-x` option or set the `exerciseld` parameter in *logger.mtl*, for example:

```
Logger -x 2
```

Disabling Data Scaling (DIS Only)

You can disable the generation of velocity and acceleration scaling data when playing at speeds above or below normal. See [“Generating Interim PDUs During Slow-Speed DIS Playback,”](#) on page 3-6 for a complete discussion.

- To disable data scaling, start the Logger with the `-d 1` startup option.



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 4, *Using The Logger* and Chapter 5, *Using The Text Interface*.

What the Logger Does During Recording and Playback	3-2
What The Logger Does During Recording.....	3-2
What The Logger Does During Playback	3-3
Recording and Playback Using HLA Time Management.....	3-3
Timescale and Direction During Playback	3-5
Using Time Scaling with Time Management.....	3-5
Compensating for Timescale Effects During Playback.....	3-5
Generating Interim PDUs During Slow-Speed DIS Playback.....	3-6
Updating Entity Information Quickly After A Time Jump	3-7
Velocity and Acceleration Scaling	3-7
Specifying the End-of-File Action	3-8
Exporting Logger Data to Databases	3-8
HLA Support.....	3-8
FOM Flexibility	3-8
Backward Compatibility	3-9

3.1. What the Logger Does During Recording and Playback

This section describes, in general, what happens when you record and play back exercises. [Figure 3-1](#) illustrates the Logger window (for DIS) in playback and record modes.

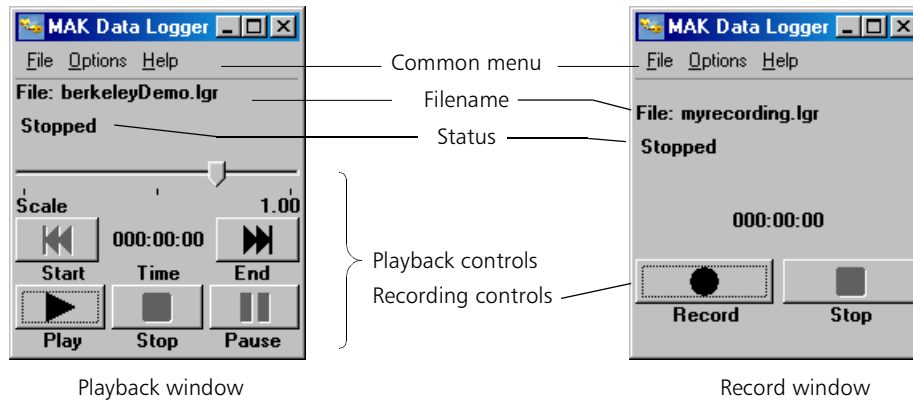


Figure 3-1. The Logger windows (DIS)

3.1.1. 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. The Logger can subscribe actively or passively.

3.1.2. 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 and direction to produce fast motion, slow-motion, and reverse playback effects. For more information, see [“Timescale and Direction During Playback,”](#) on page 3-5.

The Logger lets you navigate to a specified timepoint in a recording, or skip forward or backward by a specified time increment. For procedures, see [“Navigating to A Specific Time in A Recording,”](#) on page 4-7 and [“Navigating Time,”](#) on page 5-5.

DIS only

In DIS, when you start the Logger and specify a file for playback, the Logger sends the recorded PDUs to UDP port 3000 (or another port, if specified in the *logger.mtl* file or on the command line.)

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.

3.1.3. Recording and Playback Using HLA Time Management

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



HLA Time Management is not supported in the text interface.

Recording Using Time Management

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

Playback Using Time Management

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

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

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

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

Once it starts playback, the Logger advances at whatever rate is determined by the time scale setting. For more details, see “[Timescale and Direction During Playback](#),” on page 3-5.

Some Logger Features are Restricted When You Use Time Management

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

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

3.2. Timescale and Direction During Playback

The Logger lets you play a recording at any speed within the capabilities of your platform, forward or backward.

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. A negative timescale causes reverse playback. Figure 3-2 shows a set of typical timescale values and their effect on speed and direction.

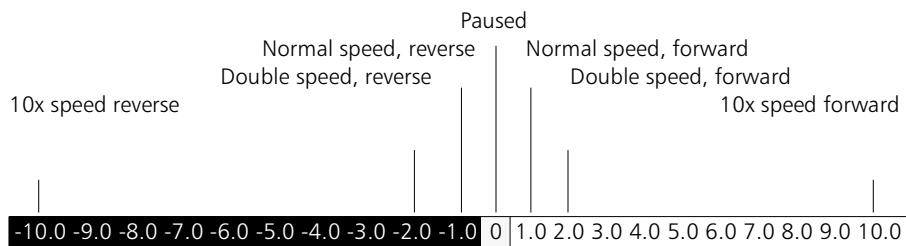


Figure 3-2. Sample timescale values

For procedures, see [“Playing Back Recordings,”](#) on page 5-3 and [“Setting the Timescale at Startup,”](#) on page 5-6.

3.2.1. Using Time Scaling with Time Management

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



You cannot run a recording in reverse if you are using Time Management.

3.3. 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.

3.3.1. 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 [“Setting Configuration Options in the GUI,”](#) on page 4-21 and [“Controlling the DIS Heartbeat and Timeout,”](#) on page 5-6.

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 [“Setting Configuration Options in the GUI,”](#) on page 4-21 and [“Controlling the DIS Heartbeat and Timeout,”](#) on page 5-6.

3.3.2. Updating Entity Information Quickly After A Time Jump

During playback, you can jump to a specified 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.

Entity maintenance is applied when you issue a Skip Time command or a Set Time command.

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.

3.3.3. 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.

This scaling feature is active by default. To disable it, use the `-d 1` option when you start the Logger.

DIS only

Scaling must be enabled in order for the Logger to send PDUs while in the paused state. If a file contains bundled PDUs, the Logger does not unbundle them on playback, and therefore cannot scale their speed and velocity.

3.4. Specifying the End-of-File Action

When the Logger reaches the end of a file, it can enter a stopped or paused state, or it can wrap to the other end of the file and continue playback.

During reverse playback, if wrap is in effect and the beginning of the recording is reached, the recording wraps to the end and continues reverse playback.

You can set the end-of-file action in the Configuration dialog box or with a startup option. See [“Setting Configuration Options in the GUI,”](#) on page 4-21 and [“Controlling End-of-File Behavior,”](#) on page 5-5.

3.5. Exporting Logger Data to Databases

The Logger’s SQL export feature allows you to export a significant portion of the simulation data stored in a Logger recording to a SQL database. Once the data is in the database, you can manipulate and analyze the data as you please, including transferring it to a spreadsheet, another database, a simulation analysis program, and so on.

The Logger provides two ways to export data to a SQL database: the `sqlSender` plug-in and the `sqlSender` application. The `sqlSender` plug-in lets you export Logger data while you are playing a recording or while you are making a new recording. However, its use could affect the Logger’s ability to keep up with incoming traffic when recording, or with time advancement when playing a recording.

The `sqlSender` application is a stand-alone executable that operates directly on Logger recordings. The `sqlSender` application is a simple console-based application that does not provide any recording or playback capabilities.

3.6. HLA Support

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

3.6.1. FOM Flexibility

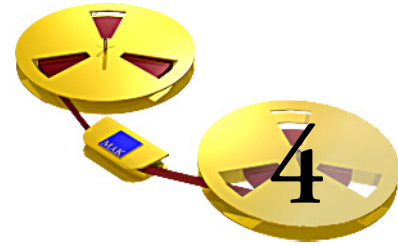
You can change the FED file between recording and playback. As long as the FED file you are using on playback contains object and interaction classes, and attributes and parameters with the same names and format as those that were recorded, you can play back your file.

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

3.6.2. 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 *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):

The Logger Windows.....	4-3
Starting the Logger	4-4
Recording Exercises.....	4-5
Playing Logger Recordings	4-6
Loading A Logger File for Playback	4-6
Playing A Logger File.....	4-6
Pausing Playback of A Logger File	4-6
Stopping Playback of A Logger File	4-6
Navigating to A Specific Time in A Recording	4-7
Jumping to the Beginning or End of A Recording	4-7
Moving to A Specific Time in A Recording	4-8
Skipping Forward or Backward in A Recording	4-8
Filtering Logger Input and Output	4-9
Filtering Options for DIS	4-9
Filtering Options for HLA	4-13
Saving and Loading Filtering Settings.....	4-14
Annotating A Logger Recording.....	4-16
Adding An Annotation.....	4-16
Saving An Annotation File.....	4-17
Opening An Annotation File.....	4-17
Importing An Annotation File.....	4-17
Moving to An Annotation Point.....	4-18
Editing An Annotation File	4-18
Deleting An Annotation	4-18
Clearing the Annotation List.....	4-19

Using The Logger

Setting the Timescale and Direction	4-19
Using the Timescale Slider	4-19
Entering A Timescale Value Numerically.....	4-20
Setting Configuration Options in the GUI.....	4-21
Summary of Startup Options for the Logger.....	4-23

4.1. The Logger Windows

The Logger has separate windows for playback and record modes. The Playback window has a set of navigation controls and a slider for scaling and reversing time. The Record window allows you to start and stop recording. [Figure 4-1](#) illustrates these windows. Both windows allow you to set configuration options.

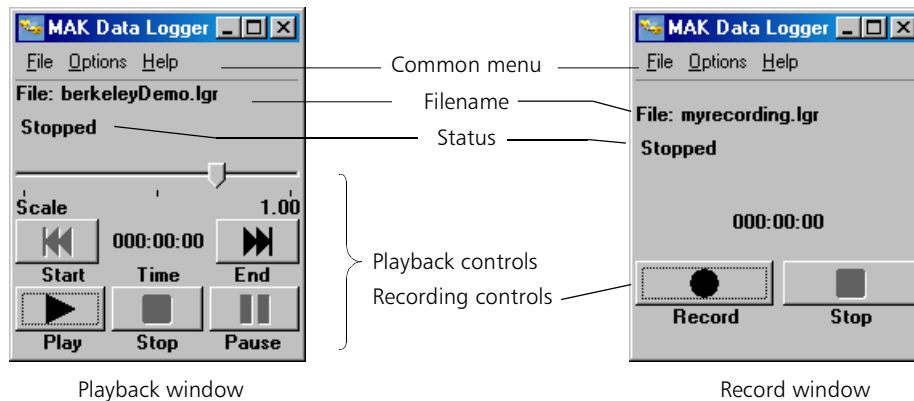


Figure 4-1. The Logger windows (DIS)

If you are running the Logger in HLA, the window has a check box for specifying use of Time Management ([Figure 4-2](#)).

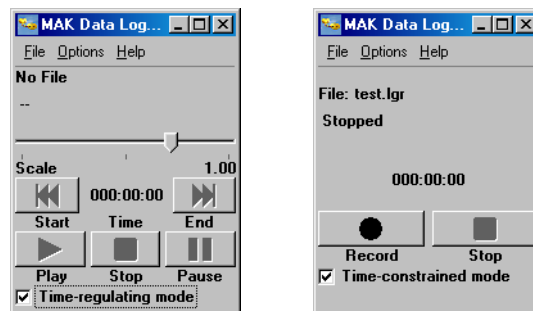


Figure 4-2. The Logger windows (HLA)

i

This chapter describes procedures that typical Logger users use frequently. Two GUI features, the SQL Send Plugin Control dialog box and the Logger Pipeline Stages Manager require a more advanced understanding of the Logger and are described in Chapter 6, [Exporting Logger Data to A Database](#), and “[Managing Pipeline Stages](#),” on page 8-24, respectively.

4.1.1. Starting the Logger

You can use startup options to specify many of the Logger's initial settings. [Table 4-2](#) lists startup options for the Logger.



Some options (such as the federation execution name in HLA, or the DIS port or destination address) cannot be set in the Logger windows. You must specify them in the startup command.

Starting the Logger in UNIX

To start the Logger:

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

```
Logger [options]
```

By default, the Logger opens up in playback mode.

Starting the Logger in Windows

Start Logger in any of the following ways:

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

```
Logger [options]
```

When the Logger window opens, you can close the command window if you want to.

By default, the Logger opens up in playback mode.

4.2. Recording Exercises

To record an exercise:

1. In either the Playback window or Record window, choose File → Record Exercise. The Select File dialog box opens. The Select File dialog box is a standard open file dialog box for your windowing system.
2. If necessary, navigate to the directory where you want to create the file, then enter 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 OK. If the Logger was in playback mode, it changes to record mode.
4. For HLA only, to record using Time Management, check the Record In Time Constrained Mode check box.

The Logger enters record mode in the state specified in the Configuration dialog box.

- To begin recording, in the Record window, click the Record button.
- To stop recording, in the Record window, click the Stop button.

The Status field displays either Recording or Stopped depending on the status.

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.

4.3. Playing Logger Recordings

If a Logger file is at its beginning or end, the Status field displays Start or End. Otherwise, it displays Stopped, Paused, or Playing to reflect the Logger's current operating state.

4.3.1. Loading A Logger File for Playback

To load a Logger file for playback:

1. In either the Playback window or Record window, choose File → Play Logger File. The Select File dialog box opens. This is a standard open file dialog box for your windowing system.
2. If necessary, navigate to the desired directory, then select the file to be played.
3. Click OK. If the Logger was in record mode, it changes to playback mode.
4. For HLA only, if you want to play the recording using Time Management, check the Play In Time Regulated Mode check box.



Playback options are restricted in time-managed mode. See [“Some Logger Features are Restricted When You Use Time Management,”](#) on page 3-4.)

4.3.2. Playing A Logger File

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

- To play a Logger file, in the Playback window, click the Play button.

4.3.3. Pausing Playback of A Logger File

- To pause playback of a Logger file, in the Playback window, click the Pause button.



You cannot pause the Logger if you are playing a recording in time-managed mode.

4.3.4. Stopping Playback of A Logger File

- To stop playback of a Logger file, in the Playback window, click the Stop button.



If you are playing a recording with Time Management enabled, when you stop playback, the Logger resigns from the federation. To play the recording again, you must restart the entire federation.

4.4. 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 forward or backward in time by a specified offset
- ♦ Jump to a point defined by an annotation.

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. For more information, see section 3.4, [“The End-of-File Action”](#), and section 4.6, [“Setting Configuration Options”](#).
- ♦ During slow-speed playback, when you use the Skip Time or 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 3-7.
- ♦ You cannot navigate a recording if you are using Time Management.

4.4.1. Jumping to the Beginning or End of A Recording

- To jump to the beginning of a recording, in the Playback window, click the Start button.
- To jump to the end of a recording, in the Playback window, click the End button.

4.4.2. 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. In the Playback window, choose Options → Set Time. The Set Time dialog box opens ([Figure 4-3](#)).

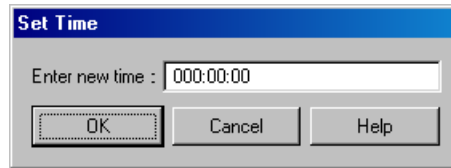


Figure 4-3. Set Time dialog box

2. In the Enter New Time box, enter a time in hours:minutes:seconds format. This time is measured from the start of the recording.
3. Click OK.

4.4.3. Skipping Forward or Backward in A Recording

The Skip Time feature lets you move forward or backward from the current point in a recording by a specified amount of time.

To skip forward or backward in a recording:

1. In the Playback window, choose Options → Skip Time. The Skip Time dialog box opens ([Figure 4-4](#)).

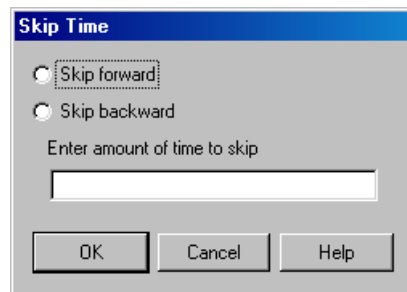


Figure 4-4. Skip Time dialog box

2. Select the Skip forward or Skip backward option.
3. In the Enter Amount of Time to Skip box, enter a time in hours:minutes:seconds format.
4. Click OK.

4.5. 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, entity ID, or PDU. In HLA, you can filter by class and object. You can also control which objects you subscribe to (see “[Specifying Class and Interaction Subscriptions](#),” on page 2-24.) During a Logger session, filtering settings apply to both recording and playback. Settings do not persist between sessions.

- To enable (0) or disable (1) filtering globally, start the Logger with the `-y` startup option or set the `noFiltering` parameter in `logger.mtl`.

4.5.1. Filtering Options for DIS

This section describes the filtering options for DIS – filtering by entity type, entity ID, and PDU kind.

Filtering by Entity Type

To filter Logger input or output by entity type:

1. Choose Options → Filter By Entity Type. The Filter by Entity Type dialog box opens ([Figure 4-5](#)). The Common Entity Types window lists entity types that you have previously added to the list (either manually or by configuring the list in `logger.mtl`).

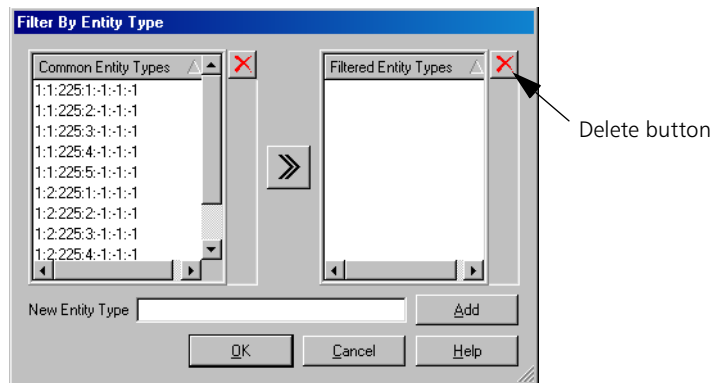


Figure 4-5. Filter By Entity Type dialog box

2. If the entity type that you want to filter is not listed in the Common Entity Types window, enter the 7-digit enumeration for the entity in the New Entity Type box. Separate enumeration values with colons (:).



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

3. Click Add. The entity type is added to the Common Entity Types list.

4. To filter out an entity type, select it in the Common Entity Types list.
5. 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` parameter in *logger.mtl*, as in the following example:

```
(setq commonEntityTypes
  (list
    "1:1:225:1:-1:-1:-1"      ; US tank
    "1:1:225:2:-1:-1:-1"      ; US armored vehicle
    "1:1:225:3:-1:-1:-1"      ; US armored utility vehicle
    "1:1:225:4:-1:-1:-1"      ; US self-propelled artillery
    "1:1:225:5:-1:-1:-1"      ; US towed artillery
  )
)
```

Filtering by Entity ID

In the Filter by EntityID-PDUKind Pairs dialog box, objects that are being recorded or published are marked with a check mark. Objects being filtered are marked with a red x. You can sort by any column by clicking the column title.

To filter Logger input or output by entity ID:

1. Choose Options → Filter By Entity ID. The Filter by EntityID-PDUKind Pairs dialog box opens (Figure 4-6).

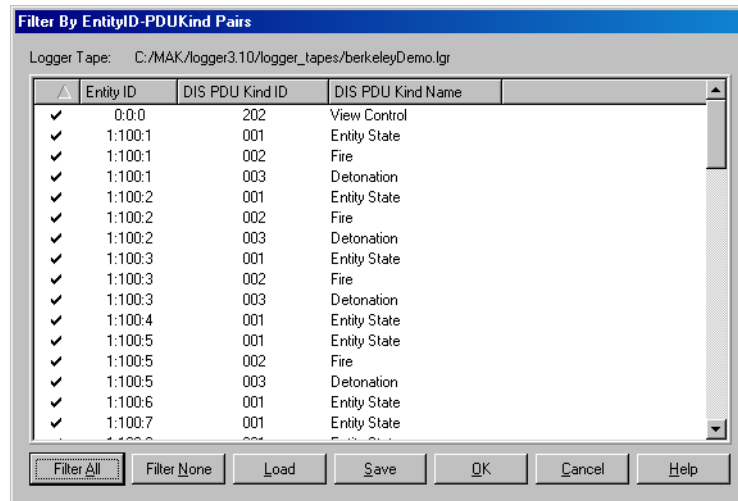


Figure 4-6. Filter by EntityID-PDUKind Pairs dialog box

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.

Filtering by PDU Kind

To filter Logger input and output by PDU kind:

1. Choose Options → Filter By PDU Kind. The Filter by PDU Kind dialog box opens (Figure 4-7).

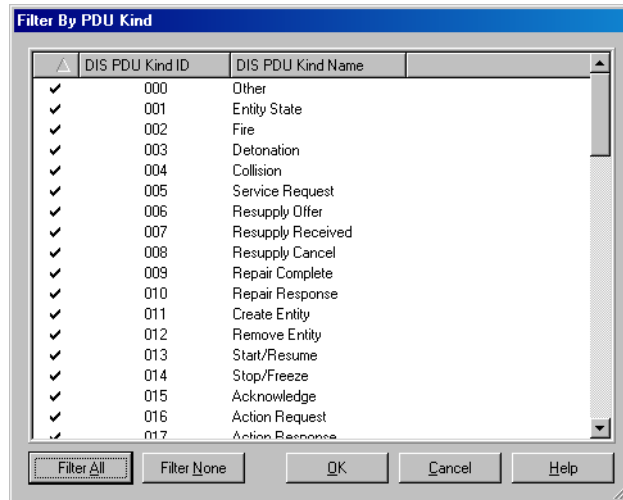


Figure 4-7. Filter By PDU Kind dialog box

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 *logger.mtl*, as in the following example:

```
(setq customPdu
  (list
    (list "MyCustomPdu1" 186 12 24)
    (list "MyCustomPdu2" 187 12 24)
  )
)
```

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.

4.5.2. Filtering Options for HLA

This section describes the filtering options for HLA – filtering by object or by class.

Filtering by Object

To filter Logger output by object:

1. Load a recording.
2. Choose Options → Filter By Object. The Filter by Object dialog box opens (Figure 4-8).

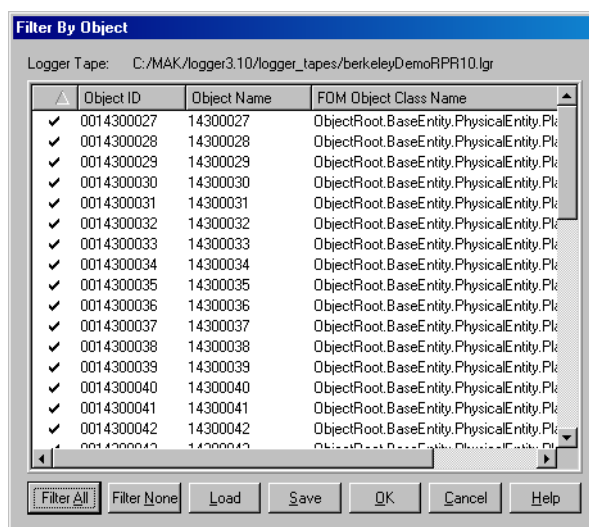


Figure 4-8. Filter By Object dialog box

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 objects, click the Filter All button.
5. To filter out no objects, click the Filter None column.

Filtering by Class

To filter Logger input and output by class:

1. Choose Options → Filter By Class. The Filter by Class dialog box opens (Figure 4-9).

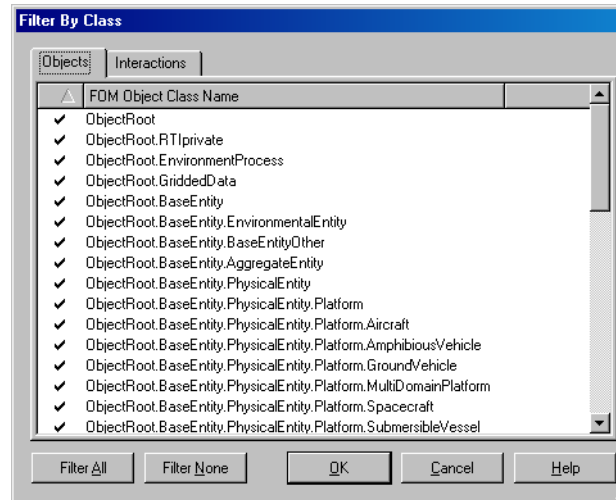


Figure 4-9. Filter By Class dialog box

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.

4.5.3. Saving and Loading Filtering Settings

You can save and re-use the filtering settings in the Filter By EntityID-PDUKind Pairs dialog box (DIS) and the Filter by Object dialog box (HLA). When you load settings, if the list of entities or objects in the current session does not exactly match the list in the saved settings file, the Logger applies settings to any matching types and leaves non-matching types as they are. The settings files are text files, so if you want to, you can edit them to create a core set of filter options that you always want to have available.

In addition to saving settings, this option provides a text file listing all of the objects in the recording, which you may want to examine for other purposes.

Saving Filtering Settings

To save filtering settings:

1. In the filtering dialog box, click Save. A standard Save dialog box opens.
2. Enter a name for the settings file.
3. Click Save.

Loading Filtering Settings

To load filtering settings:

1. In the filtering dialog box, click Load. A standard Open dialog box opens.
2. Select the settings file you want to load.
3. Click Load. The settings are loaded and the dialog box is updated.

4.6. Annotating A Logger Recording

An annotation specifies a particular elapsed time in a Logger recording to which you can jump. It differs from the Set Time command in that you can save annotations to a file and reuse them. Optionally, you can associate text with an annotation.

The maximum number of annotations is 1000. You can override this limit with the Logger API.

4.6.1. Adding An Annotation

To annotate a Logger recording:

1. In the Record or Playback window, choose Options → View Annotations. The View Annotations window opens (Figure 4-10).

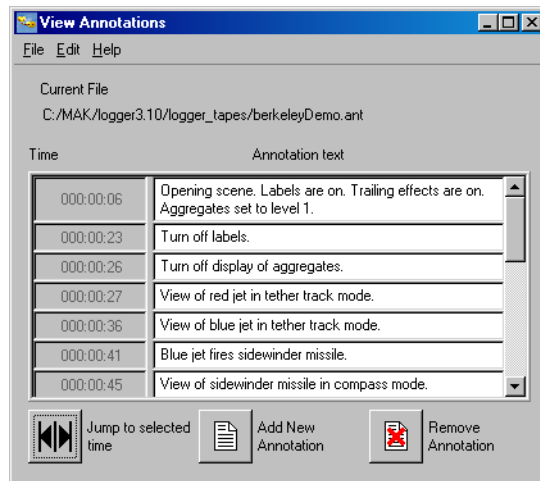


Figure 4-10. View Annotations window

2. Play a Logger recording or begin recording an exercise.
3. At any point while the Logger is playing, recording, or paused, choose Edit → Add (or click the Add New Annotation button.) The Logger adds a new line to the View Annotations window, at the current time.
4. Click in the Annotation Text column of the new line and type the text you want to associate with this annotation.

You can leave the View Annotations window open and continue to add annotations as often as you want. Annotations can be as long as you want.

4.6.2. Saving An Annotation File

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

To save annotations to a file:

1. Create annotations as described in [“Adding An Annotation,”](#) on page 4-16.
2. In the View Annotations window, choose File → Save As. The Save As dialog box opens.
3. Navigate to the directory in which you want to save the annotation file. The default directory is the directory from which you started the Logger.
4. Enter a name for the annotation file.
5. Click Save.

4.6.3. Opening An Annotation File



If you open an annotation file, any annotations currently listed in the View Annotations window are overwritten.

To open an annotation file:

1. In the Record or Playback window, choose Options → View Annotations.
2. In the View Annotations window, choose File → Open. An Open dialog box opens.
3. Select an annotation file.
4. Click Open. The annotations in the file are displayed.

4.6.4. Importing An Annotation File

If you import an annotation file, the annotations in the file are added to the annotations listed in the View Annotations window.

To import an annotation file:

1. In the Record or Playback window, choose Options → View Annotations.
2. In the View Annotations window, choose File → Import. An Open dialog box opens.
3. Select an annotation file.
4. Click Open. The annotations in the file are added to the list.

4.6.5. Moving to An Annotation Point

To move a recording to an annotated point:

1. In the Playback window, choose Options → View Annotations.
2. Open an annotation file or create new annotations for the recording you are working with.
3. Do one of the following:
 - Double-click the time point to which you want to jump.
 - Select the annotation to which you want to move the recording and click the Jump to Selected Time button. (A selected annotation has a yellow background.)

Notes:

- ♦ The recording does not have to be playing to move to a specific time. You can select an annotation and then start the recording. It will start at the selected time.
- ♦ This feature is not available if you are playing a recording using Time Management.

4.6.6. Editing An Annotation File

You can edit the text associated with an annotation. You cannot edit the time points, except to delete them.

To edit an annotation:

1. Click in the text for the annotation you want to edit.
2. Edit the text.
3. Save your changes.

4.6.7. Deleting An Annotation

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

To delete an annotation:

1. In the View Annotations window, select the annotation you want to delete.
2. Choose Edit → Remove, or click Remove Annotation. The annotation is removed from the list.

If you removed an annotation from an annotation list that you had previously saved to a file, save the edited list with the File → Save As command.

4.6.8. Clearing the Annotation List

Clearing the View Annotations window removes all annotations from the display. If an annotation file is open, it closes the annotation file without closing the View Annotations window.

- To clear the View Annotations, in the View Annotations window, choose File → Clear.

i

If you have opened a list of annotations from a file, clearing the View Annotations window is not the same as deleting all of the annotations in the list. Clearing the window closes the file without editing it. If you open an annotation file and remove each annotation with the Edit → Remove command, the file is still open, but now it does not have any annotations in it.

4.7. Setting the Timescale and Direction

The Logger lets you play a recording forward or backward at any speed, limited only by the capabilities of your hardware. You can set the timescale in the following ways:

- ♦ Adjust a slider control in the Logger playback window
- ♦ Specify an initial timescale at Logger startup
- ♦ Enter the desired timescale into a dialog box.

These options are described in the following sections. For conceptual information about timescale values, particularly in the context of HLA Time Management, see section 3.2, “[Timescale and Direction During Playback](#)”.

4.7.1. Using the Timescale Slider

The Timescale slider control in the Logger playback window is the most convenient way to change playback speed and direction. In its default configuration, the slider contains nineteen preset values ranging from -10 to +10.

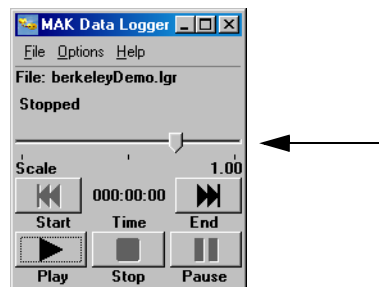


Figure 4-11. Timescale slider

- To change the timescale, in the Playback window, drag the slider button.

Adjusting the Time Scale from the Keyboard

If you need finer control of the time scale than the mouse provides, you can adjust it from the keyboard.

To adjust the time scale from the keyboard:

1. In the Playback window, click once on the slider button to give it focus.
2. Use the left and right arrows on your keyboard to step through the values.

4.7.2. Entering A Timescale Value Numerically

If the time scale you want is not available through the slider control, you can specify it directly.

To set the time scale numerically:

1. Choose Options → Set Timescale. The Set Time Scale dialog box opens (Figure 4-12).

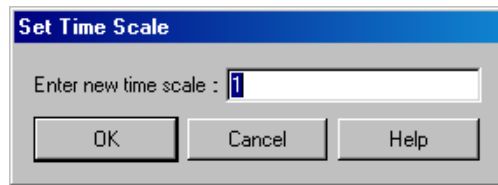


Figure 4-12. Set Time Scale dialog box

2. Enter the desired time scale.
3. Click OK.

You can enter the timescale value in scientific notation, for example, $1e+4$ instead of 10,000.



Although the Logger lets you specify a very large timescale (beyond $1e+35$), the use of an unusually large timescale may produce unexpected results because of hardware limitations.

If a recording is played at a very high speed, the Logger may be unable to keep up, especially during passages that contain a high concentration of data. If the Logger falls significantly behind, it continues to process the messages as quickly as possible. It does not discard any messages.

4.8. Setting Configuration Options in the GUI

You can configure the following aspects of Logger behavior in the GUI:

- ♦ Play action – what the Logger does when you open a recording
- ♦ Record action – what the Logger does when you create a new recording
- ♦ End-of-file action – what the Logger does when it reaches the end of a recording
- ♦ DIS heartbeat
- ♦ DIS timeout.

To set configuration options:

1. Choose Options → Configuration. The Configuration dialog box opens (Figure 4-13).
2. Set options as desired. Table 4-1 describes the configuration options.

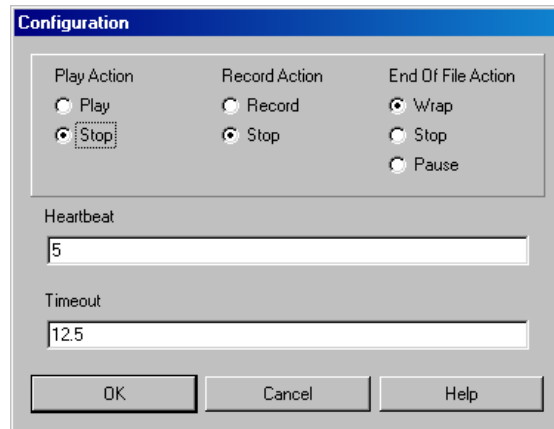


Figure 4-13. The Configuration dialog box



Configuration changes are valid only for a given Logger session. They are not retained for future sessions. If you want to create a set of startup options that you can reuse, write a script or batch file that uses startup options to specify the configuration you want.

Table 4-1: Logger Configuration Options

Option	Description
Play Action	Determines whether the Logger enters a stopped state (default) or begins playing immediately when a file is opened for playback.
Record Action	Determines whether the Logger enters a stopped state (default) or begins recording immediately when a file is opened for recording.
End of File Action	Determines the Logger's behavior when it reaches the end of a recording. It can stop, pause, or wrap around to the other end of the recording to continue playback. Default: Stop. See "Specifying the End-of-File Action," on page 3-8.
Heartbeat (DIS only)	Determines how frequently (in seconds) the Logger generates interim PDUs for each entity during slow-speed playback. For information, see "Generating Interim PDUs During Slow-Speed DIS Playback," on page 3-6.
Timeout (DIS only)	Determines how long the Logger continues to generate interim PDUs for an entity once that entity's PDUs have stopped occurring in the recording. See "Generating Interim PDUs During Slow-Speed DIS Playback," on page 3-6.

4.9. Summary of Startup Options for the Logger

Table 4-2 lists startup options. Startup options override the comparable settings in the *logger.mtl* file. They are grouped by mode and indicate if they are for HLA only or DIS only.

Table 4-2: Startup options (Sheet 1 of 4)

Option	Description
Record and Playback	
<code>-a {"site:host:entity" "hla_object_name"}</code>	DIS: Specifies the site:host:entity for remote control of the Logger. HLA: Specifies the object name (global object designator in VR-Link.)
<code>-B recvBSz:sendBSz</code>	DIS: Specifies the receive and send buffer size.
<code>-c file_name</code>	Specifies the Logger configuration file (MTL file).
<code>-D {0 1}</code>	Allow files to be deleted in response to Logger Control messages. 0 = Do not allow deletion. 1 = Allow deletion.
<code>-f {v1RPRxx library:init_data}</code>	HLA: Specifies the FOM Mapper to use if you are not using the default FOM.
<code>-F fedfile</code>	HLA: Specifies the FED file for the exercise.
<code>-g {w t b h}</code>	Specifies interface to use, where: w = Graphical user interface (default). t = Text interface. b = Both. h = Hidden (no interface).
<code>-G language</code>	Specify the language for the Logger GUI to use.
<code>-h</code>	Displays a brief summary of Logger command-line arguments.
<code>-i {0 1}</code>	HLA: Ignore HLA advisory messages. 0 = Ignore advisory messages. 1 = Do not ignore advisory messages.
<code>-I {0 1}</code>	DIS: Use asynchronous I/O. 0 = Do not use asynchronous I/O (default). 1 = Use asynchronous I/O.
<code>-k script</code>	Windows: Specifies a script file to run when the Logger starts.

Table 4-2: Startup options (Sheet 2 of 4)

Option	Description
<code>-L {0 1 2}</code>	Specifies the response to Logger Control messages for remote control. 0 = Do not respond to Logger Control messages. 1 = Interpret Logger Control messages only if they are addressed to the entity ID specified by the <code>-a</code> argument (default). 2 = Interpret all Logger Control messages
<code>-m {0 1}</code>	Send a Logger control PDU each time you change the state of the Logger. 0 = Do not send Logger control PDU. 1 = Send PDU.
<code>-n genNotify:lgrNotify</code>	Specify a general notification level and a Logger-specific notification level. Range 0-4, from least verbose to most verbose.
<code>-N federate_name</code>	Specifies the federate name for the Logger federate.
<code>-P dis_port</code>	DIS: 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>-Q {0 1}</code>	Use 64 bit mode. This mode supports recordings greater than 2 GB. 0 = Do not use 64 bit mode (default). 1 = Use 64 bit mode.
<code>-w {0 1}</code>	HLA: Enables or disables use of Time Management, but does not select it. 0 = Disable Time Management. 1 = Enable Time Management. Default: 0.
<code>-x ExName</code>	HLA: Sets federation execution name. Default: VR-Link.
<code>-x exerciseID</code>	DIS: Specifies the exercise ID.
<code>-y {0 1}</code>	Enable or disable filtering. 0 = Enable filtering. 1 = Disable filtering. Default: 0.
<code>-Y {0 1}</code>	Enable or disable articulated parts scaling. 0 = Enable scaling of articulated parts. 1 = Disable scaling of articulated parts. Default: 0.
<code>-z logfilename</code>	Specifies a Logger log file. Default: <i>logger.txt</i> .

Table 4-2: Startup options (Sheet 3 of 4)

Option	Description
<code>-z lownum:highnum</code>	Specifies VR-Forces version-specific values to filter out messages between the front-end and back-end. If filtering is enabled in the <i>logger.mtl</i> file, specifying <code>-z highnum:lownum</code> (for example, <code>-z 5:4</code>) overrides the <i>logger.mtl</i> file and disables filtering.
Record Mode Only	
<code>-b state</code>	Specifies whether the Logger enters a stopped state or begins recording immediately when opening a file for recording. r = Record. s = Stopped (default).
<code>-P {0 1}</code>	HLA: Subscribe passively. Do not force federates to send all updates to the Logger. Record only those updates that other federates subscribe to. 0 = Active subscription. 1 = Passive subscription.
<code>-r filename</code>	Causes the Logger to open in record mode. Specifies the destination file for recorded data.
<code>-R {0 1}</code>	Specifies whether or not to record Logger Control messages during remote control. 0 = Do not record Logger Control messages (default). 1 = Record Logger Control messages.
<code>-S mcastAddr</code>	DIS: Subscribes the Logger to the multicast address specified by <i>mcastAddr</i> . To subscribe to several multicast addresses, use this option more than once in the command. See “Subscribing to Multicast Groups,” on page 2-28.
<code>-S sublist.mtl</code>	HLA: Specifies a subscription control file to load.
<code>-t hostname:port</code>	Specifies the TCP host to get data from.
<code>-u {0 1}</code>	HLA: Enable and select or disable constrained-time mode. Overrides <code>-w</code> . 0 = Do not enable Time Management (default). 1 = Enable Time Management.
Playback Mode Only	
<code>-A addr</code>	DIS: 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 “Specifying A Destination Address for Outgoing and Incoming Data,” on page 2-27.
<code>-C output_port</code>	Specifies the TCP port to send data to.

Table 4-2: Startup options (Sheet 4 of 4)

Option	Description
<code>-d {0 1}</code>	Disables the generation of velocity and acceleration scaling data when playing at speeds above or below normal. See “Velocity and Acceleration Scaling,” on page 3-7. 0 = Enable data scaling. 1 = Disable data scaling.
<code>-e action</code>	Specifies the Logger’s action upon reaching the end of the file. p = Pause at the file’s end. w = Wrap to the other end of the recording and continue. s = Stop (default).
<code>-l multicastTtl</code>	DIS: Specifies the multicast time to live value. Default: 2 seconds.
<code>-l lookahead</code>	HLA: Specify the lookahead, in seconds.
<code>-o state</code>	When a file is opened for playback, specifies whether the Logger enters a stopped state, or begins playing immediately. p = Playing. s = Stopped (default).
<code>-O</code>	Dumps output to <i>screendump.txt</i> .
<code>-p filename</code>	Causes the Logger to open in playback mode; specifies the source file to be played.
<code>-q {0 1 2}</code>	HLA: Specifies an interpretation for message tags. 0 = None; 1 = Time interval from last packet with message in tape-time; 2 = Interval is simulation time.
<code>-s timescale</code>	Sets the initial playback timescale factor. For more information, see “Timescale and Direction During Playback,” on page 3-5.
<code>-T timepoint</code>	Sets the timepoint in the file where playback begins. Enter the <i>timepoint</i> in HH:MM:SS format. Default: 00:00:00
<code>-U {0 1}</code>	HLA only: Enable and select or disable regulated-time mode. Overrides <code>-w</code> . 0 = Do not enable Time Management (default). 1 = Enable Time Management.



Using The Text Interface

The text (command-line) interface has the same features as the graphical user interface, plus script-based recording and playback capability.

Starting the Text Interface	5-2
Recording Exercises	5-2
Beginning A Recording Session in Record Mode	5-2
Logger Run-time Commands for Recording	5-3
Displaying the Status of A Recording	5-3
Playing Back Recordings	5-3
Basic Logger Playback Commands	5-4
Displaying Playback Status	5-4
Beginning A Playback Session in Play Mode	5-4
Navigating Time	5-5
Controlling End-of-File Behavior	5-5
Setting Playback Timescale and Direction	5-6
Setting the Timescale at Startup	5-6
Controlling the DIS Heartbeat and Timeout	5-6
Setting the Heartbeat Rate	5-7
Setting the Timeout	5-7
Using A Script to Control Recording and Playback	5-7
Controlling Processing with the Wait Command	5-8
Stopping A Script	5-8
Summary of Logger Run-Time Commands	5-9

5.1. Starting the Text Interface

To start the Logger's text interface:

1. Open a command window.
2. Enter the following command:

```
Logger -g t
```

The Logger starts running and waits for instructions.

At startup, a message shows the NetSocket status.

```
NetSocket ok for port 3000 to 123.456.789.255
```

You can start the Logger in record or playback mode. See [“Recording Exercises,”](#) on page 5-2 and [“Playing Back Recordings,”](#) on page 5-3.



HLA Time Management is not supported in command-line mode.

5.2. Recording Exercises

- To record an exercise if the Logger is running, in the command window, enter the following command:

```
record ../logger_tapes/logger_file.lgr
```

where *logger_file.lgr* is the name of the file to which you want to record the exercise.

To record an exercise if the Logger is not running:

1. In a command window, change to the *loggerx.x/bin5*, *./binHLA13*, or *./binHLA1516* directory.
2. Enter the following command:

```
Logger -g t -r ../logger_tapes/logger_file.lgr
```

5.2.1. Beginning A Recording Session in Record Mode

By default, when you start the Logger in record mode, it begins in a stopped state.

- To begin recording immediately, use the **-b** option to set the record action as in this example:

```
Logger -b r -r logger_file.lgr
```

5.2.2. Logger Run-time Commands for Recording

Table 5-1 lists the run-time commands for the Logger.

Table 5-1: Logger run-time commands for recording exercises

Command	Description
go	Begins or resumes recording.
info	Displays a brief status report.
quit	Closes the file and exits the Logger.
record <i>filename</i>	Starts recording to the specified file. If the file already exists, it is overwritten.
stop	Stops recording without ending the session.

5.2.3. Displaying the Status of A Recording

At any time while you are recording an exercise, you can enter the `info` command (or the letter `i`) to display a brief status summary. The summary shows the destination file name, the time elapsed since the start of recording, and the current operating status.

```
> info
FILE: my-exercise.lgr
TIME: 000:01:43
SCALE: 1.000000
STATE: Recording
>
```

5.3. Playing Back Recordings

- To play a Logger file if the Logger is already running, in the command window, enter:

```
open ../logger_tapes/logger_file.lgr
go
```

To play a Logger file if the Logger is not running:

1. In a command window, change to the `loggerx.x/bin5`, `./binHLA13`, or `./binHLA1516` directory.
2. Enter the following command:

```
Logger -g t -o p 2 -p ../logger_tapes/logger_file.lgr
```

If you are running the DIS version of the Logger, a status message similar to the following is displayed:

```
NetSocket ok for port 3000 to 123.456.789.255
Logger file my-exercise.lgr is version 2 (writing)
```

5.3.1. Basic Logger Playback Commands

The Logger starts in a playing, paused, or stopped state. [Table 5-2](#) describes the most common commands for playback.

Table 5-2: Logger run-time commands for playing recordings

Command	Description
go	Begins or resumes playback.
info	Displays status information.
open <i>filename</i>	Opens the specified file for playback.
pause	Pauses playback. When paused, the Logger stops time and, for DIS only, repeatedly rebroadcasts each entity's most recent entity state PDU.
quit	Closes the file and exits the Logger.
stop	Stops playback. When stopped, the Logger stops time and does not transmit data.

5.3.2. Displaying Playback Status

- To display a playback status summary, enter the `info` command (or the letter `i`) at the Logger prompt.

The summary shows the source file name, the current timepoint in the Logger file, the timescale (playback speed), and the current operating status.

```
> info
FILE: my-exercise.lgr
TIME: 000:00:09
SCALE: 1.000000
STATE: Playing
```

5.3.3. Beginning A Playback Session in Play Mode

By default, when you start the Logger in playback mode, it is in a stopped state.

- To begin playing immediately, use the `-o p` option to set the play action as in this example:

```
Logger -o p -p logger_file.lgr
```

5.4. Navigating Time

During playback, four commands let you navigate to a specific point in a recording:

Table 5-3: Runtime commands during playback

Command	Description
<code>start</code>	Moves to the beginning of the recording.
<code>end</code>	Moves to the end of the recording.
<code>time</code>	Sets the timepoint to a specified time. The command format is: <code>time HH:MM:SS</code>
<code>skip</code>	Causes the Logger to jump forward or backward in time by a specified offset. Enter the offset in an HH:MM:SS format, or as a number of seconds. Use a negative value to skip backward in time. For example: <code>skip 00:02:30</code> <code>skip -170</code>

5.4.1. Controlling End-of-File Behavior

► To specify the end-of-file action, start the Logger with the `-e` option, as follows:

```
Logger -p logger_file.lgr -e {p|w|s}
```

where:

- ♦ `p` causes the Logger to enter a paused state at the end of a file.
- ♦ `w` causes the Logger to wrap around to the other end of the file and continue playback
- ♦ `s` causes the Logger to stop at the end of a file.

5.5. Setting Playback Timescale and Direction

You can play a recording at any practical speed, forward or backward.

- To set the playback speed and direction, enter the **scale** command:

```
scale timescale
```

where *timescale* is the factor by which time is to be multiplied. For information about choosing timescale values, see [“Timescale and Direction During Playback,”](#) on page 3-5.

You can enter the timescale value in scientific notation, as in the following example for 10,000:

```
scale 1e+4
```



Although the Logger lets you specify a very large timescale (beyond 1e+35), the use of an unusually large timescale may produce unexpected results because of hardware limitations, and is not recommended.

If a recording is being played at a high speed, the Logger may be unable to keep up, especially during passages that contain an unusually high concentration of data. If the Logger falls behind, it continues to process messages as quickly as possible. No messages are discarded.

5.5.1. Setting the Timescale at Startup

- To specify a timescale at Logger startup, use the **-s** argument, as in the following example:

```
Logger -p logger_file.lgr -s 2.5
```

5.6. Controlling the DIS Heartbeat and Timeout

During slow-speed playback in DIS, the Logger generates interim PDUs as needed to maintain an appropriate heartbeat rate and timeout threshold for each entity in the recording. Heartbeat and timeout concepts are explained in section 3.3.1, [“Generating Interim PDUs During Slow-Speed DIS Playback”](#).

5.6.1. Setting the Heartbeat Rate

The heartbeat rate determines how frequently the Logger generates interim PDUs for each entity. By default, the Logger maintains a heartbeat rate of five seconds.

- To set the heartbeat rate, use the `heartbeat` command:

```
heartbeat n
```

where *n* represents the maximum allowable time between PDUs.

- To view the current `heartbeat` setting, enter the command name without a value.

5.6.2. Setting the Timeout

The timeout threshold determines how long the Logger continues to generate interim PDUs for an entity once the entity's PDUs stop occurring in the recording. The Logger's default timeout setting is twelve seconds.

- To set the timeout value, use the `timeout` command:

```
timeout n
```

where *n* represents the number of recorded seconds the Logger should wait before timing an entity out.

- To view the current `timeout` setting, enter the command without a value.

5.7. Using A Script to Control Recording and Playback

A script is a text file that can control the Logger in either playback or record mode. Scripts are especially useful for creating a prepared demonstration, or for automatically bypassing unwanted segments during playback.

A script can contain any Logger run-time commands listed in [Table 5-4](#). Create the script in any text editor, then redirect it to the Logger as in these examples:

```
Logger -p my_exercise.lgr < my_play_script.txt
Logger -r new_exercise.lgr < my_record_script.txt
```

At startup, the Logger enters a playing or recording state, then immediately begins processing commands in the script.



On Windows, you cannot redirect the script file as input to the Logger. To run a script file, start the Logger with the `-k` startup option, for example:

```
Logger -k myscript.txt
```

5.7.1. Controlling Processing with the Wait Command

The `wait` command controls how long the Logger runs in its current state before processing the next command in the script. When you use the `wait` command, you can specify a time interval or change in mode. See [Table 5-4](#) for a list of the wait modes. The sample script in [Figure 5-1](#) provides an example.

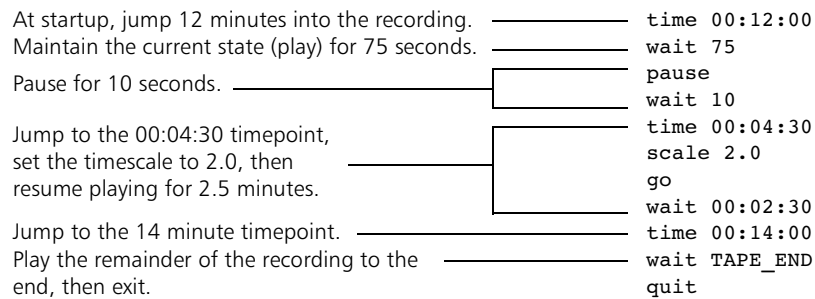


Figure 5-1. Sample Logger script

5.7.2. Stopping A Script

The Logger does not respond to keyboard input when running a script, so you cannot manually enter `quit` to stop execution.

- To stop a script, enter `Ctrl-C`.

5.8. Summary of Logger Run-Time Commands

Table 5-4 lists the commands you can use in the text interface.

Table 5-4: Logger text interface commands (Sheet 1 of 2)

Command	Description
Record Mode and Play Mode	
<code>fio64bit {0 1}</code>	Enables (1) or disables (0) use of 64 bit file IO.
<code>go</code>	Starts or resumes recording or playback from a previously stopped or paused state.
<code>stop</code>	Halts playback at the current timepoint, stops recording or transmission.
<code>quit</code>	Closes the Logger file and exits.
<code>info</code> or <code>i</code>	Displays a brief status report including the log file name, current timepoint, timescale, and current play state.
<code>h</code> , <code>help</code> , or <code>?</code>	Displays a summary of Logger run-time commands.
<code>wait mode</code>	Used only in scripts. Specifies how long the Logger performs its current action before processing the next command in the script. <i>Mode</i> can be one of following: ♦ <code>wait MODE_STOP</code> – wait until the Logger is in STOP state ♦ <code>wait MODE_PAUSE</code> – wait until the Logger is in PAUSE state ♦ <code>wait TAPE_END</code> – wait until the Logger reaches the end of the recording ♦ <code>wait TAPE_START</code> – wait until the Logger reaches the beginning of the recording ♦ <code>wait TAPE_TIME time</code> – wait until the Logger reaches tape-time <i>time</i> ♦ <code>wait time-interval</code> – wait (in real time) for <i>time-interval</i> seconds. Enter <i>time</i> in the HH:MM:SS format or as a number of seconds.
Play Mode Only	
<code>start</code>	Sets the current timepoint to the beginning of the recording.
<code>open filename</code>	Opens the specified Logger recording for playback.
<code>end</code>	Sets the current timepoint to the end of the recording.
<code>lookahead time</code>	HLA: Specifies the lookahead value.
<code>onEofAct mode</code>	Sets end of file action. <i>Mode</i> can be <code>wrap</code> , <code>stop</code> , or <code>pause</code> .
<code>onPbAct mode</code>	Sets Logger action when you load a file for playback <i>Mode</i> can be <code>play</code> or <code>stop</code> .

Table 5-4: Logger text interface commands (Sheet 2 of 2)

Command	Description
<code>pause</code>	Pauses playback at the current timepoint. In DIS, begins repeated resending of each entity's most recent entity state PDU with velocity and acceleration set to zero.
<code>scale factor</code>	Sets the timescale factor that determines playback speed and direction. Use 1.0 (default) for normal speed, 2.0 for double speed, 0.5 for half speed, and negative values for backward playback.
<code>time timepoint</code>	Navigates to a specific timepoint in the recording. Specify timepoint in HH:MM:SS format.
<code>skip offset</code>	Moves the current timepoint forward or backward by the specified time <i>offset</i> . Specify the offset in HH:MM:SS format or as a number of seconds. A positive offset causes a jump forward in time; a negative offset causes a backward jump.
<code>heartbeat sec</code>	(DIS only) During slow-speed playback, specifies the number of seconds the Logger allows before generating an interim PDU for a given entity. Default: 5. With no argument, heartbeat displays the current setting.
<code>timeout sec</code>	(DIS only) During slow-speed playback, specifies how long the Logger continues to transmit interim PDUs for an entity after the entity's PDUs have stopped occurring in the recording. Default: 12 recorded seconds.
<code>timeRegulating {0 1}</code>	(HLA only) Enables (1) or disables (0) time regulating mode when you are using HLA Time Management.
Record Mode Only	
<code>mirror {0 1}</code>	Enables (1) or disables (0) mirroring of user actions as Logger control messages.
<code>onRcAct mode</code>	Specifies Logger mode when it creates a file for recording. <i>Mode</i> can be <code>record</code> or <code>stop</code> .
<code>passiveSubscribe {0 1}</code>	(HLA only) Enables (1) or disables (0) passive subscription to HLA update messages.
<code>record filename</code>	Begins recording to the specified filename.
<code>timeConstrained {0 1}</code>	(HLA only) Enables (1) or disables (0) time constrained mode when you are using HLA Time Management.



Exporting Logger Data to A Database

This chapter describes how use the SQL database plug-in provided with the Logger. For a general introduction to the feature, see “[Exporting Logger Data to Databases,](#)” on page 3-8.

Connecting to SQL Database Servers.....	6-2
Using the sqlSender Plug-in	6-2
Loading the sqlSender Plug-in	6-2
Configuring and Enabling the sqlSender Plug-in in the Logger GUI	6-3
Configuring and Enabling the sqlSender Plug-in in logger.mtl	6-5
Disabling the sqlSender Plug-in.....	6-6
Using the sqlSender Application	6-7
Database Schema Used by the Logger SQL API	6-9
Importing Logger Data into A Spreadsheet	6-17

6.1. Connecting to SQL Database Servers

The Logger does not connect directly to SQL databases. It exports data to databases through ODBC drivers. To export data to a SQL database, you must have an ODBC driver installed on your computer and access to a SQL database. Although in principle you can use any ODBC 3.0 compliant driver, the plug-in has been tested only with MySQL and the default plug-in values support the MySQL ODBC driver.

ODBC drivers work with SQL databases by sending them queries, which are text sequences composed of pairs of keyword=value entries, separated by semi-colons. Different SQL database servers often use different keywords for the same concept. Therefore, you must configure the Logger to use your chosen database by providing the keywords, and values for the server you are using. For example, to specify the Data Source Name (DSN) and the Master Data Source Name, MySQL uses the keyword DSN (you must include the separation character (=) in the keyword specification.) If you were to specify a DSN keyword of DSN=, a DSN Name of mysql, and a Master DSN Name of DISLoggerData, the ODBC driver would create the string:
DSN=mysql;DSN=DISLoggerData.

6.2. Using the sqlSender Plug-in

To export data using the sqlSender plug-in:

1. Install and configure a database to receive the exported data. (Follow the installation instructions of your database application.)
2. Load the sqlSender plug-in.
3. Configure the plug-in. You can configure the plug-in in the *logger.mtl* file or in the Logger GUI.
4. Enable the plug-in for use while recording, playing, or both.

6.2.1. Loading the sqlSender Plug-in

A version of the sqlSender plug-in is installed in each of the bin directories under *./plugins*. To use the sqlSender plug-in, you have to configure the Logger to load it at startup. You can do this by loading all plug-ins at startup or specifically loading the sqlSender plug-in.

To specifically load the sqlSender plug-in, use the **pluginDir** and **plugins** MTL parameters in the *logger.mtl* configuration file, for example:

```
(setq pluginDir "../plugins/binHLA13")  
(setq plugins list "sqlSenderHLA13.dll")
```

For details about configuring the Logger to load plug-ins, see [“Loading Plug-Ins,”](#) on page 8-23.



On UNIX the file-extension for shared libraries is *.so*.

6.2.2. Configuring and Enabling the sqlSender Plug-in in the Logger GUI

You can configure the sqlSender plug-in through the Logger GUI. Settings made in the GUI override settings in the *logger.mtl* file.

After you configure the sqlSender plug-in, you can enable it for use with playback, recording, or both. If the plug-in is enabled for recording, when the Logger records an exercise, it records it to the Logger format and simultaneously exports it to the database. If you enable the plug-in for playback, when the Logger plays a recording, it sends packets over the network and exports them to the specified database.



The Logger does not save configuration data. You cannot configure it, but not enable it. You should configure the plug-in in the SQL Send Plugin Control dialog box only if you also plan to enable it.

If you want to export data to a database, but do not want to send the data over the network, use the *sqlSender* application to export the data (see “[Using the sqlSender Application](#),” on page 6-7).

To configure and enable the sqlSender plug-in:

1. Choose Options → SQL Sender Control. The SQL Send Plugin Control dialog box opens ([Figure 6-1](#)).

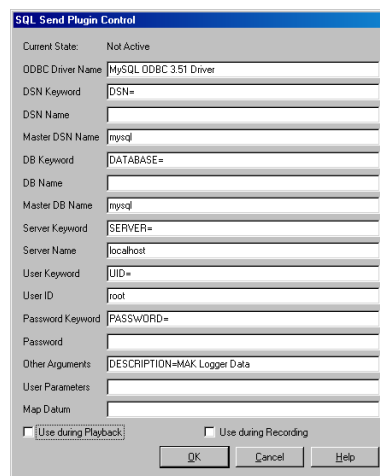


Figure 6-1. SQL Send Plugin Control dialog box

2. Enter or edit values for all parameters. (The Other Arguments and User Parameters fields might not be required. Consult your database administrator for details.) The parameters are described in [Table 6-1](#).
3. Check the Use During Recording box to enable the plug-in during record mode.
4. Check the Use During Playback box to enable the plug-in during playback.



If you disable the plug-in for both record and playback, when you click OK, all configuration information is lost and you must enter it again before you can re-enable the plug-in.

5. Click OK.

Table 6-1: SQL Send Plugin Control dialog box parameters (Sheet 1 of 2)

Parameter	Definition
ODBC Driver Name	Specifies the ODBC driver that you want to use. For example, if you are using the MySQL ODBC driver, version. 3.51, the correct name is MySQL ODBC 3.51 Driver.
DSN Keyword	Specifies the keyword and separation character for a Data Source Name (DSN) entry. Default: DSN = .
DSN Name	Specifies the name of the connection to the database to which Logger data will be exported. This name probably does not exist yet.
Master DSN Name	Specifies the connection to the master database of the SQL database server. The Master DSN probably already exists and you have to find out what its name is.
DB Keyword	The keyword and separation character to be used in the entries specifying the name of the database (to connect to) and the name of the master database (or database of databases). Default: DATABASE = .
DB Name	Specifies the name of the database to connect to. This database does not need to exist yet. Default: Use the name of the Logger recording as the database name.
	If you specify a database name, every recording you export will be exported to the same database, which will overwrite the previous data.
Master DB Name	Specifies the name of the master database. The name for the master database must match the actual master database on the server.
Server Keyword	Specifies the keyword and separation character for specifying a server. Default: SERVER = .

Table 6-1: SQL Send Plugin Control dialog box parameters (Sheet 2 of 2)

Parameter	Definition
Server Name	Specifies the network name of the computer on which the SQL database server is running. This can be an IP address, the “localhost” generic keyword, or a computer name.
User Keyword	Specifies the keyword and separation character for the user. Default: UID = .
User ID	Specifies the user name. If you have set up an account on the SQL database server you are trying to connect to, enter your user name. Otherwise try “root”.
Password Keyword	Specifies the keyword and separation character for the user password. Default: PASSWORD = .
Password	Specifies the password associated with the User ID. If you do not use a password, do not enter anything in this field.
Other Arguments	Allows you to specify other entries for the SQL query string. If the connection to your SQL database server requires additional data, enter it here (if specifying more than one entry, separate them with semi-colons).
User Parameters	User-added customizations. For example, if you use the Logger SQL API to build your own SQL plug-in on top of the default one and register it under the same name (“DtSqlSender”), you can command your own customized SQL plug-in through the standard GUI, and you can use this field to pass additional information to your plug-in.
Map Datum	Specifies a map datum. The Map datum is used in the conversions of geocentric coordinates and vectors to geodetic position (latitude, longitude, and altitude) and topographic vectors. The Map Datum can be specified either as five, colon-separated numbers, or as one of the keywords: WGS84, ED50, NASA-NAD27, NASB-NAD27, CONUS-NAD27. Default: WGS84.

6.2.3. Configuring and Enabling the sqlSender Plug-in in *logger.mtl*

You can configure and enable the sqlSender plug-in in the *logger.mtl* configuration file. You use the `enablePipelineStage` parameter and list of stages, as described in “[Loading Plug-Ins](#),” on page 8-23. The first and second entries of the list must be `sender` and `DtSqlSender`, which specify the stage type and the unique name under which the create function for the *DtSqlSender* stage is registered. The third entry is a composite string specifying all SQL database configuration parameters.

To specify the configuration parameters, use the same arguments as those used by the *sqlSender* application (described in [Table 6-2](#).) Wrap each parameter in square brackets (`[]`). [Example 6.1](#) illustrates configuration of the sqlSender plug-in. If you want to use a default value for a parameter, you do not have to include it in the list.

Example 6.1

```
1  (sequence
2    (setq pluginDir "../plugins/bin5")
3    (setq plugins (list "sqlSender5.dll"))
4    (setq enablePipelineStage
5      (list
6        (list "sender" "DtSqlSender"
7          "[-sqlDrv MySQL ODBC 3.51 Driver]
8          [-sqlDsnKwd DSN= ]
9          [-sqlDsnName loggerdb]
10         [-sqlDsnMName myDsnMName]
11         [-sqlDbKwd DATABASE=]
12         [-sqlDbName berkeleyDemoDIS]
13         [-sqlDbMName mysql]
14         [-sqlSrvKwd SERVER=]
15         [-sqlSrvName myMachine.myCompany.com]
16         [-sqlUsrKwd UID=]
17         [-sqlUsrName root]
18         [-sqlPassKwd PASSWORD=]
19         [-sqlPasswd xy34lfg9]
20         [-sqlArgs PORT=3306;DESCRIPTION=Logger Simulation Data]
21         [-mapDatum WGS84]"
22       )
23    )
24  )
25  )
```

If the value for a parameter includes square brackets, you must protect them with a backslash. For example, suppose that the description string is Exercise 12 [10/10/2004]. The composite string would be:

```
[-sqlArgs PORT=3306;DESCRIPTION=Exercise 12 \[10/10/2004\]]
```

6.2.4. Disabling the sqlSender Plug-in

If the sqlSender plug-in is enabled, you can disable it for recording, playback, or both. If you disable the plug-in for both record and playback, when you click OK, all configuration information is lost and you must enter it again before you can re-enable the plug-in.

To disable the sqlSender plug-in:

1. Choose Options → SQL Sender Control. The SQL Send Plugin Control dialog box opens ([Figure 6-1](#)).
2. Clear the Use During Recording box to disable the plug-in during record mode.
3. Clear the Use During Playback box to disable the plug-in during playback.
4. Click OK.

6.2.5. Troubleshooting Data Export

If you are having problems exporting data, to troubleshoot the Logger end, run the GUI and text interfaces together and set Logger notification to level 3. You can observe the messages issued by the Logger, which include data export messages. The startup options for this configuration would be:

```
Logger -g b -n 2:3
```

6.3. Using the *sqlSender* Application

The *sqlSender* application exports data from a Logger recording to a database. It is completely independent of the Logger application. However, it uses the same parameters as the Logger application does to communicate with the ODBC driver.

You must use the DIS version of the *sqlSender* application to export the data of a DIS recording and you must use the appropriate HLA version of the *sqlSender* application to export the data of an HLA recording. The HLA versions of the *sqlSender* application require an RTI to run.



The default values implemented in the SQL API are appropriate for use with a MySQL SQL server and the MySQL ODBC driver (ver. 3.51.07).

The *sqlSender* application takes a set of arguments to configure the connection to the SQL database server and a set of arguments to configure the network settings for the simulation standard you are using. [Table 6-2](#) lists the command-line options for the *sqlSender* application and the default value (if any) for each parameter. See [Table 6-1](#) for a more complete description of the meaning of the database-related parameters. See “[Configuring the Logger](#),” on page 2-19 for descriptions of the simulation standard parameters.

Table 6-2: *sqlSender* application parameters (Sheet 1 of 2)

Parameter	Description	Default Value
Database connection parameters		
-sqlDrv	ODBC driver name.	MySQL ODBC 3.51 Driver
-sqlDsnKwd	DSN definition keyword.	DSN =
-sqlDsnName	DSN (data source name).	
-sqlDsnMName	Master DSN (data source name).	mysql
-sqlDbKwd	Database definition keyword.	
-sqlDbName	Database name.	
-sqlDbMName	Master database name.	mysql
-sqlSrvKwd	Server definition keyword.	SERVER =

Table 6-2: sqlSender application parameters (Sheet 2 of 2)

Parameter	Description	Default Value
-sqlSrvName	Server name.	localhost
-sqlUsrKwd	User definition keyword.	UID =
-sqlUsrName	User name.	root
-sqlPassKwd	Password definition keyword.	PASSWORD =
-sqlPasswd	User password.	
-sqlArgs	Other SQL attributes.	DESCRIPTION = MAK Logger Data
-usrParams	User-defined extra parameters.	
-mapDatum	The Map Datum to use.	WGS84
Application (non-database connection) parameters		
-tape	Path to Logger recording to be exported.	
-L	Logger Entity ID string.	
HLA-only parameters		
-x	Execution-name.	
-N	Federate name.	
-F	FED-file name.	
-f	FOM-mapper-library:init-data, or v1RPRxx RPRFOM version.	
DIS-only parameters		
-a	Application number.	
-x	Exercise ID.	
-A	Destination address.	
-P	Port .	
-B	receiveBufferSize:sendBufferSize for the socket.	
-t	Multicast TTL value to use.	
-I {0,1}	Enable (1) or disable (0) asynchronous IO	
-S	Additional multicast address to use.	

6.4. Database Schema Used by the Logger SQL API

When activated, both the sqlSender plug-in and the *sqlSender* application attempt to create a database using the name of the Logger recording that is being played or recorded, or the name specified as input. The data exported to the database is structured as five independent tables, named entities, aggregates, aggregate_subordinates, detonations, and fires. This section describes the contents of these tables.

6.4.1. The entities Table

In HLA the entities table contains simulation data corresponding to object updates for objects that belong to RPR FOM 1.0 class Physical Entity (or to any of its sub-classes). In DIS this table contains simulation data corresponding to Entity State PDUs.

Table 6-3: Contents of entities table (Sheet 1 of 2)

Column	Name	Definition
1	rec_num	Record index
2	site	site:host:id (Entity ID) The Entity ID identifies the entity that is issuing the message.
3	host	
4	id	
5	force_id	The force to which the entity belongs.
6	type_kind	Entity type of the entity that is issuing the message.
7	type_domain	
8	type_country	
9	type_category	
10	type_subcategory	
11	type_specific	
12	type_extra	
13	position_x	Entity position. The position_ entries specify the Geocentric position (x, y, and z components, in meters). The next three entries represent the latitude and longitude (in degrees), and the altitude (in meters).
14	position_y	
15	position_z	
16	latitude	
17	longitude	
18	altitude	

Table 6-3: Contents of entities table (Sheet 2 of 2)

Column	Name	Definition
19	velocity_x	Velocity. The first three entries specify the x, y, z components of the velocity vector (in Geocentric reference frame, in m/s). The next three entries are the North, East, and down components of the velocity vector (in Topographic reference frame attached to the entity.)
20	velocity_y	
21	velocity_z	
22	velocity_north	
23	velocity_east	
24	velocity_down	Acceleration. The first three entries are the x, y, z components of the acceleration (in Geocentric reference frame, in meters per second squared). The next three entries are the North, East, and down components of the acceleration vector (in Topographic reference frame attached to the entity.)
25	acceleration_x	
26	acceleration_y	
27	acceleration_z	
28	acceleration_north	
29	acceleration_east	Orientation. The first three entries are the TaitBryan angles (in radians). The next three entries are the conventional Heading, Pitch, Roll (in degrees).
30	acceleration_down	
31	orient_psi	
32	orient_theta	
33	orient_phi	
34	heading_deg	Angular velocity (rate of rotation). The three entries are the x, y, z components of the vector in Body reference frame.
35	pitch_deg	
36	roll_deg	
37	ang_velocity_x	Angular velocity (rate of rotation). The three entries are the x, y, z components of the vector in Body reference frame.
38	ang_velocity_y	
39	ang_velocity_z	
40	timestmp	Arrival time of the message.
41	timestmp_type	Timestamp type.
42	appearance	OR-ed entity appearance enumerators.
43	dr_algorithm	Dead-reckoning algorithm enumerator.
44	marking	Marking text.

6.4.2. The aggregates Table

In HLA the aggregates table contains simulation data corresponding to object updates for objects that belong to RPR FOM 1.0 class “Aggregate Entity” (or to any of its subclasses). In DIS this table contains simulation data corresponding to Aggregate State PDUs.

Table 6-4: Contents of aggregates table (Sheet 1 of 2)

Column	Name	Definition
1	rec_num	Record index
2	site	site:host:id (Aggregate ID) The Aggregate ID identifies the aggregate that is issuing the message.
3	host	
4	id	
5	aggregate_state	Aggregate state enumerator.
6	formation	Formation enumerator.
7	dimension_x	Dimensions (in meters) of the aggregate's bounding volume.
8	dimension_y	
9	dimension_z	
10	sub_entity_count	The number of entities that publish.
11	sub_aggregate_count	Number of subaggregates that publish.
12	silent_entity_count	Number of entities that do not publish.
13	silent_aggregate_count	Number of subaggregates that do not publish.
14	force_id	The force to which the aggregate belong.
15	type_kind	Entity Type of the aggregate that is issuing the message.
16	type_domain	
17	type_country	
18	type_category	
19	type_subcategory	
20	type_specific	
21	type_extra	
22	position_x	Aggregate position. The position_ entries specify the Geocentric position (x, y, and z components, in meters). The next three entries represent the latitude and longitude (in degrees) and the altitude (in meters).
23	position_y	
24	position_z	
25	latitude	
26	longitude	
27	altitude	

Table 6-4: Contents of aggregates table (Sheet 2 of 2)

Column	Name	Definition
28	velocity_x	Velocity. The first three entries specify the x, y, and z components of the velocity vector (in Geocentric reference frame, in m/s). The next three entries are the North, East, and down components of the velocity vector (in Topographic reference frame attached to the entity.)
29	velocity_y	
30	velocity_z	
31	velocity_north	
32	velocity_east	
33	velocity_down	Acceleration. The first three entries are the x, y, z components of the acceleration (in Geocentric reference frame, in m/s ²). The next three entries are the North, East, and down components of the acceleration vector (in Topographic reference frame attached to the entity.)
34	acceleration_x	
35	acceleration_y	
36	acceleration_z	
37	acceleration_north	
38	acceleration_east	Orientation. The orient entries are the TaitBryan angles (in radians). The next three entries are the conventional Heading, Pitch, Roll (in degrees).
39	acceleration_down	
40	orient_psi	
41	orient_theta	
42	orient_phi	
43	heading_deg	Angular velocity (rate of rotation). The three entries are the x, y, z components of the vector in Bodyreference frame.
44	pitch_deg	
45	roll_deg	
46	ang_velocity_x	Angular velocity (rate of rotation). The three entries are the x, y, z components of the vector in Bodyreference frame.
47	ang_velocity_y	
48	ang_velocity_z	
49	timestmp	Arrival time of the message.
50	timestmp_type	Timestamp type.
51	appearance	OR-ed entity appearance enumerators.
52	dr_algorithm	Dead-reckoning algorithm enumerator.
53	marking	Marking text.

6.4.3. The aggregate_subordinates Table

The aggregate_subordinates table is an auxiliary table generated from some of the aggregate information that does not get listed in the aggregates table. Each aggregate object has a variable number of subordinates (entities or other aggregates), listed by their entity ID or aggregate ID. Due to the variability of the count these extra IDs are not included in the aggregates table, but are placed in this table instead. To associate the records in this table with records in the aggregates table you must match the Aggregate ID and the timestamp.

Table 6-5: Contents of aggregate_subordinates table

Column	Name	Definition
1	rec_num	Record index.
2	aggregate_site	site:host:id (Aggregate ID) The Aggregate ID identifies the aggregate that is issuing the message.
3	aggregate_host	
4	aggregate_id	
5	subordinate_site	site:host:id (Subordinate ID) The Subordinate ID identifies an entity or aggregate that belongs to the aggregate.
6	subordinate_host	
7	subordinate_id	
8	timestamp	Arrival time of the message.
9	timestamp_type	Timestamp type.

6.4.4. The detonations Table

In HLA the detonations table contains simulation data corresponding to the RPR FOM 1.0 MunitionDetonation interactions (or to any of the subclasses of Munition-Detonation). In DIS this table contains simulation data corresponding to Detonation PDUs.

Table 6-6: Contents of detonations table (Sheet 1 of 3)

Column	Name	Definition
1	rec_num	Record index.
2	attacker_site	site:host:id (Attacker Entity ID) The Attacker Entity ID identifies the initiating entity (the shooter).
3	attacker_host	
4	attacker_id	
5	target_site	site:host:id (Target Entity ID) The Target Entity ID identifies the targeted entity.
6	target_host	
7	target_id	

Table 6-6: Contents of detonations table (Sheet 2 of 3)

Column	Name	Definition
8	munition_site	site:host:id (Munition Entity ID) The Munition Entity ID identifies the munition that is being detonated.
9	munition_host	
10	munition_id	
11	event_site	site:host:id (Event ID) The Event ID identifies the launch of the munition (the originating Fire interaction.)
12	event_host	
13	event_id	
14	result	Detonation result enumerator.
15	world_location_x	Detonation location. The first three entries specify the Geocentric position (x, y, and z components, in meters). The next three entries represent the latitude and longitude (in degrees), and the altitude (in meters) of the detonation location.
16	world_location_y	
17	world_location_z	
18	world_location_latitude	
19	world_location_longitude	
20	world_location_altitude	The entity_location values represent the detonation location in the body reference frame of the target entity (in meters).
21	entity_location_x	
22	entity_location_y	
23	entity_location_z	Velocity. The first three entries specify the x, y, and z components of the velocity vector (in Geocentric reference frame, in meters per second). The next three entries are the North, East, and downcomponents of the acceleration vector (in Topographic reference frame at the location).
24	velocity_x	
25	velocity_y	
26	velocity_z	
27	velocity_north	
28	velocity_east	
29	velocity_down	Arrival time of the message.
30	timestmp	
31	timestmp_type	
32	munition_type_kind	Munition type.
33	munition_type_domain	
34	munition_type_country	
35	munition_type_category	
36	munition_type_subcategory	
37	munition_type_specific	
38	munition_type_extra	
39	warhead_type	Warhead type enumerator.
40	fuse_type	Fuse type enumerator.

Table 6-6: Contents of detonations table (Sheet 3 of 3)

Column	Name	Definition
41	quantity	Number of rounds fired in the burst.
42	rate	Rounds per minute.

6.4.5. The fires Table

In HLA the fires table contains simulation data corresponding to the RPR FOM 1.0 WeaponFire interactions (or to any of the subclasses of WeaponFire). In DIS this table contains simulation data corresponding to Fire PDUs.

Table 6-7: Contents of fires table (Sheet 1 of 2)

Column	Name	Definition
1	rec_num	Record index.
2	attacker_site	site:host:id (Attacker Entity ID) The Attacker Entity ID identifies the initiating entity (the shooter).
3	attacker_host	
4	attacker_id	
5	target_site	site:host:id (Target Entity ID) The Target Entity ID identifies the targeted entity.
6	target_host	
7	target_id	
8	munition_site	site:host:id (Munition Entity ID) The Munition Entity ID identifies the munition that is being detonated.
9	munition_host	
10	munition_id	
11	event_site	site:host:id (Event ID) The Event ID identifies the launch of the munition.
12	event_host	
13	event_id	
14	fire_mission	Fire mission index.
15	location_x	Launch location. The location_ entries specify the Geocentric position (x, y, and z components, in meters). The next three entries represent the latitude and longitude (in degrees), and the altitude (in meters) of the launch location.
16	location_y	
17	location_z	
18	latitude	
19	longitude	
20	altitude	

Table 6-7: Contents of fires table (Sheet 2 of 2)

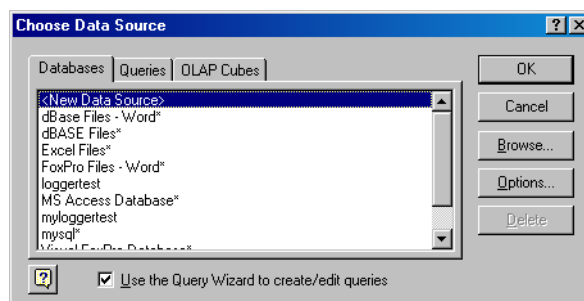
Column	Name	Definition
21	velocity_x	Velocity. The first three entries specify the x, y, and z components of the velocity vector (in Geocentric reference frame, in m/s). The next three entries are the North, East, and down components of the acceleration vector (in Topographic reference frame at the location).
22	velocity_y	
23	velocity_z	
24	velocity_north	
25	velocity_east	
26	velocity_down	
27	range	Assumed range (in meters).
28	timestmp	Arrival time of the message.
29	timestmp_type	Timestamp type.
30	munition_type_kind	Munition Type.
31	munition_type_domain	
32	munition_type_country	
33	munition_type_category	
34	munition_type_subcategory	
35	munition_type_specific	
36	munition_type_extra	
37	warhead_type	Warhead type enumerator.
38	fuse_type	Fuse type enumerator.
39	quantity	Number of rounds fired in the burst.
40	rate	Rounds per minute.

6.5. Importing Logger Data into A Spreadsheet

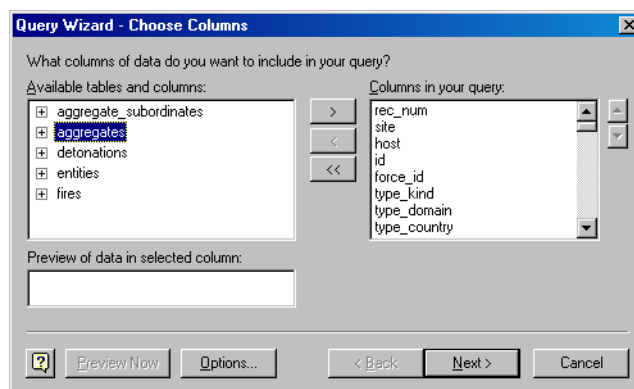
After you export Logger data to a database, you can examine and manipulate the data just as you would any database. One common way to view the data is to import it into a spreadsheet and use the spreadsheet's tools to analyze it, graph it, and so on. This section explains how to import data stored in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

To import data from a MySQL database to Microsoft Excel:

1. Start Excel.
2. Choose Data → Get External Data → New Database Query. The Choose Data Source dialog box opens.

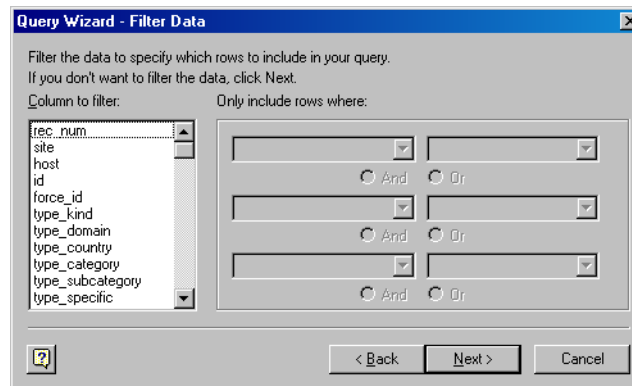


3. In the window, select the DSN Name that you configured for the sqlSender plug-in.
4. Click OK. The Query Wizard - Choose Columns dialog box opens. In the Available Tables and Columns window is a list of tables in the database. You can expand each table for a list of the columns in the table.

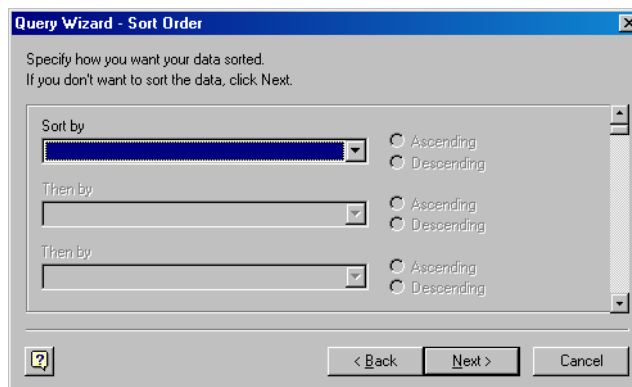


5. Select the tables and columns that you want to import to Excel.
6. Click the right-facing arrow to move the selected tables and columns to the Columns In Your Query window.

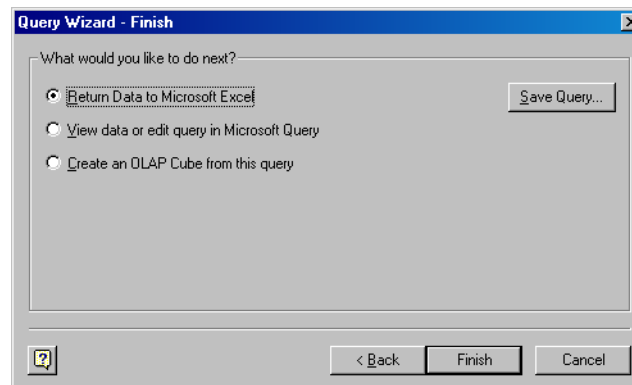
- Click Next. The Query Wizard - Filter Data dialog box opens.



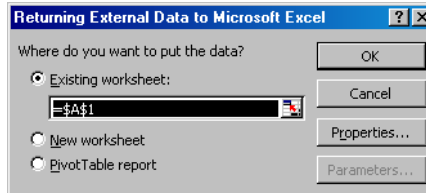
- Set up any filtering that you want, then click Next. The Query Wizard - Sort Order dialog box opens.



- Set up any sorting that you want, then click Next. The Query Wizard - Finish dialog box opens.



10. Select the Return Data to Microsoft Excel option and click Finish. The Returning Data to Microsoft Excel dialog box opens and a cell in the spreadsheet is highlighted.



11. Specify where you want the data to be imported to and click OK. The data is imported into the spreadsheet.

rec_num	site	host_id	force_id	type_kind	type_domain	type_country	type_category	type_subcategory	type_specific	type_extra	position_x
1	1	3006	8	2	1	3	222	6	0	0	3139014.166
2	2	1	3006	10	1	1	4	225	1	0	3139961.423
3	3	1	3006	10	1	1	4	225	1	0	3139961.423
4	4	1	3009	236	1	1	2	225	20	1	3139393.327
5	5	1	3009	237	1	1	2	225	20	1	3139473.84
6	6	1	3009	238	1	1	2	225	20	1	3139541.393
7	7	1	3009	239	1	1	2	225	20	1	3139445.836
8	8	1	3009	240	2	1	1	222	1	1	3139457.629
9	9	1	3009	242	2	1	1	222	1	1	3139521.736
10	10	1	3009	243	2	1	1	222	1	1	3139608.268
11	11	1	3009	244	1	1	1	225	1	3	3139697.775
12	12	1	3009	246	1	1	1	225	1	3	3139072.154
13	13	1	3009	247	1	1	1	225	1	3	3138825.173
14	14	1	3009	248	1	1	1	225	1	3	3139201.985
15	15	1	3009	254	1	1	3	225	11	0	3138906.4
16	16	1	3009	255	1	1	3	225	11	0	3139033.824
17	17	1	3009	256	1	1	3	225	11	0	3139128.459
18	18	1	3009	258	1	1	2	225	20	1	3138361.214
19	19	1	3009	259	1	1	2	225	20	1	3138467.752
20	20	1	3009	260	1	1	2	225	20	1	3138484.717



Remote Control

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

Remote Control Messages	7-2
Using Logger Commands for Remote Control	7-3
Capturing User Actions as Logger Control Messages	7-3
Deleting Logger Files	7-4
Getting the Logger's State	7-4
Example of Remote Control	7-5

7.1. 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*.

7.2. Using Logger Commands for Remote Control

Several startup options and their equivalent MTL parameters facilitate remote control of the Logger. [Table 7-1](#) lists the relevant startup options and MTL parameter.

Table 7-1: Logger commands for remote control

Startup Option	MTL Parameter	Description
-D {0 1}	allowRemoteFileDelete	Enables or disables file deletion in response to <i>DtLgrDelFilePdus</i> . 0 = Do not allow file deletions. 1 = Allow file deletions.
-m {0 1}	mirrorLc	0 = Do not capture user actions as Logger Control messages (default) 1 = Capture user actions as Logger Control messages.
-R {0 1}	recordLc	0 = Do not record Logger Control messages (default). 1 = Record Logger Control messages.
-L {0 1 2}	processLc	0 = Do not respond to Logger Control messages. 1 = Interpret Logger Control messages only if they are addressed to the entity ID specified by the -a argument (default). 2 = Interpret all Logger Control messages.

7.2.1. Capturing User Actions as Logger Control Messages

You can capture user actions through the GUI or text-based interface as Logger Control messages and add them to a recording.

- To capture user actions as Logger Control messages, start the Logger with the `-m 1` startup option or set the `mirrorLc` parameter to 1 in *logger.mtl*.

7.2.2. Deleting Logger Files

The Logger Control PDU, *DtLgrDelFilePdu* instructs the Logger to delete a Logger file. When the Logger receives this PDU, it decides whether or not to delete the file specified in the PDU based on the following criteria:

- ♦ If the MTL variable `allowRemoteFileDelete` is non-zero
- ♦ If the `-D` startup option is set to 1.

The default is to not allow deletion.

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 DthLA
    DtLgrControlInteraction lcInter;
    lcInter.setLcPdu(deleteFile);
    exerciseConn.sendStamped(lcInter);
#else
    exerciseConn.sendStamped(deleteFile);
#endif
```

7.2.3. Getting the Logger's State

The *DtLgrTotalStatePdu* reports the current state of the Logger. Although it could be issued by any application, the expectation is that the Logger will issue this PDU in response to a request from a remote application.

- To request the Logger's state, send a *DtLgrStateFlagPdu* with the `DtLGR_STATE_REQUEST` flag set.

The DtLgrStateFlagPdu

The *DtLgrStateFlagPdu* interaction can send a variety of On/Off commands to the Logger. It can send the following flags encoded as an enumerated type DtLgrCtrlFlag-States:

- ♦ DtLGR_TIME_CONSTR_ON – Toggle On time-constrained mode
- ♦ DtLGR_TIME_CONSTR_OFF – Toggle Off time-constrained mode
- ♦ DtLGR_TIME_REGLTD_ON – Toggle On time-regulating mode
- ♦ DtLGR_TIME_REGLTD_OFF – Toggle Off time-regulating mode
- ♦ DtLGR_OWN_FED_AMB_ON – Toggle On “Use Own Federate Ambassador”
- ♦ DtLGR_OWN_FED_AMB_OFF – Toggle Off “Use Own Federate Ambassador”
- ♦ DtLGR_STATE_REQUEST – Request full state update.

This PDU may have new flags added in the future. Therefore, code created using them may need to be recompiled for use with later versions of the Logger.

7.3. Example of Remote Control

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

```
#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");
```



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	8-2
The DtLogger Class	8-4
Including the Logger in An Application	8-4
Controlling the Logger Time Clock.....	8-6
Controlling Logger Files	8-8
Controlling the Logger's Configuration and Status	8-9
Controlling Logger Annotations.....	8-10
Setting Protocol-Specific Parameters.....	8-11
Managing Pipeline Objects.....	8-12
Using Large Files	8-12
The Logger Pipeline Architecture.....	8-13
Logger Pipeline Stages.....	8-14
Pipeline Stage Member Functions.....	8-14
Default Pipeline Stages	8-18
Creating Plug-ins	8-18
Initializing A Plug-in	8-19
Modifying the Pipeline Using A Plug-in	8-20
Overriding DtLogger from A Plug-in	8-22
Loading Plug-Ins	8-23
Managing Pipeline Stages.....	8-24
Adding Pipeline Stages	8-25
Removing Pipeline Stages	8-26
Compiling and Linking with the Logger API	8-26

8.1. 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 is composed of the Logger thread and the Logger Interfaces. The Logger thread is a single (main) thread in which the actual logging occurs. The Logger Interface components can run in separate threads (for example, the GUI interface of the Logger is a Logger Interface component, and runs in a separate thread) or in the main thread (for example, the Logger Text interface component runs in the main thread). The Logger Interface components provide you with custom interfaces to the Logger process, while the Logger thread component implements all basic operations of the Logger (playback, recording, data-processing, and so on). This high-level design allows the simultaneous use of multiple Logger Interface components commanding a single Logger thread. (However, a single Logger Interface component commanding multiple Logger threads is not possible.)

The requirement for multiple Logger Interface components having concurrent access to a single Logger thread necessitates the use of a mutex-protected data object shared between the interfaces and the Logger thread. Therefore the Logger architecture includes a shared data object (implemented via the class *DtLgrSharedData* (*lgrSharedData.h*)) and a mutex object (implemented via the class *DtLgrMutex* (*lgrMutex.h*)). The shared data object passes status information about the current state of the Logger thread to the interfaces, and implements a command-stack. A Logger Interface component commands the Logger thread by generating commands (instances of the class *DtLgrCmdDescriptor* (*lgrCmdDesc.h*)) and placing them on the command-stack of the shared data object. Each tick, the Logger thread executes the commands on the command stack. The Logger thread is not event-driven, instead, it advances in response to tick() calls. A host application that uses a Logger thread object must tick it.

For backwards compatibility, the Logger thread is enveloped in the class *DtLogger* (*logger.h*). The *DtLogger* class is in many respects very similar to a Logger Interface component. In particular, any given command to the Logger thread can be issued by calling a corresponding member function of the *DtLogger* class. Just as in the case of a command issued by a Logger Interface component the actual action is not executed until the Logger thread is ticked. You tick the Logger thread by calling the tick() member function of the *DtLogger* class. Unlike a Logger Interface component, *DtLogger* is in the Logger thread. Since the Logger architecture allows only one Logger thread, there must be only one instance of the *DtLogger* class per process.

The Logger Interface components are implemented through the class *DtLgrInterface* (*lgrInterface.h*) and its derived classes. For example:

- ♦ The text interface of the Logger application is implemented by the *DtTextInterface* class, derived from *DtLgrInterface*.
- ♦ The GUI interface is implemented by the *DtQtGuiInterface* class (*qtGuiInterface.h*), derived from the *DtLgrGuiInterface* class (*lgrGuiInterface.h*), which is derived from *DtLgrInterface*.
- ♦ The remote control interface is implemented by the *DtLgrCtrlListener* class (*lgrCtrlListener.h*), derived from *DtLgrInterface*.

You can use any (or all) of these interfaces in your application by creating an instance of the corresponding class. Alternatively, you can create a customized interface by deriving your own class from one of the standard interface classes. The areas of functionality that are common to all Logger interfaces such as commanding of the Logger thread, communicating with the mutex and the shared data object, and the management and storage of the local copy of the Logger thread status information are implemented in a separate class – the *DtLgrCommander* (*lgrCommander.h*). Each instance of a *DtLgrInterface* has its own *DtLgrCommander* object. The commander object is instantiated prior to the creation of the interface object, and then passed to it as an argument to its constructor. When it is destroyed, *DtLgrInterface* deletes the commander object. The *DtLgrCommander* class has a public interface that is almost identical to that of *DtLogger* – each public member function that corresponds to issuing a Logger command or querying for a status parameter has an identical name and prototype in *DtLogger* and *DtLgrCommander*.

If your goal in using the Logger API is to add standard Logger features to your application, the *DtLogger* and *DtLgrInterface* classes may be the only portion of the API that you need to understand. (In fact, it is possible to use the *DtLogger* class alone, without any corresponding Logger Interface components. However, you would need to provide an alternative means of controlling the Logger and you will have to separately implement thread synchronization, or build a single-threaded application.)

If you need to extend the Logger’s functionality by, for example, adding new file formats, adding new data sources and destinations, or extending the processing of data inside the Logger thread, then you will need to understand the Logger thread internal architecture, which is described in [“The Logger Pipeline Architecture,”](#) on page 8-13.

8.2. The DtLogger Class

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.



You can also create the *DtLogger* object in your plug-in code. In this case the standard Logger application uses the *DtLogger* instance created by your plug-in.

The Logger thread only executes commands during the call to the `tick()` member function of *DtLogger*. This means that when you make calls to the `play()`, `record()`, `stop()`, `pause()`, and other member functions of the *DtLogger* class, all that happens internally is that commands are being placed on the command-stack, to be executed during the next call to `tick()`.

8.2.1. Including the Logger in An Application

The Logger includes two examples of using *DtLogger* in a custom application. The examples are `./examples/loggerAPI/playFile.cxx` and `./examples/loggerAPI/writeFile.cxx`.

The `playFile` example opens a file for playback; `writeFile` opens a file for recording. Both examples demonstrate the use of the *DtLogger* object with and without a corresponding Logger Interface component. When the constant `DT_USE_LGRGUI` is defined, the example application uses Logger's standard GUI interface. When `DT_USE_LGRGUI` is not defined, the example application uses the *DtLogger* object only and provides user control through a very simple text-based interface.

The first part of the code (in each example) is devoted to processing the command-line arguments and is self-explanatory.

The next part of the example creates the *DtLogger* instance by passing to the constructor the arguments needed for creating an exercise connection. The exercise connection is the application's interface to the network regardless of whether you are using HLA or DIS. You can indicate that you want the Logger to use an existing exercise connection by passing a pointer to this connection to *DtLogger*'s constructor. In fact, if you are creating a *DtLogger* from within a VR-Link application, we recommend that you pass your application's normal exercise connection to the *DtLogger* in this fashion.

To prepare for playback, the timescale is set to 1.0 and a sample Logger recording is loaded. The timescale affects the speed at which the recording is read. This example is simple; it plays the recording back in real time.

By default, the Logger is in a stopped state after opening a file – unless the default play option has been set using `DtLogger::setPlayAction()`. To enter play mode, call `DtLogger::play()`. This member function initializes the pipeline engine for playing back the file; however, no data is read from the file and sent until `DtLogger::tick()` is called.

If the constant `DT_USE_LGRGUI` is defined, the next part of the example instantiates the Logger GUI interface. Each Logger Interface component requires an individual instance of a *DtLgrCommander* class. *DtLgrCommander* implements the issuance of standard Logger commands, access to the shared data object, management and storage of the local copy of the Logger thread status data, and so on. Therefore, the constructor of a *DtLgrInterface* class takes as input a pointer to its *DtLgrCommander* object. In turn, the instance of the *DtLgrCommander* object is constructed indirectly through a call to the class factory (this allows you to replace the default implementation of the *DtLgrCommander* with a class derived from *DtLgrCommander*), and subsequently initialized. The `initSelf()` member function takes a pointer to a data structure that carries information about the current *DtLogger* object, the shared data object, the mutex object, the current thread ID, and a pointer to any extra user-data (to be used by custom implementations of classes derived from *DtLgrCommander*). Finally, the created and initialized instance of *DtLgrCommander* is passed to the constructor of *DtQtGuiInterface*.

The rest of the main function consists of the main execution loop and the exit cleanup. On each loop cycle we must call `DtLogger::tick()` to give it a chance to continue the execution of its current action, and grab and execute any new commands that are on the command-stack).

If the `DT_USE_LGRGUI` constant is not defined we look for user keyboard input by calling the `keybrdTick()` function. If the `DT_USE_LGRGUI` constant is defined we also tick the GUI to give it time to update itself, process user actions, and translate them into issuance of Logger commands.

8.2.2. Controlling the Logger Time Clock

The Logger thread maintains an internal clock object (an instance of *DtLgrPbClock* (*lgrClock.h*)), which it uses for all time-related activities (for example, measuring tape-time, real-time, federation-time, and so on.). This clock object is wrapped around the *DtClock* object of the exercise connection.



DtLogger maintains the simulation-time of the exercise connection, and is affected (usually adversely) by external modification of the *DtClock* object directly through the exercise connection.

The clock object is updated automatically during the call to *tick()* based on the current mode (recording or playback), time-scale, user-commands, and so on. The time displayed on the Logger GUI (reported by the *time()* member functions of *DtLogger*, *DtLgrCommander*, *DtLgrBaseState*, and *DtLgrState*) refers to tape-time. During recording you cannot modify the state of this internal clock, but during playback you can control the speed and direction of the clock advance, and set or get the current tape-time. [Table 8-1](#) lists the *DtLogger* and *DtLgrCommander* member functions for controlling the clock.



Query member functions return status information as it was cached at the end of the last tick. Command member functions place commands on the command-stack (to be executed during the next tick). This note applies to all of the set/get functions described in the tables in this chapter.

[Table 8-1](#) lists the *DtLogger* member functions for controlling the time clock.

Table 8-1: Time clock functions (Sheet 1 of 2)

Function	Description
<i>idealTime()</i>	Returns the intended scaled tape-time. The <i>idealTime()</i> is the value that <i>time()</i> will always return if the Logger can keep up playing at the current scale. When <i>time()</i> falls significantly behind <i>idealTime()</i> it is time to reduce the time-scale. Meaningful only during playback. Available only in <i>DtLogger</i> .
<i>timeScale()</i>	Gets the timescale.

Table 8-1: Time clock functions (Sheet 2 of 2)

Function	Description
setTimeScale()	Sets the timescale. The timescale affects the speed at which a recording is read. ♦ A negative timescale causes the recording to be read backwards. ♦ A timescale between 1.0 and -1.0, excluding 0.0, plays the recording more slowly than real time. ♦ A timescale greater than 1.0 or less than -1.0 plays the recording faster than real time. ♦ A timescale equal to zero suspends the Logger clock, resulting in a paused state.
time()	Gets the Logger clock's time. The value returned by time() always corresponds to the current time in a loaded Logger recording.
setTime()	Sets the internal Logger clock's time. setTime() is only valid in playback mode, and only accepts values that fall between the beginning and end of a recording. Calling this function jumps playback to the specified time.
skipTime()	Specifies a time interval to add to the current value of time(). skipTime(5.0) and setTime(5.0 + time()) are equivalent. Has same restrictions as setTime().
HLA Time Managed Mode Only	
timelsManaged()	Reports whether or not the Logger can operate in time-managed mode.  To use time management in the Logger, you must configure your RTI for Time Management services.
setTimelsManaged()	Enables the Logger to operate in time-managed mode.
timelsRegulated()	Reports whether or not the Logger is in time-regulating playback mode.
setTimelsRegulated()	Puts the Logger in time-regulating playback mode.
timelsConstrained()	Reports whether or not the Logger is in time-constrained recording mode.
setTimelsConstrained()	Puts the Logger in time-constrained recording mode.
dfltLookahead()	Gets the lookahead value used in time-regulating playback.
setDfltLookahead()	Sets the lookahead used in time-regulating playback.
federateTime()	Gets the current federate time.

8.2.3. Controlling Logger Files

DtLogger and *DtLgrCommander* have member functions that provide high-level information about the current file (its name, length, and so on). *DtLogger::file()* provides access to a *LtPlaybackFile* or *LtRecordFile* object. Chapter 9, [Using the Logger File API](#) describes how you can use these objects to iterate or search through the packets in a file, and modify or examine the contents of a file. [Table 8-2](#) describes *DtLogger* and *DtLgrCommander* member functions that control Logger files.

Table 8-2: Logger file control functions

Function	Description
<code>filePosition()</code>	Returns a <i>DtLgrFilePosition</i> (defined in VR-Link's <i>lcPdu.h</i> file) enumeration value describing the current file position. Valid values are <code>DtLGR_NOFILE</code> , <code>DtLGR_START</code> , <code>DtLGR_MIDDLE</code> , <code>DtLGR_END</code> .
<code>file()</code>	Returns a pointer to either a <i>LtPlaybackFile</i> or a <i>LtRecordFile</i> . (<i>DtLogger</i> only.)
<code>gotoStart()</code> <code>gotoEnd()</code>	Moves to the beginning or end of a file. Valid in playback mode only.
<code>loadFileForRecording()</code>	Opens a file for recording and prepares the Logger pipeline for record mode.
<code>loadFileForPlayback()</code>	Opens a file for playback and prepares the Logger pipeline for playback mode.
<code>pause()</code>	Pauses playback of a recording.
<code>play()</code>	Causes the Logger to play a recording.
<code>record()</code>	Causes the Logger to start recording an exercise.
<code>setPlaybackFileCreator()</code> <code>setRecordFileCreator()</code>	Allows you to set a creator function for an object derived from either <i>LtPlaybackFile</i> or <i>LtRecordFile</i> . <i>DtFileSource</i> and <i>DtSendToFile</i> use these functions in <code>initSelf()</code> to create their file objects. (<i>DtLogger</i> only.)
<code>stop()</code>	Causes the Logger to stop playing or recording.
<code>tapeLength()</code>	Returns the time-length (duration) of the recording in seconds. If no recording is loaded this returns -1.0. Meaningful only during playback.
<code>tapeFileName()</code>	Returns the name of the current recording. If no recording is loaded, returns <code>NULL</code> .

8.2.4. Controlling the Logger's Configuration and Status

Table 8-3 describes *DtLogger* and *DtLgrCommander* member functions for setting Logger default behaviors and for obtaining Logger state information.

Table 8-3: Configuration and status functions (Sheet 1 of 2)

Function	Description
eofAction() setEofAction()	Gets or sets the action the Logger takes when it reaches the end of a recording. Valid settings are: ♦ Stop at the end of the recording (which deletes all entities). ♦ Pause at the end of a recording (which does not delete any entities). ♦ Wrap to the beginning or ending of the recording, depending on the direction it is being played.
inPlayback()	Returns whether the Logger is set to play a Logger recording or record a simulation. This value changes when <code>loadFileForPlayback()</code> or <code>loadFileForRecording()</code> is called.
mode()	Returns the current mode. This value is set internally when you open files for playback or for recording and when you call functions such as <code>play()</code> , <code>stop()</code> , <code>pause()</code> , and <code>record()</code> . This function returns a <code>DtLgrMode</code> enumeration (which is defined in VR-Link in <i>lcPdu.h</i>) value that can be either <code>DtLGR_PLAY</code> , <code>DtLGR_PAUSE</code> , <code>DtLGR_STOP</code> , or <code>DtLGR_EMPTY</code> .
playAction() setPlayAction()	Gets or sets the Logger state when it opens a new file for playback. Valid settings are stop mode and play mode.
recordAction() setRecordAction()	Gets or sets the Logger action when it opens a new file for recording. Valid settings are stop mode and record mode.
recordingLoggerControlInfo() setRecordingLoggerControlInfo()	Gets or sets whether to record Logger control messages.
mirroring()	Gets the status of mirroring.
setMirroring()	Specifies whether or not the Logger sends Logger control messages with certain commands. For example, when mirroring is on and <code>play()</code> is called, a <code>DtLgrPlay</code> message is sent.

Table 8-3: Configuration and status functions (Sheet 2 of 2)

Function	Description
setDefaultRecordSource-Parameter()	Specifies a parameter to be sent to the default recording source. For example, if you want to override a default recording source (for example, <i>DtDisSource</i>) so that it uses the <i>DtString</i> parameter in its constructor, you set the value to pass here. The next time <i>loadFileForRecording()</i> is called, the value you have set is passed to your class's constructor. The default value is "None".
packetLimit()	Gets the packet limit. (For <i>DtLogger</i> only.)
setPacketLimit()	Allows you to configure how many packets the Logger can generate during one pass through its internal processing pipeline. The default limit is 1000 packets and should only need to be modified in rare cases.

8.2.5. Controlling Logger Annotations

An individual Logger annotation (represented by a *DtNote* object (*note.h*)) consists of a unique ID, a time point, and text. The *DtAnnotationManager* class (*annMgr.h*) maintains a list of annotations. You can access a pointer to the annotation manager with the *annotations()* member function of *DtLogger* or *DtLgrCommander*. When you add an annotation to the list (by calling the *add()* member function of *DtAnnotationManager*), the annotation manager creates a *DtNote* object and returns a pointer to the new *DtNote* object (on failure *add()* returns NULL). The *DtAnnotationManager* class also contains member functions for querying for a specific annotation, storing the annotations to a file, loading new annotations from file, looping through the annotations in the list, and so on. For example if you want to print out the contents of *montyDemo.ant*, you could do the following:

```
DtAnnotationManager* mgr = myLogger->annotations();
mgr->loadFromFile( "../logger_tapes/montyDemo.ant" );
for (DtNote* n = mgr->first(); n; n = mgr->next())
{
    printf("%f: %s\n", n->time(), n->msg().string());
}
```

If you want to jump to the first annotation in the list, you could do the following:

```
DtNote* note = mgr->first();
myLogger->setTime(note->time());
```

To remove an annotation from the list, call *DtAnnotationManager::remove()*. To remove all the stages in the list, call *DtAnnotationManager::empty()*.

8.2.6. Setting Protocol-Specific Parameters

The API lets you change protocol-specific parameters so that you can customize the Logger to your simulation and improve its performance. You have access to the Logger's exercise connection, so you can use network and protocol options that are available in VR-Link, in your Logger applications.

DIS only

For DIS, you can customize the heartbeat and timeout intervals.

HLA only

In HLA, you can specify the FOM classes to which the Logger should subscribe. Also, you can optimize the way interactions are recorded if you use the Logger's federate ambassador.

Table 8-4 describes *DtLogger* and *DtLgrCommander* member functions for getting and setting protocol-specific parameters.

Table 8-4: Protocol-specific functions

Function	Description
exerciseConn()	Returns a pointer to the Logger's exercise connection. (<i>DtLogger</i> only.)
DIS	
heartbeat() setHeartbeat()	Set or get the heartbeat interval.
timeout() setTimeout()	Set or get the timeout interval.
HLA	
passiveSubscription() setPassiveSubscription()	Set or get the status of the flag specifying whether the Logger should subscribe actively or passively to HLA updates during recording.
usingLoggerFederateAmbassador() setUsingLoggerFederateAmbassador()	Set or get the status of the flag specifying whether the Logger should use its own implementation of a federate ambassador (instead of the default one provided in VR-Link). The set() member function is a static member function of the <i>DtLogger</i> class, and should be called prior to the instantiation of <i>DtLogger</i> . It has no effect after the construction, and it also has no effect if the Logger was passed an externally-created exercise connection. (<i>DtLogger</i> only.)
subscriptions() setSubscriptions()	Specifies a list of object and interaction classes to which the Logger should subscribe (during recording).

8.2.7. Managing Pipeline Objects

Table 8-5 describes *DtLogger* and *DtLgrCommander* member functions for registering, adding, and removing pipeline stage objects from the Logger pipeline. For a more detailed discussion of the pipeline and pipeline stages, see “[The Logger Pipeline Architecture](#),” on page 8-13.

Table 8-5: Pipeline object functions

Function	Description
<code>addObject()</code>	Creates, initializes, and adds a pipeline stage object using the <code>creator()</code> member function that is registered under the specified name. The new object must be specified as being one of the four basic types (source, action, filter, or sender). If there is already a stage object of the same stage type and instantiated by the <code>create()</code> function listed under the same registered name, it is deleted and reinstantiated (with the new input parameter and flags).
<code>registerObject()</code>	Registers a pipeline stage object creator. If there was a <code>create()</code> member function already registered under this name, any stage objects that were created with it are deleted and reinstantiated with the new <code>creator()</code> member function and the original input parameters and flags.
<code>removeObject()</code>	Removes an existing pipeline stage object.

8.2.8. Using Large Files

If you want the Logger to use 64-bit file IO, which supports files larger than 2 GB, call the `setUseLargeFiles()` member function of *DtLogger* or *DtLgrCommander*. To find out the type of file IO that the Logger is currently using, use the `useLargeFiles()` member function.

8.3. The Logger Pipeline Architecture

This section describes how the Logger generates and processes packets. This information is useful if you want to change the core functionality of the Logger (how it generates packets to or from a file, or how it reads packets from the network) or want to add to the core functionality (for example, filter certain packet types or direct recorded traffic to two separate files.)

The *DtLogger*'s *DtLgrPipeline* object (defined in *lgrPipeline.h*) organizes the different stages of Logger packet processing. The Logger has four distinct processing steps:

- ♦ It generates packets
- ♦ Optionally, it inspects the generated packets
- ♦ Optionally, it filters the generated packets
- ♦ It sends the packets.

Figure 8-1 illustrates the basic stages of the Logger's core engine or packet processing pipeline. They are sources, actions, filters, and senders. Sources generate Logger packets from messages received from a file or the network. The packets are passed to actions, filtered if appropriate, and exit the pipeline through the senders.

The pipeline makes no distinction between playback and record modes. It just has different sources and different outputs in each mode. For example, when the Logger records a DIS exercise, it chooses a DIS connection as its source stage and a file as its sender stage. The Logger can filter out packets so that they do not get sent to the recorded file or the network. You can add actions to monitor the packet stream and compile statistics.

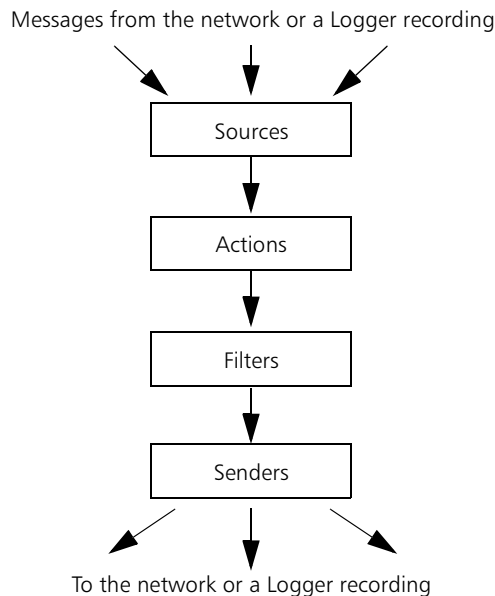


Figure 8-1. Logger pipeline

8.4. Logger Pipeline Stages

The Logger pipeline has four types of stages:

- ♦ Sources
- ♦ Actions
- ♦ Filters
- ♦ Senders.

Each of the stages is derived from a pipeline stage class called *DtLoggerStage(logger-Stage.h)*. All objects derived from *DtLoggerStage* contain abstract functions to initialize, activate, deactivate, and verify themselves. Source objects can generate Logger packets, which package the messages from the external source. Filters, actions, and senders can process a list of packets generated by a source object.

All *DtLoggerStage* classes have a *tick()* member function for doing extra processing. All *DtLoggerStage* objects are given a *DtLgrState* object (defined in *lgrState.h*) to initialize themselves. *DtLoggerState* is a repository for Logger state data. This data includes the Logger's current play state, clock time, and exercise connection.

The pipeline stage objects are implemented by the following classes:

- ♦ Source objects are implemented by *DtSourceObj* classes (defined in *sourceObj.h*).
- ♦ Filter objects are implemented by *DtFilterObj* classes (defined in *filterObj.h*).
- ♦ Action objects are implemented by *DtActionObj* classes (defined in *actionObj.h*).
- ♦ Sender objects are implemented by *DtSendObj* classes (defined in *sendObj.h*).

The implementation for all these classes is in the *libLgrCore* library.

8.4.1. Pipeline Stage Member Functions

Table 8-6 describes the member functions for the pipeline stage classes. Most of the member functions are common to the four types of stages.



If you want to derive new stage classes, you must override some of the functions, as indicated.

Table 8-6: Member functions for pipeline stages (Sheet 1 of 3)

Function	Description
Functions that apply to all stages:	
<i>deactivate()</i>	Indicates that the object should no longer produce data. If it currently has stored data, it can delete the data. The pipeline no longer calls <i>process()</i> to receive data. This is a pure virtual function and you must override it if you want to derive a new type of stage.

Table 8-6: Member functions for pipeline stages (Sheet 2 of 3)

Function	Description
getName()	Returns the attribute <code>myName</code> . The <code>myName</code> attribute is a <i>DtString</i> that holds the name of the instance of the class. The name does not have to be unique; however, it is often useful for debugging purposes to give it a meaningful name. This name is typically passed in or created during object construction.
initSelf()	The <code>initSelf()</code> member function is the first function called after construction of an object. The object is passed the <i>DtLoggerState</i> and is expected to perform initialization routines. <i>DtLoggerState</i> is a pointer to a repository of Logger state data such as the Logger's current play state, clock, and exercise connection. If the object needs the <i>DtLoggerState</i> pointer later, it should save it here. This is a pure virtual function and you must override it if you want to derive a new type of stage.
isOk()	Returns the attribute <code>myStatus</code> . The <code>myStatus</code> attribute is a boolean that is used to indicate that there is a problem with this object and the object should be removed. The status of a pipeline stage is checked whenever the stage has been invoked by the pipeline object. If it returns <code>false</code> , the pipeline removes the object and destroys it.
printInfo()	Returns identifying information about the object. Depending on the type of object, it returns either "A Source", "A Filter", "An Action", or "A Sender" by default.
reset()	Returns a stage object to its initial state. For example a file object should return to the beginning of the file. This is a pure virtual function and you must override it if you want to derive a new source stage class.
sendPacket()	The <code>sendPacket()</code> member function is called by <code>processData()</code> to act on each individual packet. This is a pure virtual function and you must override it if you want to derive a new source stage class.
tick()	Yields execution time to the object. This function is virtual, but not abstract. The default version of this function does nothing.
Functions that apply only to action, filter, and sender stages:	
activate()	The <code>activate()</code> member function is called to indicate that the object will begin to receive data in its <code>processdData()</code> member function. This is a pure virtual function and you must override it if you want to derive a new source stage class.

Table 8-6: Member functions for pipeline stages (Sheet 3 of 3)

Function	Description
processData()	The pipeline sends the list of packets to the processData() member function so the object can perform appropriate actions. Action and Sender objects should only parse the list and read the data. Filter objects can alter the list, but if a filter removes a packet, it must also delete it. By default: - Action objects call actOnPacket() for each packet in the list - Filter objects call filterPacket() on every packet in the list - Sender objects call sendPacket() on every packet in the list.
Functions that apply only to source stages:	
activate()	When the pipeline wants to begin receiving data from a source, it calls the activate() member function. The parameter startTime holds the time from which the source should begin sending data. If it has earlier data it should ignore it. The return value should indicate to the pipeline what time this object will be ready to send its first packet. If unknown, the object should return -1. The pipeline begins calling processes to receive the data. This is a pure virtual function and you must override it if you want to derive a new source stage class.
generatePackets()	The generatePackets() member function is used by the pipeline to grab waiting data from the object. The parameter currentTime informs the object of the current simulated time. The object should create LtPacketInfo objects and add them to the list. The responsibility for deleting the LtPacketInfo objects belongs to the pipeline. The return value is the next time the object will have data to send. If unknown, the object should return -1. The default action of this function is to call generatePacket() until it returns a NULL.
generatePacket()	The generatePacket() member function creates one packet at a time and is called by the generatePackets() member function. The currentTime parameter is the simulation time that packets should be generated for. The nextTime parameter is a return value for when the next packet will be ready. This is a pure virtual function and you must override it if you want to derive a new source stage class.
Functions that apply only to action stages:	
actOnPacket()	The actOnPacket() member function is where the action performs any packet-specific operation.
Functions that apply only to filter stages:	
filterPacket()	The filterPacket() member function is where the filter can make a decision about whether a packet should pass its filter or not. This function is allowed to change the packet. A return value of true indicates to the processData() member function that the packet should be removed from the list.

The Logger calls `initSelf()` immediately after a stage is added to the pipeline. In the case of source and sender objects, which are continually being added and removed from the pipeline as a result of calls to `loadFileForPlayback()` and `loadFileForRecording()`, this call occurs often. Filters and actions are reset whenever a new object is loaded for playback or recording. Whenever `play()` or `record()` are called, the Logger calls `activate()` on each stage to indicate that all stages should prepare to either generate or process packets. Conversely, whenever `stop()` is called, the Logger calls `deactivate()` on each stage to indicate that all stages should stop producing data.

Packets are processed as follows:

1. The `tick()` function is called on every stage object.
2. The pipeline checks the status of each pipeline stage using the function `isOk()`. If an error has occurred and `isOk()` returns `false`, the pipeline removes the offending object and destroys it.
3. The pipeline calls `generatePackets()` for each source stage.
4. The pipeline checks the status of each pipeline stage using the function `isOk()`. If an error has occurred and `isOk()` returns `false`, the pipeline removes the offending object and destroys it.
5. The pipeline calls `processData()` for each action.
6. The pipeline checks the status of each pipeline stage using the function `isOk()`. If an error has occurred and `isOk()` returns `false`, the pipeline removes the offending object and destroys it.
7. The pipeline calls `processData()` on each filter stage.
8. The pipeline checks the status of each pipeline stage using the function `isOk()`. If an error has occurred and `isOk()` returns `false`, the pipeline removes the offending object and destroys it.
9. The pipeline calls `processData()` for each sender stage.
10. The pipeline checks the status of each pipeline stage using the function `isOk()`. If an error has occurred and `isOk()` returns `false`, the pipeline removes the offending object and destroys it.

8.4.2. Default Pipeline Stages

The Logger has several *DtLoggerStage* objects, which implement basic Logger functionality such as the playing and recording of files using DIS, HLA, or a TCP/IP connection. Each of the default pipeline stage classes is registered under a unique name. [Table 8-7](#) lists the each default pipeline stage class, its reserved name, and its description.

Table 8-7: Pipeline stage objects and reserved names

Class	Name	Description
<i>DtFileSource</i>	FileSource	Reads packets from a file.
<i>DtHlaActiveSource</i>	HlaActiveSource	Obtains packets from an active subscription to an HLA exercise.
<i>DtHlaPassiveSource</i>	HlaPassiveSource	Obtains packets from a passive subscription to an HLA exercise.
<i>DtHlaSimulate</i>	HlaSimulate	Sends packets to an HLA exercise.
<i>DtDisSource</i>	DisSource	Reads packets from a DIS exercise.
<i>DtDisSimulate</i>	DisSimulate	Sends packets to a DIS exercise.
<i>DtSendToFile</i>	SendToFile	Sends packets to a file.
<i>DumpObj</i>	ScreenDump	Writes packets, in human readable form, to the screen or to a file.
<i>DtTcpSource</i>	TCPSource	Reads packets from a TCP port.
<i>TcpSendObj</i>	TcpPort	Sends packets to a TCP connection.

You can override any of these stages by deriving your own classes from them and registering the `create()` function for the custom class under the unique name of the default stage. For example, suppose you derived a file source object called *MyFileSource*. You would replace *DtFileSource* with your own class as follows:

```
myLogger->registerObject("FileSource", MyFileSource::create);
```

8.5. Creating Plug-ins

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. When you use a plug-in, all the basic functionality of the Logger 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 Pipeline Architecture,” on page 8-13 explains the procedure for adding new stages to the pipeline. First, a creator function for the stage must be registered with a call to the `registerObject()` member function for *DtLogger* or *DtLgrCommander*. Then, an instance of the stage can be created, initialized, and added to the pipeline with a call to the `addObject()` member function for *DtLogger* or *DtLgrCommander*.

The first part of this section describes how to register and add new stages to the Logger's pipeline using a plug-in. The second part of this section describes how to override the application's default *DtLogger* instance with your own derived class.

8.5.1. Initializing 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. When you want to add new pipeline stages or override existing ones you must export a function called `InitPlugin()`. If you want to override *DtLogger*, you must export `CreateLogger()`. You can export either or both, but the functions must look like the following:

```
#include "initMethods.h"

extern "C"
{
    LGR_EXPORT DtLogger* CreateLogger(void* data)
    {
        // User Logger creation belongs here.
    }

    LGR_EXPORT void InitPlugin(void* data)
    {
        // User register code belongs here.
    }
}
```

The Logger application calls each of these functions during initialization. The data parameter is a pointer to the Logger application object, *DtLoggerApp*. If the Logger loads a module that exports `CreateLogger()`, it uses the function to create its *DtLogger* instance. If more than one module contains this function, the Logger only uses the first one. After a *DtLogger* instance is created, the Logger calls every module's `InitPlugin()` function. You can access start-up options and command line arguments through the *DtLoggerApp* object. Most importantly, the *DtLoggerApp* provides access to its *DtLogger* object.

8.5.2. Modifying the Pipeline Using A Plug-in

When the application calls `InitPlugin()`, the *DtLogger* instance has been created and can be used to register and add pipeline stages. For example, if you wanted to add the *ExampleFilter* object to the pipeline, the contents of `InitPlugin()` could look as follows:

```
LGR_EXPORT void InitPlugin(void* data)
{
    DtLogger* logger = ((DtLoggerApp*) data)->logger();
    logger->registerObject("ExampleFilter", ExampleFilter::create);

    // Add some instances of the filter to the pipeline here, too.
    logger->addObject(DtFilter, "ExampleFilter", "1:2:225:2:1:0:0");
    // filter US planes

    logger->addObject(DtFilter, "ExampleFilter", "1:2:222:2:1:0:0");
    // filter Soviet planes
}
```

This sample code registers and adds *ExampleFilter* to the pipeline. You also could have simply registered the creator, then later, have added the filters using an `EnableObjects()` function, or the `enablePipelineStage` MTL parameter.

It is very easy to replace one of the default Logger pipeline stages using a plug-in's `InitPlugin()` function. For example, to override *DtFileSource* with your own class called *MyFileSource*, you simply need to do the following:

```
LGR_EXPORT void InitPlugin(void* data)
{
    ((DtLoggerApp*) data)->registerObject("DtFileSource",
        MyFileSource::create);
}
```

A pipeline stage class that has been registered with the Logger can be instantiated and added to the Logger pipeline in the following ways:

- The objects can be created in code with the `EnableObjects()` function in the plug-in DLL
- The objects can be specified by the `enablePipelineStage` parameter in the *logger.mtl* file.
- Objects can be added using the Logger Pipeline Stages Manager in the Logger GUI. See [“Managing Pipeline Stages,”](#) on page 8-24 for details.

The `EnableObjects()` function is similar to `InitPlugin()`; however, it is not mandatory. The function must be a “C” style function of the following form:

```
LGR_EXPORT DtList* EnableObjects()
```

The function returns a *DtList* containing arrays of three-character strings, as follows:

- The first string describes the stage type of the object to be created. The valid values for this string are “source”, “action”, “filter”, and “sender”.
- The second string is the name of the object to be created. This value should match the name the object was registered with.
- The third string is the parameter that will be passed to the object's constructor.

The following code is an example of an `EnableObjects()` member function using `ExampleFilter`:

```
extern "C"
{
    LGR_EXPORT DtList* EnableObjects( )
    {
        // Always build this object.
        static char enableTriple1[][30] = { "filter",
                                             "ExampleFilter",
                                             "1:2:225:2:1:0:0" };

        static char enableTriple2[][30] = { "filter",
                                             "ExampleFilter",
                                             "1:2:222:2:1:0:0" };

        static DtList aList;
        aList.add(enableTriple1);
        aList.add(enableTriple2);
        return &aList;
    }
}
```

A pipeline stage can be enabled by adding the `enablePipelineStage` MTL parameter to *logger.mtl*. This parameter is a list that contains the same three string sets that are used in the `EnableObjects()` member function:

```
(setq enablePipelineStage
  (list
    (list "stage_type_keyword" "registered_name" "input_parameter")
  )
)
```

The MTL command that would create the object for the `ExamplePlugin` is:

```
(setq enablePipelineStage
  (list
    (list "filter" "ExampleFilter" "1:2:225:2:1:0:0")
    (list "filter" "ExampleFilter" "1:1:222:2:1:0:0")
  )
)
```

The MTL lines are equivalent to the `EnableObjects()` function in the previous example.

i

EnableObject settings in the *logger.mtl* file override objects enabled when a plug-in is loaded. When you call `addObject()` with a stage type and registered name that match an existing object, the existing object gets removed and a new object is instantiated with the new parameter value.

8.5.3. Overriding *DtLogger* from A Plug-in

To create your own Logger class, create a class that inherits from *DtLogger*. You now have access to all the functions defined in *logger.h*. It is important for the overridden class to call the original *DtLogger* functions to preserve the original Logger functionality. For example:

```
void MyLogger::play()
{
    // Perform my own action on play.
    doSomething();

    // Call the original Logger play() member function.
    DtLogger::play();
}

void MyLogger::stop()
{
    // Perform my stop call.
    stopSomething();

    // Call the original stop() member function.
    DtLogger::stop();
}
```

To tell the application to use your new *DtLogger* class, define *CreateLogger()* in your plug-in and have it return a pointer to your new class.

```
extern "C"
{
    LGR_EXPORT DtLogger* CreateLogger(void* data)
    {
        DtLoggerApp* app = (DtLoggerApp*) data;
        return new MyLogger(app->port(), 1, 12, 3);
    }
}
```

The application pointer contains member functions for obtaining configuration data such as the port number, federation name, and RPR FOM version. This allows you to use command line arguments to initialize your Logger object. Also, you can add your own command line arguments and interpret them here. The previous example gets the port number for the application and uses it to instantiate our Logger.

8.5.4. Loading Plug-Ins

By default, the DIS Logger looks for plug-ins in the `../plugins/bin5` directory, the HLA 1.3 Logger looks for plug-ins in the `../plugins/binHLA13` directory, and the HLA 1516 Logger looks for plug-ins in the `../plugins/bin1516` directory. A file is recognized as a plug-in library if it has the appropriate extension (`.dll` or `.so`), can be loaded as a DLL or shared library, and exports one (or both) of the functions `InitPlugin()` or `CreateLogger()`.

You can override the default process for loading plug-ins by specifying parameters in the `logger.mtl` configuration file. The MTL settings always override the corresponding default settings. This file defines the following plug-in-related variables:

- ♦ **pluginDir** – Specifies (as a relative path) the directory in which the plug-in modules are located. When the variable is not defined, the Logger uses the default locations.
- ♦ **plugins** – Specifies the plug-ins that should be loaded by the Logger. If the variable is not defined, the Logger tests all files in the `./plug-ins` directory that have the expected extension. If the variable is defined, the Logger tries to load only the files in its list. If the list is empty, it does not load any plug-ins.
- ♦ **enablePipelineStage** – Specifies a list of pipeline stage objects that are to be created and added to the pipeline immediately. (The outer list represents the different stages to be enabled. Each of the inner lists must consist of three elements specifying the stage-type keyword, the unique name under which the `create()` member function (to be used for the instantiation of the plugin) has been registered, and the input parameter (to be passed as an argument to the `initSelf()` member function of a successfully instantiated stage object

The following is an example of the plug-in section of `logger.mtl` for DIS:

```
;; Setup for DIS plugins
(if (equal DIS 1)
  (sequence
    (setq pluginDir "../plugins/bin5/")
    (setq plugins (list "sqlSendPlugin5.dll"))
    (setq enablePipelineStage
      (list
        (list "sender" "DtSqlSender"
              "[-sqlSrvName localhost]
              [-sqlDsnName mysql]
              [-mapDatum WGS84]" )
        )
      )
  )
)
```

8.6. Managing Pipeline Stages

Once a `create()` function is registered for a pipeline stage, a stage object can be added to and removed from the pipeline programmatically using the *DtLogger* or *DtLgrCommander* `addObject()` and `removeObject()` member functions. When you load a plug-in at startup and enable a pipeline stage with the `enablePipelineStage` parameter, the Logger uses this mechanism. The Logger Pipeline Stages Manager allows you to dynamically add and remove pipeline stages from the Logger GUI.

The Logger Pipeline Stages Manager has tabs for each of the four stage types. On each tab, it displays the pipeline stages of that type that are enabled, the mode for which they are enabled, and their initialization parameters. (Figure 8-2) shows two sources enabled. Both are enabled for playback. They are not enabled for record. The FileSource stage shows its status in gray, because it is a default Logger stage that cannot be changed by users.



You can only enable one instance of a registered name at a time. If you want to use more than one instance of a type of stage, associate that stage's `create()` function with multiple registered names. For example, suppose that you have defined a custom filter stage *DtMySpecialFilter*, and you want to use more than one instance of that stage. Register the create function for the *DtMySpecialFilter* class under the registered names `SpecialFilter00`, `SpecialFilter01`, `SpecialFilter02`, and so on. Then add and remove the same pipeline stage class using the different names you have defined.

8.6.1. Adding Pipeline Stages

If you register a `create()` function for a pipeline stage when you start the Logger, you can add an instance of that stage or replace the current instance. When you add a pipeline stage of a type that is already enabled, the Logger removes the current instance and adds a new instance with the new parameters you specify.

To add a pipeline stage:

1. Choose Options → Pipeline Stages Manager. The Logger Pipeline Stages Manager dialog box opens (Figure 8-2).

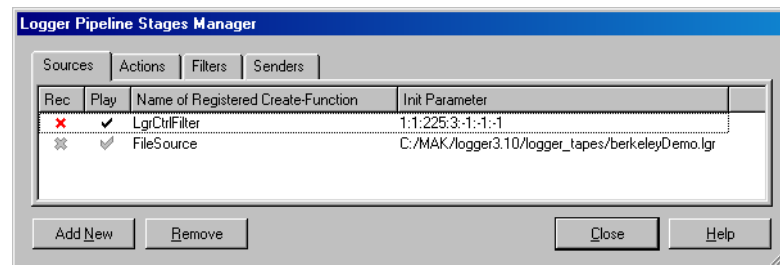


Figure 8-2. Logger Pipeline Stages Manager

2. Click Add New. The New Pipeline Stage Object dialog box opens.

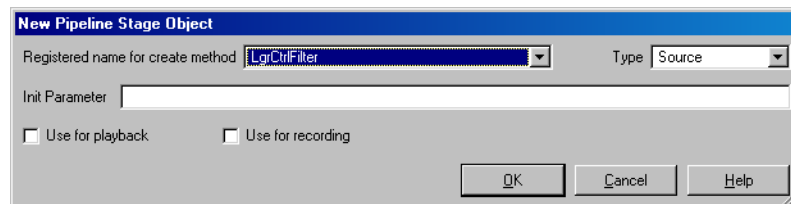


Figure 8-3. New Pipeline Stage Object dialog box

3. In the Registered Name For Create Method list, select the registered name for the pipeline stage that you want to instantiate.
4. In the Type list, select the stage type.
5. In the Init Parameter text box, enter the appropriate text string for the stage. (You need to know the initialization requirements for your code. If there are multiple arguments, separate them with square brackets ([]), as you would in the `enablePipelineStage` parameter in *logger.mtl*.)
6. Check Use For Playback, Use For Recording, or both, to specify which mode the pipeline stage should affect.
7. Click OK.

8.6.2. Removing Pipeline Stages

When you remove a pipeline stage, you disable its functionality. It remains registered with the Logger and you can add it back at a later time.



You cannot remove pipeline stages that the Logger requires for basic functioning. The record and play status of protected stages is displayed in gray, rather than black or red.

To remove a pipeline stage:

1. Choose Options → Pipeline Stages Manager. The Logger Pipeline Stages Manager dialog box opens (Figure 8-2).
2. Select the tab on which the pipeline stage is displayed.
3. Select the pipeline stage in the window.
4. Click Remove.

8.7. 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.

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
```

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.

HLA only

For HLA 1.3, link against

```
-LloggerDir/lib -lLoggerHLA13 -lLgrCoreHLA13
-lLgrFileHLA13 -L$(MAK_VRLDIR)/lib -lv1HLA13 -lmatrix
-lmtl -lvlutil -L$(MAK_RTIDIR)/lib -lRTI-NG -lfedtime
```

For HLA 1516, link against:

```
-LloggerDir/lib -lLoggerHLA1516 -lLgrCoreHLA1516
-lLgrFileHLA1516 -L$(MAK_VRLDIR)/lib -lv1HLA1516
-lmatrix -lmtl -lvlutil -L$(MAK_RTIDIR)/lib -lrti1516
-lfedtime1516
```

DIS only

For DIS, link against:

```
-LloggerDir/lib -lLogger5 -lLgrCore5 -lLgrFile5
-L$(MAK_VRLDIR)/lib -lv15 -lmatrix -lmtl -lvlutil
```

Table 8-8: Static libraries

DIS	HLA 1.3	HLA 1516
<i>libLoggerApp5.a</i>	<i>libLoggerAppHLA13.a</i>	<i>libLoggerAppHLA1516.a</i>
<i>libLoggerGui5.a</i>	<i>libLoggerGuiHLA13.a</i>	<i>libLoggerGuiHLA1516.a</i>
<i>libLogger5.a</i>	<i>libLoggerHLA13.a</i>	<i>libLoggerHLA1516.a</i>
<i>libLgrCore5.a</i>	<i>libLgrCoreHLA13.a</i>	<i>libLgrCoreHLA1516.a</i>
<i>libLgrFile5.a</i>	<i>libLgrFileHLA13.a</i>	<i>libLgrFileHLA1516.a</i>

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.

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 five Logger DLLs (*libLoggerApp_xxx*, *libLoggerGui_xxx*, *libLogger_xxx*, *libLgrCore_xxx*, and *libLgrFile_xxx*). Consult [Table 8-9](#) and [Table 8-10](#) for the exact name (based on protocol and build- flavor) of a specific DLL or its corresponding export library.

Table 8-9: DLLs

DIS	HLA 1.3	HLA 1516
<i>libLoggerApp5RT.dll</i>	<i>libLoggerAppHLA13RT.dll</i>	<i>libLoggerAppHLA1516RT.dll</i>
<i>libLoggerApp5RTd.dll</i>	<i>libLoggerAppHLA13RTd.dll</i>	<i>libLoggerAppHLA1516RTd.dll</i>
<i>libLoggerGui5RT.dll</i>	<i>libLoggerGuiHLA13RT.dll</i>	<i>libLoggerGuiHLA1516RT.dll</i>
<i>libLoggerGui5RTd.dll</i>	<i>libLoggerGuiHLA13RTd.dll</i>	<i>libLoggerGuiHLA1516RTd.dll</i>
<i>libLogger5RT.dll</i>	<i>libLoggerHLA13RT.dll</i>	<i>libLoggerHLA1516RT.dll</i>
<i>libLogger5RTd.dll</i>	<i>libLoggerHLA13RTd.dll</i>	<i>libLoggerHLA1516RTd.dll</i>
<i>libLgrCore5RT.dll</i>	<i>libLgrCoreHLA13RT.dll</i>	<i>libLgrCoreHLA1516RT.dll</i>
<i>libLgrCore5RTd.dll</i>	<i>libLgrCoreHLA13RTd.dll</i>	<i>libLgrCoreHLA1516RTd.dll</i>
<i>libLgrFile5RT.dll</i>	<i>libLgrFileHLA13RT.dll</i>	<i>libLgrFileHLA1516RT.dll</i>
<i>libLgrFile5RTd.dll</i>	<i>libLgrFileHLA13RTd.dll</i>	<i>libLgrFileHLA1516RTd.dll</i>

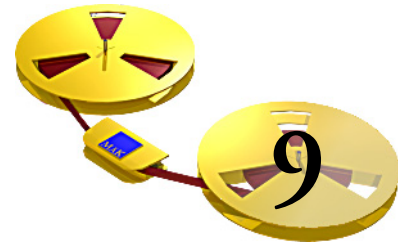
Table 8-10: Export libraries

DIS	HLA 1.3	HLA 1516
<i>libLoggerApp5RT.lib</i>	<i>libLoggerAppHLA13RT.lib</i>	<i>libLoggerAppHLA1516RT.lib</i>
<i>libLoggerApp5RTd.lib</i>	<i>libLoggerAppHLA13RTd.lib</i>	<i>libLoggerAppHLA1516RTd.lib</i>
<i>libLoggerGui5RT.lib</i>	<i>libLoggerGuiHLA13RT.lib</i>	<i>libLoggerGuiHLA1516RT.lib</i>
<i>libLoggerGui5RTd.lib</i>	<i>libLoggerGuiHLA13RTd.lib</i>	<i>libLoggerGuiHLA1516RTd.lib</i>
<i>libLogger5RT.lib</i>	<i>libLoggerHLA13RT.lib</i>	<i>libLoggerHLA1516RT.lib</i>
<i>libLogger5RTd.lib</i>	<i>libLoggerHLA13RTd.lib</i>	<i>libLoggerHLA1516RTd.lib</i>
<i>libLgrCore5RT.lib</i>	<i>libLgrCoreHLA13RT.lib</i>	<i>libLgrCoreHLA1516RT.lib</i>
<i>libLgrCore5RTd.lib</i>	<i>libLgrCoreHLA13RTd.lib</i>	<i>libLgrCoreHLA1516RTd.lib</i>
<i>libLgrFile5RT.lib</i>	<i>libLgrFileHLA13RT.lib</i>	<i>libLgrFileHLA1516RT.lib</i>
<i>libLgrFile5RTd.lib</i>	<i>libLgrFileHLA13RTd.lib</i>	<i>libLgrFileHLA1516RTd.lib</i>

Table 8-11: Link requirements for HLA and DIS

Application	Libraries
For HLA 1516, Release configuration, link against:	
Logger:	<i>libLoggerHLA1516RT.lib libLgrCoreHLA1516RT.lib libLgrFileHLA1516RT.lib</i>
VR-Link:	<i>vHLA1516RT.lib matrixRT.lib mtlRT.lib vlutilRT.lib</i>
RTI 1516:	<i>librti1516.lib libfedtime1516.lib</i>
For HLA 1516, Debug configuration, link against:	
Logger:	<i>libLoggerHLA1516RTd.lib libLgrCoreHLA1516RTd.lib libLgrFileHLA1516RTd.lib</i>
VR-Link:	<i>vHLA1516RTd.lib matrixRTd.lib mtlRTd.lib vlutilRTd.lib</i>
RTI 1516:	<i>librti1516d.lib libfedtime1516d.lib</i>
For HLA 13, Release configuration, link against:	
Logger:	<i>libLoggerHLA13RT.lib libLgrCoreHLA13RT.lib libLgrFileHLA13RT.lib</i>
VR-Link:	<i>vHLA13RT.lib matrixRT.lib mtlRT.lib vlutilRT.lib</i>
RTI 13:	<i>libRTI-NG.lib libFedtime.lib</i>
For HLA 13, Debug configuration, link against:	
Logger:	<i>libLoggerHLA13RTd.lib libLgrCoreHLA13RTd.lib libLgrFileHLA13RTd.lib</i>
VR-Link:	<i>vHLA13RTd.lib matrixRTd.lib mtlRTd.lib vlutilRTd.lib</i>
RTI 13:	<i>libRTI-NGd.lib libFedtimed.lib</i>
For DIS, Release configuration, link against:	
Logger:	<i>libLogger5RT.lib libLgrCore5RT.lib libLgrFile5RT.lib</i>
VR-Link:	<i>vI5RT.lib matrixRT.lib mtlRT.lib vlutilRT.lib</i>
For DIS, Debug configuration, link against:	
Logger:	<i>libLogger5RTd.lib libLgrCore5RTd.lib libLgrFile5RTd.lib</i>
VR-Link:	<i>vI5RTd.lib matrixRTd.lib mtlRTd.lib vlutilRTd.lib</i>

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



Using the Logger File API

In addition to the Logger API (see Chapter 8, *Using The Logger API*), the Logger supports a Logger File API. The Logger File API allows you to read and write Logger recordings from applications.

Using the Logger File API	9-2
Reading A Logger File	9-2
Creating An LtLoggerPlaybackFile	9-3
Reading Data	9-4
Writing A Logger File	9-5
Creating An LtLoggerRecordFile	9-5
Writing Data	9-6
Interpreting LtPacketInfo	9-6
Using Large Files	9-9
Editing Logger Files	9-9
Compiling and Linking with the Logger Library	9-10

9.1. Using the Logger File API

In addition to the standalone tools that help you view or modify MÄK Data Logger files, we provide an API (application programmer's interface) to the Logger file format, and a library that implements the API. The API allows you to read and write MÄK Logger files from your own programs.



You must have a VR-Link developer's license to develop applications that use the Logger file library.

The Logger file API is in *./include/loggerFile.h*. In UNIX, the libraries are in the *lib* directory:

- *libLgrFileHLA13.a* – supports HLA 1.3
- *libLgrFileHLA11516.a* – supports HLA 1516
- *libLgrFile5.a* – supports DIS.

For PCs, the following libraries are provided:

- *libLgrFile5RT.lib* – DIS release export library
- *libLgrFile5RTd.lib* – DIS debug export library
- *libLgrFile5RT.dll* – DIS release DLL
- *libLgrFile5RTd.dll* – DIS debug DLL
- *libLgrFileHLA13RT.lib*, *libLgrFileHLA1516RT.lib* – HLA release export libraries
- *libLgrFileHLA13RTd.lib*, *libLgrFileHLA1516RTd.lib* – HLA debug export libraries
- *libLgrFileHLA13RT.dll*, *libLgrFileHLA1516RT.dll* – HLA release DLLs
- *libLgrFileHLA13RTd.dll*, *libLgrFileHLA1516RTd.dll* – HLA debug DLLs

See “Compiling VR-Link Applications” in *VR-Link Developer's Guide* for information about the different libraries.

The Logger file API has two main classes: *LtLoggerRecordFile* (*loggerRecFile.h*) and *LtLoggerPlaybackFile* (*loggerPbFile.h*). Use *LtLoggerRecordFile* when you want to write a Logger file, and *LtLoggerPlaybackFile* when you want to read a Logger file. Both of these classes are derived from *LtLoggerFile* (*loggerFile.h*).

9.1.1. Reading A Logger File

To read a Logger file, create an instance of *LtLoggerPlaybackFile*, and use its member functions to navigate through a file and read packets. The term packet refers to a packet of data in the Logger file: either a DIS PDU, or an HLA update, interaction, or object removal. All Logger packets are represented by instances of the *LtPacketInfo* class (*packetInfo.h*).

9.1.2. Creating An *LtLoggerPlaybackFile*

To read data from standard input, rather than from an actual file, pass the string “stdin” as the filename argument to the *LtLoggerPlaybackFile* constructor.



LtLoggerPlaybackFile has a non-default constructor, which accepts arguments, for backwards compatibility. However, MÄK strongly recommends that you always use the default constructor and then call the appropriate `init()` function to initialize the file object and provide the appropriate initialization arguments.

HLA only

In HLA, a Logger file contains information about the FOM being used in the recorded exercise, because during playback, the Logger may have to convert class, attribute, and parameter handles stored in the recording to their corresponding handles in the current federation execution (there is no guarantee that the same classes, attributes, and parameters are assigned the same handles in different federation executions, especially if the FED file has changed).

Internally, the Logger playback file object handles the conversion process by maintaining two FOM descriptor objects: the FOM descriptor as read from the recording, and the FOM descriptor for the current federation execution (the target FOM). The former is obtained directly from the recording, while the latter is specified in the `init()` call by:

1. Passing a pointer to the exercise connection. The *LtLoggerPlaybackFile* object uses this pointer to query the exercise connection about the current FOM and construct a corresponding FOM descriptor object. If you pass NULL as the pointer to the exercise connection, you are instructing the *LtLoggerPlaybackFile* object to use the FOM descriptor from the recording as the target FOM, and to not convert any handles.
2. Passing a pointer to an already existing FOM descriptor object. The *LtLoggerPlaybackFile* creates the target FOM FOM descriptor object by cloning the specified FOM descriptor object. If you pass NULL as the pointer to the FOM descriptor, you are instructing the *LtLoggerPlaybackFile* object to use the FOM descriptor from the recording as the target FOM, and to not convert any handles.

9.1.3. Reading Data

When you first create an *LtLoggerPlaybackFile*, you are at the beginning of the file. The functions described in [Table 9-1](#) either read and return a packet, or change the location of a file pointer that determines where you are in the file:

Table 9-1: *LtLoggerPlaybackFile* functions for reading or returning packets

Function	Description
<code>readPacket()</code>	Reads and returns the next packet, NULL if none. In HLA, packets might be skipped if the file was created with a non-NULL exercise connectio, and the packet cannot be converted to the exercise connections's FOM.
<code>readPacketBackwards()</code>	Reads and returns the previous packet, NULL if none. In HLA, packets might be skipped if the file was created with a non-NULL exercise connection and the packet cannot be converted to the exercise connection's FOM.
<code>skipPacketForwards()</code> <code>skipPacketBackwards()</code>	Skip over the next packet either forwards or backwards. Return 1 if successful, 0 if there are no packets in that direction.
<code>jumpToStart()</code> <code>jumpToEnd()</code>	Go to the beginning or end of the file.
<code>seekTime()</code>	Jump to time t (in seconds) in the file.
<code>seekForward()</code> <code>seekBackward()</code>	Jump forward or backwards in the file by t seconds.

The functions described in [Table 9-2](#) provide information about the file and about your current location in the file:

Table 9-2: *LtLoggerPlaybackFile* functions for reporting time and location

Function	Description
<code>atStart()</code> <code>atEnd()</code>	Returns 1 if at you are at the beginning or end of the file, otherwise returns 0.
<code>fileTimeBackwards()</code> <code>fileTimeForwards()</code>	Fills t with the time of the next or previous packet. Returns 1 if successful, 0 if there is no packet in that direction.
<code>firstPacketTime()</code> <code>lastPacketTime()</code>	Fills t with the time of the first or last packet in the file.

The functions `readPacket()` and `readPacketBackwards()` return their data in the form of a pointer to an instance of *LtPacketInfo*. See “[Interpreting LtPacketInfo](#),” on page 9-6 for information about how to interpret this object.



The *LtPacketInfo* instance returned is dynamically allocated by the callee. It is the responsibility of the caller to delete it.

9.1.4. Writing A Logger File

To write a Logger file, create an instance of *LtLoggerRecordFile*, and use its `writePacket()` member function to write each DIS PDU or HLA message.

9.1.5. Creating An *LtLoggerRecordFile*

To write data to standard output, rather than to a file, pass the string `stdout` as the filename argument to the *LtLoggerRecordFile* constructor.

Notes:

- This option is not available if you are using files that are larger than 2 GB.
- *LtLoggerRecordFile* has a non-default constructor, which accepts arguments, for backwards compatibility. However, MÄK strongly recommends that you always use the default constructor and then call the appropriate `init()` function to initialize the file object and provide the appropriate initialization arguments.

HLA only

In HLA, a Logger file contains information about the FOM being used in the recorded exercise, because during playback, the Logger may have to convert class, attribute, and parameter handles stored in the recording to their corresponding handles in the current federation execution (there is no guarantee that the same classes, attributes, and parameters are assigned the same handles in different federation executions, especially if the FED file has changed).

The Logger record file object must record a FOM descriptor for the federation. You must specify the FOM in the `init()` call by:

1. Passing a pointer to the exercise connection. The *LtLoggerRecordFile* object uses this pointer to query the exercise connection about the current FOM and construct a corresponding FOM descriptor object.
2. Passing a pointer to an already existing FOM descriptor object. The *LtLoggerRecordFile* creates the target FOM FOM descriptor object by cloning the specified FOM descriptor object.

9.1.6. Writing Data

LtLoggerRecordFile's `writePacket()` member function looks like this:

```
virtual int writePacket(LtPacketInfo *packetInfo);
```

Its argument is a pointer to an instance of *LtPacketInfo*. See “[Interpreting LtPacketInfo](#),” on page 9-6 for information about filling out this object with the data you would like written.

9.1.7. Interpreting *LtPacketInfo*

This section describes aspects of the *LtPacketInfo* class.

The `arrivalTime()` member function returns the time associated with this packet. It is not the timestamp of the HLA or DIS message. It is a time offset into the file, written by the Logger or other application writing the file. If the file was written by the MÄK Logger, `arrivalTime()` is the time, since the Logger started recording, that the packet was received ([Figure 9-1](#)).

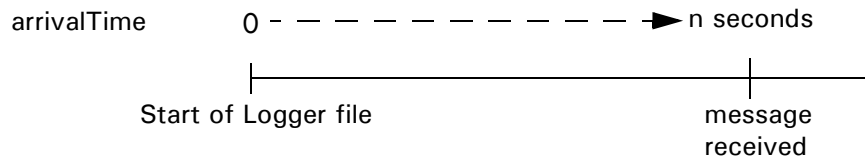


Figure 9-1. `arrivalTime`

The `readPacket()` and `writePacket()` member functions represent data serialization methods.

The class includes the member function `packetKind()` that returns an *LtPacketKind*. This value indicates the type of data the packet represents. Its values are as follows:

```
enum LtPacketKind
{
    LtPacketInteraction = 0,
    LtPacketUpdate,
    LtPacketRemoval,
    LtPacketAnnotation,
    LtPacketPdu,
    LtPacketFom
    LtPacketAudio
    LtPacketNone
};
```

DIS only

DIS packets all use the value *LtPacketPdu*. The DIS PDU data can be retrieved using the `packet()` member function.

The size of the packet in bytes is found with the `size()` member function.

HLA only

The interpretation of the fields in *LtPacketInfo* depends on the packet kind, as follows:

- ♦ If the packet kind is *LtPacketInteraction*, then `classHandle()` returns the interaction handle associated with the interaction class in the FOM. The `myHandleValueSet` field (accessible through `handleValueSet()`) points to an instance of `LtParameterHandleValueSet` object that contains the actual interaction data (the handle-value pair set). In files recorded by the Logger, this is the representation of the RTI object (of type `RTI::ParameterHandleValuePairSet` when using HLA 1.3, and of type `rti1516::ParameterHandleValueMap` when using HLA 1516) that was passed to the Logger by the RTI through the `receiveInteraction` service. To obtain a corresponding RTI object from `myHandleValueSet`, or to set the `myHandleValueSet` object from an RTI object, call `params()` or `setParams()` respectively.

The HLA 1.3 and HLA 1516 versions of the `params()` member function have a subtle difference:

- The the HLA 1.3 version expects a reference to a `RTI::ParameterHandleValuePairSet` pointer that is initialized to `NULL`. (If the pointer is not initially set to `NULL`, the Logger will attempt to delete the object it initially points to). If successful, the HLA 1.3 `params()` member function allocates an instance of a `RTI::ParameterHandleValuePairSet` object, and sets its contents in accordance with the current contents of the `myHandleValueSet` object and the handle-mapping specified by the FOM descriptor object (specified as input), and returns a pointer to the `RTI::ParameterHandleValuePairSet` object via the input reference. Therefore, in HLA 1.3 it is the caller's responsibility to delete the returned `RTI::ParameterHandleValuePairSet` object.
- The HLA 1516 version expects a reference to a (preallocated) instance of `rti1516::ParameterHandleValueMap` object, which upon success is set in accordance with the current contents of the `myHandleValueSet` object and the handle-mapping specified by the FOM descriptor object (specified as input). In this case it is the caller that manages the instance of the `rti1516::ParameterHandleValueMap` object all along. When the packet kind is *LtPacketInteraction*, the return values of the `objectId()`, `name()`, and `nameSize()`, calls are meaningless.

- ♦ If the packet kind is *LtPacketUpdate*, then `classHandle()` returns the object handle associated with the interaction class (in the FOM). The `objectId()` member function returns the unique integer ID associated with the corresponding HLA object (within the context of the particular tape). The `name()` member function returns the unique name associated with the corresponding HLA object within the context of the particular tape. (Although on playback the Logger will attempt to publish the corresponding object with the same name as listed in the tape, there is no guarantee that the RTI will allow that; if the name in question is already in use, the RTI forces the Logger to create the object without a name, and to accept whatever name the RTI assigns for it.) The `myHandleValueSet` field (accessible through `handleValueSet()`) points to an instance of *LtAttributeHandleValueSet* object that contains the actual update data (the handle-value pair set). In files recorded by the Logger, this is the representation of the RTI object (of type `RTI::AttributeHandleValuePairSet` when using HLA 1.3, and of type `rti1516::AttributeHandleValueMap` when using HLA 1516) that was passed to the Logger by the RTI through the `reflectAttributeValues` service. To obtain a corresponding RTI object from `myHandleValueSet`, or to set the `myHandleValueSet` object from an RTI object, call `attrs()` or `setAttrs()` respectively. The HLA 1.3 and HLA 1516 versions of the `attrs()` call have a subtle difference:
 - The HLA 1.3 version expects a reference to a `RTI::AttributeHandleValuePairSet` pointer that is initialized to NULL. (If the pointer is not initially set to NULL, the Logger will attempt to delete the object it initially points to). If successful, the HLA 1.3 `attrs()` allocates an instance of a `RTI::AttributeHandleValuePairSet` object, and sets its contents in accordance with the current contents of the `myHandleValueSet` object and the handle-mapping specified by the FOM descriptor object (specified as input), and returns a pointer to the `RTI::AttributeHandleValuePairSet` object via the input reference. Therefore, in HLA 1.3 it is the callers responsibility to delete the returned `RTI::AttributeHandleValuePairSet` object once it has outlived its usefulness.
 - The HLA 1516 version expects a reference to a (preallocated) instance of `rti1516::AttributeHandleValueMap` object, which upon success is set in accordance with the current contents of the `myHandleValueSet` object and the handle-mapping specified by the FOM descriptor object (specified as input). The caller that manages the instance of the `rti1516::AttributeHandleValueMap` object all along.
- ♦ If the packet kind is *LtPacketRemoval*, then the packet represents the removal of a persistent HLA object from the exercise. In Logger files this type of packet is added in response to a `removeObjectInstance` RTI service invocation. Here the calls `classHandle()`, `objectId()`, `name()`, and `nameSize()` are all valid, and have the same meaning as in the case of the *LtPacketUpdate* packet kind described previously.
- ♦ If the packet kind is neither *LtPacketInteraction*, *LtPacketUpdate*, nor *LtPacketRemoval*, then the packet represents non-HLA execution data (for example, annotations). This data is encoded much the same way a DIS packet is encoded. The `packet()` member function returns the data, and the `size()` member function returns the size (in bytes) of the data. All of the other member functions are meaningless in this context.

9.1.8. Using Large Files

If you want to specify use of files larger than 2GB, pass a value of 1 for the `useFD64` parameter when invoking either the constructors or the `init()` functions for *LtLoggerRecordFile* or *LtLoggerPlaybackFile*.

9.2. Editing Logger Files

One of the most common reasons for manipulating *LtLoggerFiles* is to read data from one Logger file, edit it, and write it to another Logger file.

i

The examples use several classes from VR-Link.

The following DIS example shows how to filter out all entity state messages from the entity with an ID of 3:4:5.

```
main()
{
    // Create the input and output file objects
    LtLoggerPlaybackFile infile;
    LtLoggerRecordFile outfile;

    // Initialize the input and output file objects
    infile.init( "inFile.lgr" );
    outfile.init( "outFile.lgr" );

    // Create a PDU factory that can create instances of the right kind
    // of DtPdu subclass depending on the PDU kind in the packet
    DtDisPduFactory factory;

    LtPacketInfo* info = infile.readPacket();
    for( ; info; info = infile.readPacket() )
    {
        bool writeIt = true;
        // Use the factory to create an instance of a DtPdu subclass
        DtPdu* pdu = factory.createPdu( (DtNetPacket*)info->packet() );
        if ( pdu )
        {
            if ( pdu->kind() == DtEntityStatePduKind )
            {
                // Cast to DtEntityStatePdu -- this is safe because we know that
                // this is the class the factory uses for DtEntityStatePduKind
                DtEntityStatePdu* esPdu = (DtEntityStatePdu*)pdu;
                // Check if the entity id matches the specified 3:4:5
                writeIt = ( esPdu->id().site() != 3 ||
                           esPdu->id().host() != 4 ||
                           esPdu->id().entityNum() != 5 );
            }
            delete pdu;
        }

        if ( writeIt )
        {
            outfile.writePacket( info );
        }
    }
}
```

```
    }
    delete info;
}
}
```

The following HLA example shows how to filter out all interactions:

```
int main()
{
    // Create the input and output file objects
    LtLoggerPlaybackFile infile;
    LtLoggerRecordFile outfile;

    // Initialize the input file object; pass it a NULL pointer as the
    // FOM descriptor to the target FOM (this means no handle-conversion)
    infile.init( "inFile.lgr", (const DtBaseFomDesc*)0 );

    // Initialize the output file object; pass it the FOM-descriptor from
    // the input file object
    outfile.init( "outFile.lgr", infile.fomDesc() );

    LtPacketInfo* info = infile.readPacket();
    for( ; info; info = infile.readPacket() )
    {
        if ( info->packetKind() != LtPacketInteraction )
        {
            outfile.writePacket( info );
        }
        delete info;
    }
}
```

9.3. Compiling and Linking with the Logger Library

To use the Logger file API, you must add the following include directive to your compile line:

```
-I loggerDir/include
```

where *loggerDir* is the path to the root of your Logger installation.

You must link with *libLgrFile*, as follows:

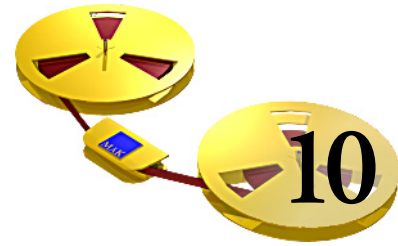
```
-L loggerDir/lib -lLgrFilex
```

The exact name of *LgrFile*x** depends on the simulation standard you are using (it ends with 5, HLA13, or HLA1516).

In addition, you should follow the directions for compiling and linking with VR-Link, in your VR-Link documentation. The Logger library link flag above must come before the VR-Link linking flags on your link line.



The preceding instructions are oriented towards UNIX. On a PC, make sure you are using the correct libraries in your project files, as listed in [“Using the Logger File API,”](#) on page 9-2.



Using Logger Tools

The Logger tools are utilities that let you combine, modify, and inspect Logger recordings. The Logger tools are provided in separate versions for HLA and DIS.

Viewing Logger Files with lgrDump	10-2
lgrDump for HLA.....	10-3
How lgrDump Displays Unexpected Message Types.....	10-5
Displaying Raw Data.....	10-5
Concatenating Files with lgrCat.....	10-6
Inserting Time Between Concatenated Files	10-7
Processing Input from Other Programs.....	10-7
Merging Files with lgrMerge	10-8
Using lgrOffset to Shift Messages in Time.....	10-9
Using lgrTrim to Extract Part of A File.....	10-10

10.1. Viewing Logger Files with *lgrDump*

The *lgrDump* utility displays the contents of a Logger file in a readable form. The output format is very similar to that of the *netdump* and *hlaNetdump* utilities included with the VR-Link Toolkit.

Usage:

```
lgrDump [filename]
```

Because *lgrDump* directs its output to standard output, you can redirect the output to a destination file or program as in these examples:

```
lgrDump logfile.lgr > output.txt
lgrDump logfile.lgr | more
```

- To pipe output from another program to *lgrDump*, omit the **filename** argument as in these examples:

```
lgrMerge File1.lgr File2.lgr | lgrDump
cat exercise_1.lgr | lgrDump
```



Owners of VR-Link can filter messages by specified field values by piping *lgrDump*'s output through the VR-Link *buffgrep* utility (UNIX only). See the VR-Link documentation for more information.

The following text shows a sample PDU as displayed by the DIS version of *lgrDump*.

```
*****
***** PDU #1077   Size=144           39.016 secs *****
*****
PduKind:           EntityStatePduKind   (1)
Version:           4
Exercise:          1
ProtocolFamily:    FamilyEntityInteraction   (1)
TimeStamp:         3297.213(seconds past hour)
TimeStampType:     Relative
Size:              144(bytes)
From:              1:15:1   (site:application:entity)
ForceID:           1
NumArtParams:      0
NumAttachedParts:  0
Type:              1:2:225:2:1:0:0 (kind:domain:country:cat:subcat:specific:extra)
Guise:             1:2:225:2:1:0:0 (kind:domain:country:cat:subcat:specific:extra)
Location:          { -2661471.288, -4358184.189, 3809002.286}
Velocity:          { 37.102, 87.402, 127.072}
Orientation:       { 1.169, -0.929, -2.305}
Acceleration:      { 0.000, 0.000, 0.000}
AngularVel:        { 0.000, 0.000, 0.000}
DrAlgorithm:       DrDrmRvw   (4)
Appearance:        0x00000000
  PaintScheme      _PaintSchemeUniform
  MobilityKill      _MobilityKillNone
  FirePowerKill     _FirePowerKillNone
  Damage            _DamageNone
```

```

Smoke           _SmokeNone
TrailingEffects _TrailingEffectsNone
Hatch           _HatchNA
Lights          _LightsNone
Flaming         _FlamingNo
FinalPdu        _NotFinalPdu
AfterBurner     UnknownEnum
Marking:        A10
CharSet:        1
Capabilities:    0x00000000

```

Each PDU description begins with a heading containing a PDU number, packet size, and time elapsed since the start of the Logger recording. This information is added by the Logger during recording, and is not a part of the actual PDU.

After the heading, *lgrDump* displays all fields of the PDU.

lgrDump prints version information as described in [Table 10-1](#):

Table 10-1: PDU versions and DIS protocol versions

PDU Version	DIS Protocol Version
4	2.0.4
5	IEEE 1278.1
6	2.1.4

10.1.1. lgrDump for HLA

The class, attribute and parameter handles printed by the HLA version of *lgrDump* may not be the actual handles that were originally recorded if you are not using the same FED file with *lgrDump* that you used when you recorded the file with the Logger. The handle numbers that are printed are those associated with the recorded names in the new FOM.

If you are using a different FED file with *lgrDump*, but would like to see the actual handles that were used at the time of recording, you can use *lgrDump*'s `-x` argument with an `execName` of "orig". This is only allowed when using *lgrDump* in raw mode. For example:

```
lgrDump -r -x orig myFile.lgr
```

In non-raw mode, we need to actually interpret the data, so we need to use the handles associated with the current FED file.

The HLA version of *lgrDump* decodes only the attribute data for entity objects. For attribute updates from other object classes, *lgrDump* prints the hex values of the attributes.

- To decode all the data, use the Logger to play the data stream for the VR-Link *hlaNetdump* utility.

The following is an example of output from lgrDump for an HLA exercise:

```
*****
***** Packet 143: Update at          3.496 secs *****
*****
ObjectID:      14300028
ObjectName:    "14300028"
ObjectClass:   ObjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft
TimeStamp:     0.000000
TimeStampType: Relative
Received 5 attributes:
  -AccelerationVector (1)
    12 bytes:
      c0 91 ca b7
      c0 e6 53 89
      40 da a1 00
  -AngularVelocityVector (2)
    12 bytes:
      b2 71 97 20
      3d 27 da 57
      2f af 95 aa
  -Orientation (7)
    12 bytes:
      c0 08 a5 b4
      3f 5e 77 62
      40 49 0f d9
  -WorldLocation (8)
    24 bytes:
      c1 44 8e ca
      fe 26 14 e1
      c1 50 3d 28
      97 a9 45 b8
      41 4d c3 71
      50 8d fc 21
  -VelocityVector (9)
    12 bytes:
      c2 c8 05 fe
      c3 1e 00 54
      c3 6a 33 cd
-----
EntityId:      0:0:0
EntityType:    0:0:0:0:0:0:0
Location:      { -2694549.986, -4256930.370, 3901154.629}
Velocity:      { -100.012, -158.001, -234.202}
Acceleration:  { -4.556, -7.198, 6.832}
Orientation:   { -2.135, 0.869, 3.142}
RotationVelocity: { -0.000, 0.041, 0.000}
DrAlgorithm:   DrOther (0)
Frozen:        FALSE
RelativeLocation: { 0.000, 0.000, 0.000}
RelativeVelocity: { 0.000, 0.000, 0.000}
RelativeAccel:  { 0.000, 0.000, 0.000}
RelativeOrientation: { 0.000, 0.000, 0.000}
RelativeRotationVel: { 0.000, 0.000, 0.000}
RelativeDrAlgorithm: DrOther (0)
RelativeFrozen: FALSE
HostGlobalId:
PartPosition:  OtherPosition (0)
Nature:        OtherNature (0)
RelativePosition: { 0.000, 0.000, 0.000}
```

```
StationName:      OtherStationName   (0)
StationNumber:    0
Range:            0.000000
Bearing:          0.000000
ForceId:          ForceOther         (0)
Guise:            0:0:0:0:0:0:0
Marking:
Acoustic Signature Index: 0
Radar Signature Index: 0
Infra-red Signature Index: 0
Measured Speed:   0
Capabilities:     0
Appearance:       0
```

10.1.2. How lgrDump Displays Unexpected Message Types

If *lgrDump* encounters unfamiliar message types, it displays formatted data for the messages.

DIS only

If the DIS version of *lgrDump* encounters a PDU of an incorrect DIS version (or a non-DIS PDU), it displays available header information followed by a hex dump of the rest of the PDU.

10.1.3. Displaying Raw Data

When you run *lgrDump* with the `-r` option (for raw), it behaves like the *nethex* utility that ships with VR-Link. For each attribute update or interaction received, it prints the name of the attribute, the size of the value of the attribute, and a hexadecimal dump of the value of the attribute.

10.2. Concatenating Files with lgrCat

The *lgrCat* utility concatenates messages from two or more Logger files. The resulting file contains all messages from the first file followed by all messages from the second file, and so on.

In the output file, messages from the first file retain their original times and message numbers. For each subsequent file, *lgrCat* increases the starting time and message numbers to create a continuous message sequence, as in [Example 10.1](#).

Example 10.1

If *File1.lgr* contains:

```
Msg #1 at time 5.0
Msg #2 at time 10.0
Msg #3 at time 20.0
```

and *File2.lgr* contains:

```
Msg #1 at time 2.0
Msg #2 at time 4.0
Msg #3 at time 6.0
```

then the concatenated file contains:

```
Msg #1 at time 5.0
Msg #2 at time 10.0
Msg #3 at time 20.0
Msg #4 at time 22.0
Msg #5 at time 24.0
Msg #6 at time 26.0
```

lgrCat modifies only the message numbers and time values generated by the Logger (those that appear in the message heading when viewed with *lgrDump*).

DIS only

Under DIS, *lgrCat* does not change the TimeStamp values contained in original PDUs.

HLA only

When you use the HLA version of *lgrCat*, all of the input Logger files must have been recorded using the same FOM. Files recorded using FOMs that differ from that used in the first file specified in the command are skipped.

Usage:

```
lgrCat [-s spacing] file1 [file2 file3...]
```

lgrCat directs its output to standard output similarly to the UNIX *cat* or DOS *type* command. In most cases, you will want to redirect or pipe the output to a destination file or program, and specify two or more files to be concatenated, as in these examples:

```
lgrCat file1 file2 [file3 file4...] > output.file
lgrCat exercise1.lgr exercise2.lgr > ex_1-2.lgr
lgrCat ex1.lgr ex2.lgr ex3.lgr | lgrDump | more
```

10.2.1. Inserting Time Between Concatenated Files

You can use the `-s spacing` argument to insert a period of time between files. The *spacing* argument specifies the number of seconds to insert between the final message in each file and the starting point of the next file (timepoint 00:00:00). For example, the command,

```
lgrCat -s 4.0 File1.lgr File2.lgr > combined.lgr
```

creates a *combined.lgr* file whose first three messages match those in the output file in [Example 10.1](#). Now, however, the last three messages have been delayed by four seconds and now occur at 26.0, 28.0, and 30.0 seconds, as in [Example 10.2](#):

Example 10.2

If *File1.lgr* contains:

```
Msg #1 at time 5.0
Msg #2 at time 10.0
Msg #3 at time 20.0
```

and *File2.lgr* contains:

```
Msg #1 at time 2.0
Msg #2 at time 4.0
Msg #3 at time 6.0
```

then the concatenated file will contain:

```
Msg #1 at time 5.0
Msg #2 at time 10.0
Msg #3 at time 20.0
(four-second space)
Msg #4 at time 26.0
Msg #5 at time 28.0
Msg #6 at time 30.0
```

10.2.2. Processing Input from Other Programs

- To pipe the output of another program to *lgrCat*, replace one of *lgrCat*'s input file arguments with a hyphen character to specify where the incoming data should appear in the concatenated output.

In the following example, *lgrCat* receives output piped from the *lgrOffset* program as its first input (indicated by the hyphen), and concatenates that data with *File2.lgr*. The combined data is directed to *Out.lgr*:

```
lgrOffset 20.0 < File1.lgr | lgrCat - File2.lgr > Out.lgr
```

10.3. Merging Files with lgrMerge

The *lgrMerge* utility combines one or more Logger files into a single file containing all messages from all files, sorted in chronological order and renumbered. [Example 10.3](#) illustrates a merge of two Logger files:

Example 10.3

If *File1.lgr* contains:

```
Msg #1 at time 1.0
Msg #2 at time 3.0
Msg #3 at time 5.0
```

and *File2.lgr* contains:

```
Msg #1 at time 2.0
Msg #2 at time 4.0
Msg #3 at time 6.0
```

then the merged file contains:

```
Msg #1 at time 1.0
Msg #2 at time 2.0
Msg #3 at time 3.0
Msg #4 at time 4.0
Msg #5 at time 5.0
Msg #6 at time 6.0
```

lgrMerge modifies only the message numbers and time values generated by the Logger (those that appear in the message heading when viewed with *lgrDump*).

DIS only

Under DIS, *lgrMerge* does not change the TimeStamp values contained in the original PDUs.

HLA only

When you use the HLA version of *lgrMerge*, all of the input Logger files must have been recorded using the same FOM. Files recorded using FOMs that differ from that used in the first file specified in the command are skipped.

Usage:

```
lgrMerge file1 file2 [file3 file4...]
```

Because *lgrMerge* directs its output to standard output, you typically redirect or pipe its output to a destination file or program as in these examples:

```
lgrMerge file1 file2 file3 > outputfile
lgrMerge exercise1.lgr exercise2.lgr > ex_1-2.lgr
lgrMerge ex1.lgr ex2.lgr ex3.lgr | lgrDump > PDUs.txt
```

10.4. Using *lgrOffset* to Shift Messages in Time

The *lgrOffset* utility can adjust the time of all messages in a file by a specified number of seconds. You can:

- ♦ Decrease or increase the timepoints of all messages in a file so they occur earlier or later.
- ♦ Remove an unknown length of dead time from the beginning of a file.

During recording, the Logger records the arrival time of each message relative to the start of recording. This is the only value that *lgrOffset* modifies; it is not a part of the original message. The original TimeStamp values in the messages are not affected.

Usage:

```
lgrOffset {time|first}
```

where:

- ♦ *time* is the desired time offset in seconds. *time* can be a positive or negative value.
- ♦ *first* removes a period of dead time from the beginning of a file. *lgrOffset* determines the timepoint of the first message, deletes the dead time from the start of the recording, and subtracts that number of seconds from the timepoint for each message in the file.

Because *lgrOffset* receives its input from standard input and directs its output to standard output, piping or redirection are used to specify the input source and output destination.

In either of the following two examples, *lgrOffset* increases the timepoints of all messages in *infile.lgr* by 15 seconds and saves the output in *outfile.lgr*:

```
lgrOffset 15.0 < infile.lgr > outfile.lgr  
lgrCat infile.lgr | lgrOffset 15.0 > outfile.lgr
```

In the next example, *lgrOffset* removes an unknown amount of dead time from the beginning of *infile.lgr* and saves the output in *outfile.lgr*:

```
lgrOffset first < infile.lgr > outfile.lgr
```

10.5. Using *lgrTrim* to Extract Part of A File

The *lgrTrim* utility reads and outputs a specified portion of a Logger file, in effect, trimming packets from the beginning and end of the file. You can specify the part of the file to be output, as either a range of messages or a span of time.



lgrTrim does not change the time stamps of messages. If you extract messages starting at, for example, 15 seconds, to a new Logger file, when you play the file, the first message does not get sent until 15 seconds after you start playing it.

Usage:

```
lgrTrim [-n] start [end]
```

where:

- ♦ *start* specifies the time point at which to begin extracting messages from the recording.
- ♦ *end* specifies the time point at which to stop extracting messages from the recording.
- ♦ *-n* indicates that *start* and *end* specify a numerical range of messages to extract.

lgrTrim discards any packets that fall outside of the specified range. If you do not specify an *end* value, *lgrTrim* outputs all packets from the specified *start* point to the end of the file.

By default, *lgrTrim* assumes that the *start* and *end* arguments are time values, in seconds.

Because *lgrTrim* receives its input from standard input and directs its output to standard output, piping or redirection are used to specify the input source and output destination.

Example 10.4

```
lgrTrim 5.0 10.0 < infile.lgr > outfile.lgr
```

lgrTrim extracts from *infile.lgr* all messages that occur between times 00:00:05 and 00:00:10 and saves them in *outfile.lgr*.

Example 10.5

```
lgrTrim -n 10 20 < infile.lgr > outfile.lgr
```

lgrTrim extracts the 10th through 20th messages from *infile.lgr* and saves them in *outfile.lgr*.

Example 10.6

```
lgrTrim -n 30 35 < exerc01.lgr | lgrDump
```

lgrTrim output piped to *lgrDump* for on-screen display of the 30th through 35th messages in *exerc01.lgr*.



Troubleshooting

This appendix describes how to identify and correct the most common network problems that prevent the Logger from sending and receiving data properly.

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

A.1. Troubleshooting Logger Network Communications (DIS Only)

In a typical DIS exercise, VR-Link applications (including the Logger) communicate with other applications using broadcast UDP/IP over a local area network. All applications in the exercise use the same IP broadcast address, and each application's host machine has a network interface device that is configured for that address. All machines must share a common netmask.

In addition, all participating applications must use the same UDP port.

If the Logger uses an IP address, netmask or port that is different from that used by your other applications, communication problems occur.

A.1.1. Using the Correct Broadcast Address and UDP Port

By default, the Logger communicates to the host machine's default broadcast address, using UDP port 3000. If the Logger appears to start normally, but fails to communicate with other DIS applications, the Logger may be using an incorrect broadcast address or UDP port. You can correct this problem by running the Logger with arguments that explicitly specify the correct address and port:

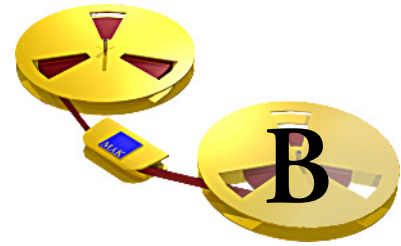
1. Determine the broadcast address used by your other DIS applications. This may be the network's default broadcast address, or may be specified through the application's startup command or configuration file at startup. For more information, see your System Administrator, or use the UNIX `ifconfig` 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
```



Logger MTL Parameters

This appendix lists the MTL commands that you can use in Logger configuration files.

MÄK Technologies Lisp (MTL) Files	B-2
 Using Environment Variables in An MTL File	B-2
 Specifying Lists of Lists.....	B-3
 Using Conditional Statements	B-3
Logger MTL Parameters	B-4

B.1. MÄK Technologies Lisp (MTL) Files

Many of the configuration files for MÄK Technologies products are ASCII text files that use MÄK Technologies LISP (MTL) to encode configuration information. MTL files have the extension *.mtl*. You can edit them in any text editor. Be sure that your text editor uses the proper end-of-line characters for your platform.

Most parameter entries take the format:

```
(setq parameter_name value)
```

For example:

```
(setq EnableSound 1)
```

Or:

```
(setqb parameter_name value)
```

The `setqb` syntax indicates a bound variable.

To comment out a line, precede the text with two semi-colons (`::`). You can also add a comment at the end of a line by preceding it with two semi-colons.

In most cases, the parameter names are known to the MÄK application and the MTL commands simply assign values to them. However, you can use the `setq` command to create arbitrary symbolic names or aliases that are then used in later commands. For example, in the following two commands (from the MÄK Stealth), the symbolic name `VaporTrail` is assigned to an `OpenFlight` file. Then the symbolic name is used in the `trail-map` command. The name `VaporTrail` has no intrinsic meaning to the Stealth. It is given meaning by the `setq` command, which allows it to be used later in the `trail-map` command.

```
(setq VaporTrail (list "../data/models/makTrails/Vapor-Trail" "VaporTrail.flt" 6.0 1.5 1.5 4.0 1.5 0.5))
(trail-map (list 1 2 -1 0 -1 -1 -1)(list VaporTrail 0.0 0.0 0.0))
```

In addition to the `setq` command, a commonly used command is the `load` command. This command instructs the application to load the file specified, for example:

```
(load mtl-path "params.mtl")
```

B.1.1. Using Environment Variables in An MTL File

If you want to use an environment variable in an MTL file, use the following syntax:

```
(setqb parameter_name (getenv (quote env_var)))
```

For example:

```
(setqb disDestAddr (getenv DEST_ADDR))
```

B.1.2. Specifying Lists of Lists

The example of setting a symbolic name in a previous section uses a list. You can also specify a list of lists, for example:

```
(trail-map (list 1 2 -1 -1 -1 -1 -1)
  (list vaporTrail -8.0 -3.0 0.0)
  (list vaporTrail -8.0 3.0 0.0)
)
```

A parameter entry can also have multiple individual lists:

```
(trail-map (list 1 2 -1 0 -1 -1 -1) (list VaporTrail 0.0 0.0
  0.0))
```

B.1.3. Using Conditional Statements

MTL commands can be conditional. Conditional statements are used frequently when parameters are different for DIS and HLA. A simple conditional statement has the format:

```
(if (condition) (statement_to_evaluate))
```

If you have more than one statement to evaluate, use a sequence block:

```
;; HLA specific parameters
(if (equal HLA 1)
  (sequence
    (setqb fomMapperLib "")
    (setqb fomMapperInitData "")
    (setqb rprFomVersion 1.0)
    (setqb pathToSublistFile "sublist.mtl")
    (setqb ignoreAdvisories 0)
    (setqb fedLookahead 1.0)
  )
)
```

B.2. Logger MTL Parameters

Table B-1 lists MTL configuration parameters for the Logger.

Table B-1: Logger MTL parameters (Sheet 1 of 3)

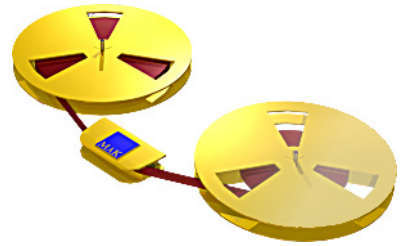
MTL Syntax	Description
allowRemoteFileDelete	Enables (1) or disables (0) file deletion in response to <i>DtLgrDelFilePdus</i> .
DtLoggerNotifyLevel	Specifies the notification level for Logger messages. Equivalent to the second value used in the <i>-n</i> startup option.
DtNotifyLevel	Specifies the verbosity of notification messages. Range 0-4. Equivalent to the first value used in the <i>-n</i> startup option. ♦ 0 = Fatal errors only ♦ 1 = Quiet ♦ 2 = Normal ♦ 3 = Verbose ♦ 4 = Debug. Default: 2.
enablePipelineStage	Specifies a list of logger pipeline stage plug-ins to enable.
entityIdStr	For DIS, specifies the site:host:entity for the Logger. For HLA, specifies the object name.
interfaceType	Specifies use of the text line interface, GUI, or both, as follows: ♦ 1 = graphical user interface ♦ 2 = text interface ♦ 3 = both.
language	Specifies the language to use if translation files are available.
mirrorLc	Enables (1) or disables (0) mirroring of Logger actions as Logger control messages.
noArtPartScaling	Enables (0) or disables (1) scaling of articulated parts data. Not recommended for use unless maximum optimization is required.
noFiltering	Enables (0) or disables (1) filtering. Not recommended for use unless maximum optimization is required.
pluginDir	Specifies the directory in which plug-ins are located.
plugins	Specifies a list of plug-ins to load.
processLc	Specifies whether or not to respond to Logger Control messages. ♦ 0 = Do not respond to Logger Control messages ♦ 1 = Interpret Logger Control messages only if they are addressed to the entity ID specified by the <i>-a</i> argument (default) ♦ 2 = Interpret all Logger Control messages.
recordLc	Enables (1) or disables (0) recording of Logger Control messages. Default: 0.

Table B-1: Logger MTL parameters (Sheet 2 of 3)

MTL Syntax	Description
useLargeFiles	Enables (1) or disables (0) support for recordings larger than 2 GB. Default: 0.
vrfMsgRangeLo	Specifies the lower range of VR-Forces data interactions (or data PDUs) to be filtered out. If value is greater than vrfMsgRangeHi, then no VR-Forces messages are filtered.
vrfMsgRangeHi	Specifies the higher range of VR-Forces data interactions (or data PDUs) to be filtered out. If value is less than vrfMsgRangeLo, then no VR-Forces messages are filtered.
HLA Only	
federateName	Specifies the name of the Logger federate.
federationName	Specifies the name of the federation execution.
fedFileName	Specifies the name of the FED file to use.
fedLookahead	Specifies the lookahead value for Time Management.
fomMapperInitData	Specifies initialization data to pass to a FOM Mapper library. Equivalent to the second portion of the <code>-f library:init_data</code> command.
fomMapperLib	Specifies a FOM Mapper library to use. Equivalent to the first portion of the <code>-f library:init_data</code> command.
ignoreAdvisories	Enables (1) or disables (0) sending of updates regardless of RTI advisory messages.
messageTagType	Specifies the content of message tags when the Logger generates messages that are not part of a recording. ♦ 0 = Use last tag for this object ♦ 1 = Compute current asciiTime timestamp using tape-time ♦ 2 = Compute current asciiTime timestamp using simulation time.
passiveSubscribe	Subscribe passively (1) or actively (0). Default: 0.
pathToSublistFile	Specifies the path of the optional subscription control file, which lists the objects and interactions that the Logger is subscribed to.
rprFomVersion	Specifies a version of the RPR FOM to use.
subscribedClasses	Specifies a list of classes for the logger to subscribe to. Use only in the sublist file.
subscribedInteractions	Specifies a list of interactions for the logger to subscribe to. Use only in the sublist file.
timeConstrainedRecord	Enables (1) or disables (0) recording using Time Management.
timeRegulatingPlayback	Enables (1) or disables (0) playback using Time Management.

Table B-1: Logger MTL parameters (Sheet 3 of 3)

MTL Syntax	Description
DIS Only	
<code>commonEntityType</code>	Specifies a list of entity type enumerations to include in the common entity types list in the Filter by Entity Type dialog box.
<code>customPdus</code>	Specifies a list of custom PDUs to be added to the list of PDUs in the Filter by PDU Kind dialog box.
<code>disPort</code>	Specifies the UDP port over which the Logger sends and receives data. (If used with <code>destAddr</code> , specifies TCPport.) Default: 3000.
<code>destAddr</code>	Specifies the IP address to which the Logger sends outgoing data packets. Default: Host machine's default broadcast address.
<code>exerciseId</code>	Specifies the exercise ID for the exercise.
<code>heartbeat</code>	Specifies the DIS heartbeat.
<code>multicastTtl</code>	Specifies the multicast time to live for messages.
<code>recvBufferSize</code>	Specifies the size of the buffer in which the Logger receives data packets. Default: 64000.
<code>scaleData</code>	Enables (1) or disables (0) scaling of velocity and acceleration data when playing greater than or lesser than real time.
<code>sendBufferSize</code>	Specifies the size of the buffer in which the Logger stores data packets for sending. Default: 64000.
<code>timeout</code>	Specifies the DIS timeout.
<code>useAsyncIO</code>	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 Logger 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.

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, Logger uses Euler angles or a rotation matrix that rotates from a reference frame to the body coordinate frame.

In the Stealth, when 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 from specific distances from the eyepoint.

clipping planes

The range of distances from the eyepoint (near clipping and far clipping) in which the Logger 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 Logger 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 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, 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.

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 Logger application connects to the network. State messages are sent through the exercise connection and information about remote entities is received through the exercise connection.

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 also reads 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 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. Logger 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 the Logger graphics display. 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 DOD-wide initiative to provide an architecture to support interoperability and reuse of simulations. The HLA supersedes DIS for the Department of Defense.

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.

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.

MTL

[MÄK Technologies Lisp](#).

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. 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.

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.

Realtime Platform Reference FOM

An [HLA](#) reference [FOM](#) based on the [DIS](#) protocol.

recording

A 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 the Logger 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.

smooth period

The period of time over which [trajectory smoothing](#) takes place, set to one second by default.

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

Logger simulation time is used in dead-reckoning of remote entities and thresholding of local entities. Typically, Logger 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

smoothing

A method of ensuring that transitions from an entity's dead-reckoned position to its actual position aren't 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.

terrain following

Causes the 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.

tape

An alternative term for a Logger recording. Not a physical magnetic tape.

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 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 the Logger to smooth positional discontinuities that could occur when new state updates arrive.

UDP port

A network channel through which the Logger 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

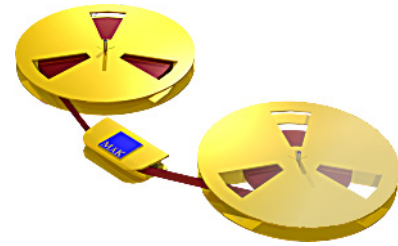
[Universal Transverse Mercator](#) coordinates.

view control messages

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

view floor

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

Symbols

? command 5-9

Numerics

64 bit mode 4-24

A

- A startup option 2-27, 4-25
- a startup option 2-23, 2-29, 4-23
- acceleration scaling 3-7, B-6
- action 8-13
- active subscription 8-11
- adding
 - annotation 4-16
 - new license file 2-10
 - pipeline stage 8-25
 - pipeline stages 8-24
- advisory message 4-23, B-5
 - ignoring 2-23
- aggregate_subordinates table 6-13
- aggregates table 6-11
- allowRemoteFileDelete parameter 7-4, B-4
- annotation 1-4, 4-16
 - clearing list 4-19
 - creating 4-16
 - deleting 4-18
 - editing 4-18
 - importing annotation file 4-17
 - moving to 4-18
 - opening annotation file 4-17
 - saving file 4-17
- API 8-2

- Logger file 9-2
- appending Logger recording 10-6
- application
 - including Logger in 8-4
- architecture 8-13
- articulated parts
 - pausing when Logger is paused 2-20
- asciiTime 2-27, B-5
 - timestamp 2-27
- asynchronous I/O 4-23, B-6
 - configuring 2-29
- attribute
 - myName 8-15
 - myStatus 8-15

B

- B startup option 2-28, 4-23
- b startup option 4-25
- beginning
 - moving to 4-7
- broadcast address 2-27, A-2, B-6
- buffer B-6
 - send and receive 2-28
 - size 4-23

C

- C startup option 4-25
- c startup option 2-19, 4-23
- case sensitivity of command 1-4
- changing
 - pipeline 8-20
 - playback speed 4-19
 - time point 4-8

- time scale 4-20
- time scale from keyboard 4-20
- class
 - DtLgrDelFilePdu* B-4
 - filtering by (HLA) 4-14
 - remote control 7-2
 - subscribed to 2-24, B-5
- clearing
 - annotations list 4-19
- clock
 - functions 8-6
- closing
 - annotations file 4-19
- command
 - fio64bit* 5-9
 - mirror* 5-10, 1-2
 - onPbAct* 5-9, 1-2
 - onRcAct* 5-10, 1-2
 - open* 5-9
 - passiveSubscribe* 5-10, 1-2
 - quite* 5-8
 - record* 5-10
 - remote control* 7-3
 - timeConstrained* 5-10, 1-2
 - timeRegulating* 5-10, 1-2
- comment
 - MTL parameter B-2
- commonEntityTypes** parameter 4-10, B-6
- compiling
 - API code 8-26
 - Logger library 9-10
- component files 2-5
- concatenating Logger recordings 10-6
- Configuration dialog box 4-5
- configuration file 4-23
 - RTI 2-15
 - specifying alternate 2-19
- configuration options 4-21
- configuring
 - asynchronous I/O 2-29
 - DIS 2-27
 - FED file 4-23
 - heartbeat 2-29
 - HLA message tags 2-27
 - list of common entity types 4-10
 - Logger 2-19
 - Logger for HLA 2-21
 - send and receive buffer size 2-28
 - sqlSender* plug-in 6-3, 6-5
 - system for MÄK RTI 2-14

- timeout 2-29
 - with API 8-9
- connecting
 - database, to 6-2
- controlling
 - classes and interactions subscribed to 2-24
 - Logger file 8-8
- conventions
 - manual 1-xiii
- creating
 - annotation 4-16
 - Logger file 9-5
 - plug-in 8-18
 - subscription control list 2-26
- custom PDU B-6
 - filtering 4-12
- customPdu** parameter 4-12, B-6

D

- d option 3-7
- D startup option 4-23
- d startup option 2-30, 4-26
- data
 - importing to spreadsheet 6-17
 - troubleshooting export of 6-7
- data scaling
 - disabling 2-30
- database
 - export to 3-8
 - keyword 6-2
 - schema 6-9
- DB Keyword 6-4
- DB Name 6-4
- dead reckoning
 - acceleration scaling 3-7
 - velocity scaling 3-7
- dead time, removing from start of recording 10-9
- deleting
 - annotation 4-18
 - files remotely B-4
 - Logger file 7-4
- destAddr** parameter B-6
- destination address 2-27
- detonations table 6-13
- direction
 - playback 4-19
- directory
 - ./bin* 2-5
 - ./doc* 2-5

.logger_tapes 2-5
 DIS 1-5
 asynchronous I/O 2-29
 configuring Logger for 2-27
 exercise ID 2-29
 host:application:entity ID 2-29
 timeout 2-29
 DIS port 4-24
 Logger default A-2
 specifying 4-24, A-2
 disabling
 articulated parts scaling B-4
 data scaling 2-30
 filtering 4-9, B-4
 sqlSender plug-in 6-6
 disPort parameter 2-28, B-6
 DMSO RTI 2-15
 DSN Keyword 6-4
 DSN Name 6-4, 6-17
 DT_USE_LGRGUI variable 8-4
 DtDisSimulate module 8-18
 DtDisSource module 8-18
 DtFileSource module 8-18
 DtHlaActiveSource module 8-18
 DtHlaPassiveSource module 8-18
 DtHlaSimulate module 8-18
DtLgrDelFilePds class B-4
 DtLoggerNotifyLevel parameter 2-20, B-4
 DtNotifyLevel parameter 2-20, B-4
 DtSendToFile module 8-18
 DtTcpSource module 8-18

E

-e option 3-8, 5-5
 -e startup option 4-26
 editing
 annotation 4-18
 Logger files 9-9
 Logger recording 10-10
 pipeline stage 8-25
 enablePipelineStage parameter 6-5, 8-20, 8-23, 8-24, B-4
 enabling
 articulated part scaling B-4
 filtering 4-9, B-4
 plug-ins B-4
 sqlSender plug-in 6-3, 6-5
 end
 moving to 4-7

end command 5-5, 5-9
 ending
 recording 4-5
 end-of-file action 3-8, 4-21
 in Logger 5-5
 end-of-file-action 4-26, 8-9
 entities table 6-9
 entity
 common types 4-10
 maintenance 3-7, 4-7
 entity ID B-4
 filtering by 4-11
 entity type
 filtering by 4-9, B-6
 removing from filtering list 4-10
 entityIdStr parameter 2-23, B-4
 DIS 2-29
 environment variable
 MAKLMGRD_LICENSE_FILE 2-9
 RTI_CONFIG 2-15
 EofAct command 5-9, 1-2
 example
 DtLogger 8-4
 remote control 7-5
 exercise
 recording 4-5
 exercise connection 8-4, 8-11
 exercise ID 2-29, B-6
 exerciseId parameter 2-29, B-6
 exiting Logger during script 5-7
 extracting part of a Logger recording 10-10

F

-F startup option 2-24, 4-23, 4-25
 -f startup option 2-21, 4-23
 FDD file 2-15
 FED file 1-5, 2-15
 changing 3-8
 for Time Management 3-4
 specifying 2-22, 4-23, B-5
 federate
 specifying Logger federate name 2-22
 specifying name B-5
 federateName parameter 2-22, B-5
 federation execution 1-5
 name 2-21, 4-24
 specifying B-5
 federation time 3-4
 federationName parameter 2-21, B-5

fedFileName parameter 2-22, B-5

fedLookahead parameter B-5

file

- configuration 2-19

- controlling Logger 8-8

- deleting B-4

- FED 2-15, 3-4

- larger than 2 GB 2-20

- license 2-8

- logger.mtl 2-19

- logger.mtl* 2-19, 8-21, 8-23

- MTL B-2

- playing 4-6

- RID 2-15

- rti.mtl 2-15

- RTI.RID* 2-15

- sublist.mtl* 2-24

- VR-Link.fed 2-22

file position 8-8

filter 8-13

Filter By Class dialog box 4-14

Filter By Entity Type dialog box 4-9

Filter By EntityID-PDUKind Pairs dialog box 4-11, 4-14

Filter By Object dialog box 4-13, 4-14

Filter By PDU Kind dialog box 4-12

filtering

- classes (HLA) 4-14

- configuring common entity types 4-10

- custom PDUs 4-12

- enabling and disabling 4-9

- enabling or disabling B-4

- entity ID 4-11

- entity type 4-9, B-6

- input and output 4-9

- loading settings 4-15

- objects 4-13

- PDU kind 4-12

- saving and loading settings 4-14

- saving settings 4-15

fio64bit command 5-9

fires table 6-15

flexlm

- directory location 2-7

- installing 2-7

- license file 2-8

- license server name 2-8

- multiple licenses 2-10

- troubleshooting 2-12

FLEXlm license manager 2-6

FOM 2-14

- flexibility 3-8

- supported 1-5

FOM class 8-11

FOM Mapper 1-5, 2-21, 2-22, 4-23, B-5

fomMapperInitData parameter 2-21, B-5

fomMapperLib parameter 2-21, B-5

G

-G startup option 4-23

-g startup option 2-19, 4-23

getting

- Logger state 7-4

global object designator

- specifying 2-23

go command 5-9

GUI

- interface 8-3

- localizing 2-16

- specifying 2-19

H

-h startup option 4-23

h command 5-9

heartbeat 5-10, 8-11, B-6

- defined 3-6

- setting 2-29, 4-21, 4-22

- setting in text interface 5-7

heartbeat parameter 2-29, B-6

help command 5-9

HLA 1-5

- advisory messages 2-23

- configuring for 2-21

- FOM flexibility 3-8

- global object designator 2-23

- specifications supported 1-5

- subscribing passively 2-23

- subscribing to classes and interactions 2-24

- subscription list 2-26

- support for 3-8

- Time Management 2-23, 3-3, 4-5, 4-6, 4-18

host ID

- getting 2-8

host name 2-8

I

- i command 5-3, 5-9
- I startup option 2-29, 4-23
- i startup option 2-23, 4-23
- IEEE 1516 1-5
- ignoreAdvisories parameter 2-23, B-5
- ignoring
 - advisory messages 2-23
- importing
 - annotation file 4-17
 - data to spreadsheet 6-17
- info command 5-3, 5-4, 5-9
- informational message 2-20
- initializing
 - plug-in 8-19
- input
 - filtering 4-9
- installing
 - DMSO RTI 2-15
 - flexIm software 2-7
 - RTI 2-14
- interaction
 - subscribed to 2-24, B-5
 - subscribing to 2-23
- interface 4-23
 - customizing 8-3
- interfaceType parameter B-4
- interim PDU 2-29, 3-6, 3-7
- IP address 4-25
 - outgoing packets B-6

J

- Jump to Selected Time button 4-18
- jumping
 - to beginning of recording 4-7
 - to end of recording 4-7

K

- keyboard
 - changing time scale 4-20
- keyword
 - database 6-2

L

- L startup option 2-23, 4-24
- l startup option 2-29, 4-26

- language
 - specifying B-4
- language parameter 2-18, B-4
- large file 2-20, 4-24, 8-12, 9-9, B-5
- LBTS 3-4
- lgrCat 10-6
 - effect on PDU timestamp data 10-6
 - pipng output from another program 10-7
- lgrDump 10-2
 - compared with netdump 10-2
 - display of unexpected PDU types 10-5
- lgrMerge 10-8
- lgrOffset 10-9
- lgrTrim 10-10
- libLgrcore library 8-14
- library
 - compiling and linking Logger 9-10
 - libLgrcore 8-14
 - Logger API 8-26
 - provided for Logger file API 9-2
- license file 2-8
 - adding new 2-10
- license server
 - name 2-8
 - running 2-11
 - status 2-11
 - stopping 2-11
 - troubleshooting 2-12
- linking
 - API code 8-26
 - Logger library 9-10
- lmdown program 2-11
- lmhostid 2-8
- lmstat program 2-11
- loading
 - filtering settings 4-14, 4-15
 - plug-in 8-23
 - recording 4-6
 - subscription control list 2-27
- localizing
 - GUI 2-16
- Logger
 - including in application 8-4
- Logger API
 - architecture 8-13
 - compiling and linking 8-26
 - configuration functions 8-9
 - protocol-specific parameters 8-11
- Logger Control

- message 4-23, 4-24, B-4
 - capturing user action 7-3
- PDU's 7-2
- Logger file
 - API 9-2
 - creating 9-5
 - deleting 7-4
 - editing 9-9
 - reading 9-2
 - reading data 9-4
 - writing 9-6
 - writing to 9-5
- Logger Interface component 8-2
 - classes 8-3
- Logger library
 - compiling and linking 9-10
- Logger Pipeline Stages Manager dialog box 8-25, 8-26
- Logger recording
 - combining, using *lgrMerge* 10-8
 - concatenating with *lgrCat* 10-6
 - editing 10-10
 - extracting a portion of 10-10
 - modifying time data with *lgrOffset* 10-9
 - removing dead time from beginning 10-9
 - samples 1-4
 - viewing PDU contents of 10-2
 - viewing with Stealth 1-4
- Logger thread 8-2
- logger.mtl* configuration file 2-19, 8-21, 8-23
- logger.mtl file 2-19
- lookahead 4-26, B-5
- lookahead time 5-9
- lower bound time stamp 3-4

M

- m startup option 4-24, 7-3
- MÄK RTI
 - configuring 2-14
- MÄK Technologies LISP B-2
- MAKLMGRD_LICENSE_FILE environment variable 2-9
- manual
 - conventions 1-xiii
- map datum 6-5
- Master DB Name 6-4
- Master DSN Name 6-4
- message
 - advisory 2-23

- informational 2-20
- message tag B-5
 - configuring 2-27
 - specifying 2-27
- messageTagType** parameter 2-27, B-5
- mirror command 5-10, 1-2
- mirroring B-4
- mirrorLc** parameter 7-3, B-4
- mode 8-9
- module
 - DtDisSimulate 8-18
 - DtDisSource 8-18
 - DtFileSource 8-18
 - DtHlaActiveSource 8-18
 - DtHlaPassiveSource 8-18
 - DtHlaSimulate 8-18
 - DtSendToFile 8-18
 - DtTcpSource 8-18
 - TcpSendObj 8-18
- moving
 - amount of time 4-8
 - to annotation point 4-18
 - to beginning of recording 4-7
 - to end of recording 4-7
 - to point in time 4-8
- MTL file B-2
- MTL parameter 2-19
 - allowRemoteFileDelete** 7-4
 - entityIDStr 2-23
 - in DIS 2-29
 - heartbeat 2-29
 - recvBufferSize 2-28
 - rprRomVersion 2-21
 - sendBufferSize 2-28
 - timeout 2-29
 - useAsyncIO 2-29
- multicast address 2-28, 4-25
- multicast time to live B-6
- multicastTtl** parameter 2-29, B-6
- myName attribute 8-15
- MySQL 6-2
- myStatus attribute 8-15

N

- N startup option 2-22, 4-24
- n startup option 2-20, 4-24, B-4
- name
 - Logger federate 2-22
- network, requirements for Logger A-2

New Pipeline Stage Object dialog box 8-25

noArtPartScaling parameter B-4

noFiltering parameter 4-9, B-4

notification level 4-24, B-4
setting 2-20

O

-o startup option 4-26

object

filtering by 4-13

name B-4

subscribing to 2-23

ODBC driver 6-2

ODBC Driver Name 6-4

onPbAct command 5-9, 1-2

onRcAct command 5-10, 1-2

open command 5-9

opening

annotation file 4-17

output

filtering 4-9

overriding

DtLogger class 8-22

P

-P startup option 2-23, 2-28

DIS 4-24

HLA 4-25

-p startup option 4-26

packet

preventing dropped 2-29

processing 8-17

sending and receiving 2-28

parameter

allowRemoteFileDelete 7-4, B-4

commenting out B-2

commonEntityTypes 4-10, B-6

customPdus 4-12, B-6

destAddr B-6

disPort 2-28, B-6

DtLoggerNotifyLevel 2-20, B-4

DtNotifyLevel 2-20, B-4

enablePipelineStage 6-5, 8-20, 8-23, 8-24, B-4

entityIDStr

in DIS 2-29

entityIdStr 2-23, B-4

exercisId 2-29, B-6

federateName 2-22, B-5

federationName 2-21, B-5

fedFileName 2-22, B-5

fedLookahead B-5

fomMapperInitData 2-21, B-5

fomMapperLib 2-21, B-5

heartbeat 2-29, B-6

ignoreAdvisories 2-23, B-5

interfaceType B-4

language 2-18, B-4

messageTagType 2-27, B-5

mirrorLc 7-3, B-4

multicastTtl 2-29, B-6

noArtPartScaling B-4

noFiltering 4-9, B-4

passiveSubscribe B-5

pathToSublistFile B-5

pluginDir 6-2, 8-23, B-4

plugins 6-2, 8-23, B-4

processLc B-4

protocol-specific in API 8-11

recordLc B-4

recvBufferSize 2-28, B-6

rprFomVersion 2-21, B-5

scaleData B-6

sendBufferSize 2-28, B-6

subscribedClasses B-5

subscribedInteractions B-5

timeConstrainedRecord 2-23, B-5

timeout 2-29, B-6

timeRegulatingPlayback 2-23, B-5

useAsyncIO 2-29, B-6

useFD64 2-20

useLargeFiles 2-20, B-5

useTimeManagement 2-23

vrfMsgRangeHi B-5

vrfMsgRangeLo B-5

passive subscription 2-23, 4-25, 8-11, B-5

passiveSubscribe command 5-10, 1-2

passiveSubscribe parameter B-5

Password 6-5

Password Keyword 6-5

path

specifying for RTI 2-14

pathToSublistFile parameter B-5

pause

command 5-10

compared to stop 5-4

pausing playback 4-6

PDU

filtering B-6

- filtering by 4-12
- filtering custom 4-12
- how modified in Logger tape 10-3
- interim 2-29
- remote control 7-2
- timing out 2-29
- viewing with *lgrDump* 10-2
- pipeline
 - architecture 8-13
 - changing with a plug-in 8-20
 - processing packets 8-17
 - stages 8-14
- pipeline stage
 - adding 8-25
 - class 8-18
 - functions 8-14
 - managing from GUI 8-24
 - object 8-12
 - removing 8-26
 - reserved name 8-18
- play action 4-21, 5-4
- Play button 4-6
- Play In Time Regulated Mode check box 4-6
- playback 3-3, 4-6, 4-26, 5-3, 5-4
 - changing speed or direction 4-19
 - described 3-3
 - direction 4-19, 5-6
 - heartbeat 2-29
 - in reverse 3-5
 - mode 3-3, 4-5, 4-26
 - moving to beginning 4-7
 - moving to end 4-7
 - pausing 4-6
 - restrictions in time-managed mode 3-4
 - speed 4-19, 5-6
 - allowable range of 4-20, 5-6
 - stopping 4-6
 - Time Management 3-3
 - time regulated B-5
 - time-managed 2-23
- playback mode 4-6
- Playback window 4-3, 4-5, 4-6
- playing
 - recording 4-6
- plug-in
 - changing pipeline 8-20
 - configuring B-4
 - creating 8-18
 - directory 2-5
 - directory for B-4

- enabling B-4
- initializing 8-19
- loading 8-23
- sqlSender 3-8
- pluginDir** parameter 6-2, 8-23, B-4
- plugins directory 2-5
- plugins** parameter 6-2, 8-23, B-4
- port B-6
 - specifying UDP 2-28
- Portable Document Format (PDF) x
- processing
 - packets 8-17
- processLc** parameter B-4
- products
 - technical support xii
 - upgrades xii
- protocol
 - supported 1-5
- protocol-specific parameter
 - Logger API 8-11

Q

- Q startup option 2-20, 4-24
- q startup option 2-27
- Qt Linguist 2-16
- query 6-2
- quit command 5-8, 5-9

R

- R startup option 4-25
- r startup option 4-25
- reading
 - data from Logger file 9-4
 - Logger file 9-2
- receive buffer B-6
- Receive Order 3-4
- receiving
 - packets 2-28
- record
 - action 4-21, 5-2
 - command 5-10
 - mode 3-2, 4-5, 4-25, 5-2
- Record In Time Constrained Mode check box 4-5
- record mode 4-6
- Record window 4-3, 4-5, 4-6
- recording 3-2, 4-25, 5-2
 - annotating 4-16
 - controlling 8-8

- described 3-2
- exercise 4-5
- exporting to database 3-8
- jumping to beginning 4-7
- jumping to end 4-7
- large 4-24
- loading 4-6
- pausing 4-6
- playing 4-6
- playing in text interface 5-3
- starting 4-5
- status of 5-3
- status, displaying 5-3
- stopping 4-5
- time constrained B-5
- Time Management 3-3
- time-managed 2-23
- recordLc** parameter B-4
- recvBufferSize** parameter 2-28, B-6
- remote control 4-25, 7-2
 - commands 7-3
 - commands for 7-3
 - example 7-5
 - Time Management 7-5
- remote control interface
 - interface 8-3
- remote file deletion B-4
- removing
 - pipeline stage 8-26
 - pipeline stages 8-24
- reserved name 8-18
- reverse playback
 - restriction 3-4
- RID file 2-15
- rid.mtl file 2-15
- RO 3-4
- RPR FOM 2-21, 3-9, B-5
 - specifying version to use 2-21
- rprFomVersion** parameter 2-21, B-5
- RTI 2-14
 - definition of 2-14
 - DMSO, installing 2-15
 - specifications supported 1-5
- RTI_CONFIG environment variable 2-15
- rtiexec 2-15
- RTI.rid* file 2-15
- runLm program 2-11
- running
 - license server 2-11
- run-time commands

- ? 5-9
- end 5-5, 5-9
- EofAct mode 5-9, 1-2
- go 5-3, 5-4, 5-9
- h 5-9
- heartbeat 5-10
- help 5-9
- info 5-4, 5-9
- pause 5-4, 5-10
- quit 5-9
- scale 5-6, 5-10
- skip 5-5, 5-10
- start 5-5, 5-9
- stop 5-3, 5-4, 5-9
- time 5-5, 5-10
- timeout 5-10
- wait 5-8, 5-9

S

- S startup option 2-28, 4-25
- s startup option 4-26, 5-6
- sample recording 1-4
- saving
 - annotation file 4-17
 - filtering settings 4-14, 4-15
 - subscription control list 2-26
- scale command 5-6, 5-10
- scaled tape-time 8-6
- scaleData** parameter B-6
- scaling 4-26
 - articulated parts 2-20, B-4
 - time 3-7
 - velocity and acceleration 2-30, B-6
- schema
 - database 6-9
- script
 - creating 5-7
 - executing 5-7
 - quitting Logger while running 5-7
 - stopping 5-8
 - wait command 5-8
- send 8-13
- send buffer B-6
- sendBufferSize** parameter 2-28, B-6
- sending
 - packets 2-28
- Server Keyword 6-4
- Server Name 6-5
- Set Time 3-7

- entity maintenance 4-7
- Set Time dialog box 4-8
- Set Time Scale menu option 4-20
- setting
 - heartbeat 2-29, 5-7
 - notification level 2-20
 - time 4-8
 - time scale 4-20
 - timeout 2-29, 5-7
- shutting down
 - license server 2-11
- simulation
 - recording 4-5
 - standard
 - supported 1-5
- SISO 1516 API 1-5
- site
 - host
 - entity 4-23, B-4
- skip command 5-5, 5-10
 - entity maintenance 3-7
- Skip Time 3-7
 - entity maintenance 4-7
- Skip Time dialog box 4-8
- skipping
 - forward or backward 4-8
- slider
 - time scale 4-19
- slow-speed playback 3-5, 5-6
- software components 1-2
- source 8-13
- specifying
 - configuration file 2-19
 - FED file 2-22, 4-23
 - federation execution name 2-21
 - global object designator 2-23
 - message tag 2-27
 - RPR FOM version 2-21
 - Time Management 2-23
 - UDP port 2-28
 - user interface type 2-19
- speed
 - changing for playback 4-19
- spreadsheet
 - importing data into 6-17
- SQL API
 - database schema 6-9
- SQL database
 - connecting to 6-2
- SQL export 3-8
- SQL Send Plugin Control dialog box 6-3, 6-6
- sqlSender
 - application 3-8
 - plug-in 3-8
- sqlSender application 6-5, 6-7
 - arguments 6-7
- sqlSender plug-in
 - configuring and enabling 6-3, 6-5
 - disabling 6-6
 - loading 6-2
- stages
 - pipeline 8-14
- start
 - command 5-5, 5-9
 - moving to 4-7
- starting
 - command line mode 5-2
 - recording 4-5
 - text interface 5-2
- starting Logger 4-4
- startup
 - from DOS 4-4
 - from icon under Windows 4-4
 - playback mode 3-3
 - record mode 3-2, 5-2
 - UNIX 4-4
 - Windows 4-4
- startup option 2-19
 - A 2-27, 4-25
 - a 2-23, 2-29, 4-23
 - B 2-28, 4-23
 - b 4-25
 - C 4-25
 - c 2-19, 4-23
 - D 4-23
 - d 2-30, 3-7, 4-26
 - e 3-8, 4-26, 5-5
 - F 2-24, 4-23, 4-25
 - f 2-21, 4-23
 - G 4-23
 - g 2-19, 4-23
 - general format 1-4
 - h 4-23
 - I 2-29, 4-23
 - i 2-23, 4-23
 - L 2-23, 4-24
 - l 2-29, 4-26
 - m 4-24, 7-3
 - N 2-22, 4-24
 - n 2-20, 4-24

- o 4-26
 - P 2-23, 2-28, 4-24, 4-25
 - p 4-26
 - Q 2-20, 4-24
 - q 2-27
 - R 4-25
 - r 4-25
 - S 2-28, 4-25
 - s 4-26, 5-6
 - startup under Windows 4-4
 - T 4-26
 - t 4-25
 - table of 4-23
 - U 2-23, 4-26
 - u 2-23, 4-25
 - w 2-23, 4-24
 - x 2-21, 2-22, 2-29, 4-24
 - Y 4-24
 - y 4-9, 4-24
 - Z 4-25
 - z 4-24
 - state
 - Logger 7-4
 - status
 - API functions 8-9
 - field 4-5
 - license server 2-11
 - message, in playback 5-4
 - recording 5-3
 - stop command
 - compared to pause 5-4
 - in playback mode 5-9
 - in record mode 5-3
 - stopping
 - license server 2-11
 - playback 4-6
 - recording 4-5
 - script 5-8
 - sublist.mtl* file 2-24
 - subscribedClasses** parameter B-5
 - subscribedInteractions** parameter B-5
 - subscribing
 - classes and interactions 2-24
 - passively B-5
 - to classes and interactions B-5
 - to objects and interactions 2-26
 - subscription
 - active or passive 8-11
 - control 2-26
 - control file 4-25, B-5
 - list 2-24
 - passive 2-23
 - subscription control list
 - creating 2-26
 - loading 2-27
 - saving 2-26
 - support
 - technical xii
- ## T
- T startup option 4-26
 - t startup option 4-25
 - tape
 - loading 4-6
 - playing 4-6
 - starting 4-5
 - tapeLength() 8-8
 - tape-time 8-6
 - TCP 4-24
 - port B-6
 - TcpSendObj module 8-18
 - technical support xii
 - text interface 8-3
 - starting 5-2
 - text-based interface
 - specifying 2-19
 - thread 8-2
 - time
 - between concatenated Logger tapes 10-7
 - interval 8-7
 - navigating during playback 4-7, 5-5
 - navigation 4-7
 - 5-5
 - setting 4-8
 - shifting PDU times 10-9
 - skipping 4-8
 - time clock
 - functions 8-6
 - time command 5-5, 5-10
 - entity maintenance 3-7
 - time constrained recording B-5
 - time jump, updating entities after 3-7
 - Time Management 2-23, 3-3, 4-3, 4-5, 4-6, 4-18, 4-25, 4-26, 5-2, 8-7, B-5
 - controlling remotely 7-5
 - FED file 3-4
 - lookahead 4-26, B-5
 - restrictions 3-4
 - scaling playback 3-5

- specifying use of 2-23
- time regulated playback B-5
- time scale
 - changing from keyboard 4-20
 - described 3-5
 - setting 4-19, 4-20, 5-6
 - slider 4-19
 - specifying at startup 5-6
 - with Time Management 3-5
- Time Stamp Order 3-4
- time-constrained 3-3
- timeConstrained command 5-10, 1-2
- timeConstrainedRecord parameter 2-23, B-5
- timeout 8-11, B-6
 - command 5-7
 - defined 3-6
 - setting 2-29, 5-10
 - setting in text interface 5-7
- timeout parameter 2-29, B-6
- timepoint 4-26
 - setting 4-7
 - setting with skip command 5-5
 - setting with time command 5-5
- time-regulating 3-3
- timeRegulating command 5-10, 1-2
- timeRegulatingPlayback parameter 2-23, B-5
- timescale 4-26, 5-6, 8-6
 - during recording 4-5
 - effects 3-5
- timestamp 2-27, B-5
- tools 10-1
- translating
 - GUI text 2-16
- trimming beginning or end of Logger recording 10-10
- troubleshooting
 - data export 6-7
 - flexlm 2-12
- TSO 3-4

U

- U startup option 2-23, 4-26
- u startup option 2-23, 4-25
- UDP port 2-28, 4-24, B-6
 - specifying 2-28, 4-24
- UNIX
 - compared to Windows versions 1-4
 - compiling and linking API 8-26

- starting
 - Logger 4-4
- upgrades xii
- useAsyncIO parameter 2-29, B-6
- useFD64 parameter 2-20
- useLargeFiles parameter 2-20, B-5
- user action as Logger Control message 7-3
- User ID 6-5
- user interface
 - specifying type 2-19
- User Keyword 6-5
- User's Guide
 - organization of ix
- useTimeManagement parameter 2-23

V

- variable
 - environment 2-9
- velocity
 - scaling 3-7, B-6
- View Annotations window 4-16
- view control PDUs 1-4
- vrfMsgRangeHi parameter B-5
- vrfMsgRangeLo parameter B-5
- VR-Forces data interactions B-5
- VR-Link.fed file 1-5, 2-22

W

- w startup option 2-23, 4-24
- wait command 5-8, 5-9
- WGS84 6-5
- window
 - Playback and Record 4-3
- Windows
 - compared to UNIX versions 1-4
 - compiling and linking API 8-26
 - starting Logger 4-4
- wrapping 4-22
 - at end of file 3-8, 5-5
 - in reverse playback 3-8
- writing
 - Logger file 9-5, 9-6

X-Y-Z

- x startup option 2-21, 2-22, 2-29
 - HLA 4-24

XML

- RTI configuration file 2-15
- Y startup option 4-24
- y startup option 4-9, 4-24
- Z startup option 4-25
- z startup option 4-24

Table QR-1: MÄK Data Logger Startup Options Quick Reference (Sheet 1 of 2)

Option	Description
Record and Playback	
-a {"site:host:entity" "hla_object_name"}	DIS: Specifies the site:host:entity for remote control of the Logger. HLA: Specifies the object name (global object designator in VR-Link.)
-B recvBSz:sendBSz	DIS: Specifies the receive and send buffer size.
-c file_name	Specifies the Logger configuration file (MTL file).
-D {0 1}	Enables (1) or disables (0) deletion of files in response to Logger Control messages.
-f {vLRPRxx library:init_data}	HLA: Specifies the FOM Mapper to use if you are not using the default FOM.
-F FED_file	HLA: Specifies FED file to use.
-g {w t b h}	Specifies the interface to use. w = GUI (default); t = Text interface; b = Both; h = Hidden.
-G language	Specify the language for the Logger GUI to use.
-h	Displays a brief summary of Logger command-line arguments.
-i {0 1}	HLA: Ignore (0) or do not ignore (1) HLA advisory messages.
-I {0 1}	DIS: Enables (1) or disables (0) use of asynchronous I/O. Default: 0.
-k script	Windows: Specifies a script file to run when the Logger starts.
-L {0 1 2}	Specifies the response to Logger Control messages for remote control, where: 0 = Do not respond to messages; 1 = Interpret messages only if they are addressed to the entity ID specified by the -a argument (default); 2 = interpret all messages
-m {0 1}	Enables (1) or disables (0) sending of a Logger control PDU each time you change the the Logger's state.
-n genNotify:logNotify	Specifies a general notification level and a Logger-specific notification level. Range 0-4.
-N federate_name	Specifies the federate name for the Logger federate.
-P dis_port	DIS: Specifies which DIS port (UDP port) the Logger uses. Default: 3000.
-Q {0 1}	Enables (1) or disables (0) 64 bit mode, which supports recordings larger than 2 GB.
-w {0 1}	HLA: Enables (1) or disables (0) use of Time Management, but does not select it. Default: 0.
-x ExName	HLA: Sets federation execution name. Default: VR-Link
-x exerciseID	DIS: Specifies the exercise ID.
-y {0 1}	Enables (0) or disables (1) filtering. Default: 0.
-Y {0 1}	Enables (0) or disables (1) articulated parts scaling. Default: 0.
-z logfilename	Specifies a Logger log file. Default: <i>logger.txt</i> .
-Z lownum:highnum	Specifies filtering of messages between the VR-Forces front-end and back-end.
Record Mode Only	
-b {r s}	Start the Logger in a stopped state (s) or record immediately (r) when opening a file for recording.
-P {0 1}	HLA: Subscribe actively (0) or passively (1). Default: 0.
-r filename	Causes the Logger to open in record mode. Specifies the destination file for recorded data.
-R {0 1}	Enables (1) or disables (0) recording Logger Control messages during remote control. Default: 0.
-S sublist.mtl	HLA: Specifies a subscription control file to load.
-S mcastAddr	DIS: Subscribes the Logger to the multicast address specified by <i>mcastAddr</i> .
-t hostname:port	Specifies the TCP host to get data from.
-u {0 1}	HLA: Enable and select (1) , or disable (0) use of time-constrained mode. Default: 0. Overrides -w.
Playback Mode Only	
-A addr	DIS: Sets the default IP address to which the Logger sends PDUs.
-C output_port	Specifies TCP port to send data to.
-d {0 1}	Enables (0) or disables (1) the generation of velocity and acceleration scaling data when playing at speeds above or below normal.
-e {p w s}	Specifies the Logger's action upon reaching the end of the file. p = pause; w = wrap; s = stop .
-l multicastTtl	DIS: Specifies the multicast time to live value.
-l lookahead	HLA: Specify the lookahead, in seconds.
-o {p s}	Open file in stopped state (s) or play immediately (p).
-O	Dumps output to <i>screendump.txt</i> .
-p [filename]	Causes the Logger to open in playback mode. Specifies the source file to be played.

Table QR-1: MÄK Data Logger Startup Options Quick Reference (Sheet 2 of 2)

Option	Description
-q {0 1 2}	HLA: Specifies an interpretation for message tags. 0 = None; 1 = Time interval from last packet with message in tape-time; 2 = Interval is simulation time.
-s <i>timescale</i>	Sets the initial playback timescale factor.
-T <i>timepoint</i>	Sets the timepoint in the file where playback begins, in HH:MM:SS format. Default: 00:00:00
-U {0 1}	HLA: Enables (1) and selects or disables (0) time-regulation. Default: 0. -U overrides -w.

Table QR-2: Logger Text Interface Commands Quick Reference

Command	Description
Record Mode and Play Mode	
fio64bit {0 1}	Enables (1) or disables (0) use of 64 bit file IO.
go	Starts or resumes recording or playback from a previously stopped or paused state.
stop	Halts playback at the current timepoint, stops recording or transmission.
quit	Closes the Logger file and exits.
info or i	Displays a brief status report including the log file name, current timepoint, timescale, and current play state.
h, help, or ?	Displays a summary of Logger run-time commands.
wait <i>mode</i>	Used only in scripts. Specifies how long to wait before processing the next command in the script. Options: <i>MODE_STOP</i> , <i>MODE_PAUSE</i> , <i>TAPE_END</i> , <i>TAPE_START</i> , <i>TAPE_TIME time</i> , <i>time-interval</i> .
Play Mode Only	
start	Sets the current timepoint to the beginning of the recording.
open <i>filename</i>	Opens the specified Logger recording for playback.
go	Begins playing the Logger recording.
end	Sets the current timepoint to the end of the recording.
onEofAct <i>mode</i>	Sets end of file action. <i>Mode</i> can be wrap, stop, or pause.
onPbAct <i>mode</i>	Sets Logger action when you load a file for playback <i>Mode</i> can be play or stop.
pause	Pauses playback at the current timepoint.
scale [<i>factor</i>]	Sets the timescale <i>factor</i> that determines playback speed and direction.
time <i>timepoint</i>	Navigates to a specific <i>timepoint</i> in the recording. Specify <i>timepoint</i> in HH:MM:SS format.
skip <i>offset</i>	Moves the current timepoint forward or backward by the specified time <i>offset</i> .
heartbeat [<i>sec</i>]	DIS: Specifies heartbeat for interim PDUs during slow-speed playback. Default: 5.
timeout [<i>sec</i>]	DIS: Specifies the timeout for interim PDUs during slow-speed playback. Default: 12 recorded seconds.
timeRegulating {0 1}	HLA: Enables (1) or disables (0) time regulating mode when you are using HLA Time Management.
Record Mode Only	
mirror {0 1}	Enables (1) or disables (0) mirroring of user actions as Logger control messages.
onRcAct <i>mode</i>	Specifies Logger mode when it creates a file for recording. <i>Mode</i> can be record or stop.
passiveSubscribe {0 1}	HLA: Enables (1) or disables (0) passive subscription to HLA update messages.
record <i>filename</i>	Begins recording to the specified filename.
timeConstrained {0 1}	HLA: Enables (1) or disables (0) time constrained mode when you are using HLA Time Management.

Table QR-3: Keyboard shortcuts

Key	Action	Key	Action	Key	Action
P	Play	Alt + C	Configuration	Alt + K	Skip Time
Ctrl + P	Pause	Alt + S	Set Time Scale	Alt + P	Filter by PDU Kind
S	Stop	Alt + T	Set Time	Alt + E	Filter by Entity ID
B	Beginning of recording.	Alt + V	View Annotations	Alt + Y	Filter by Entity Type
E	End of recording.	Alt + Q	SQL Sender Control	Alt + B	Filter by Class
R	Record	Alt + N	Subscription Control	Alt + L	Filter by Object
L/ R arrow	Move time scale (when it has focus.)	Alt + M	Logger Pipeline Stages Manager		



Link - Simulate - Visualize

LOG-3.10-1-040917