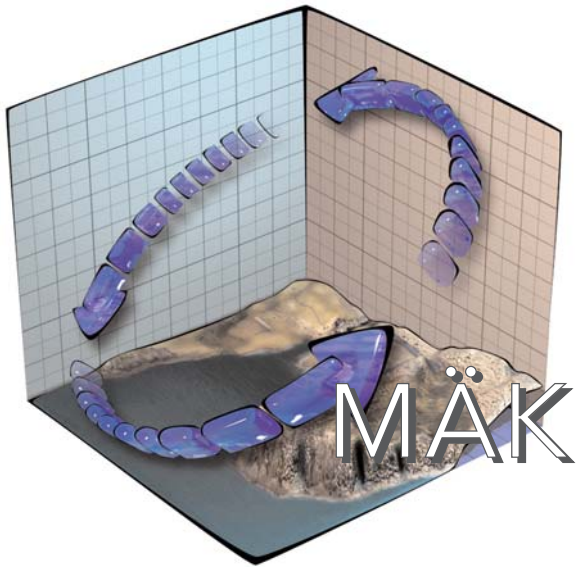
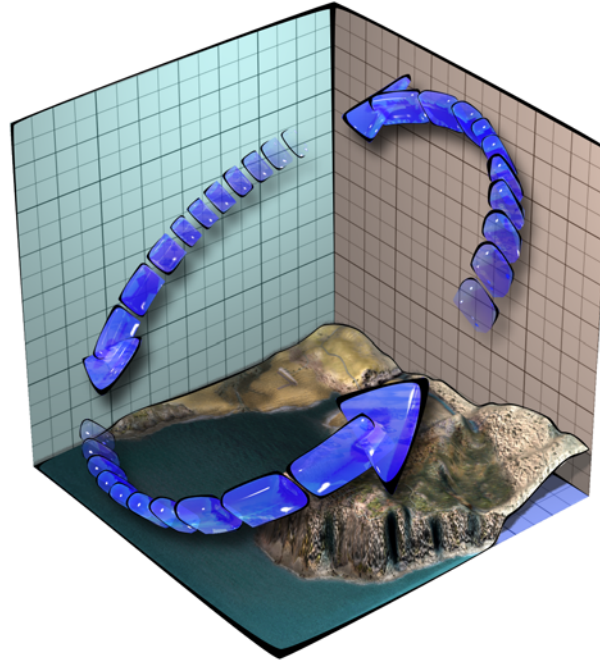




HLA/DIS Toolbox

for MATLAB and Simulink

Developer's Guide



HLA/DIS Toolbox for MATLAB and Simulink



Developer's Guide

Copyright © 2005 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®, VR-Forces®, RTIsPy®, and VR-Link® are registered trademarks of MÄK Technologies, Inc.

MATLAB® and Simulink® are registered trademarks of The Mathworks, Inc.

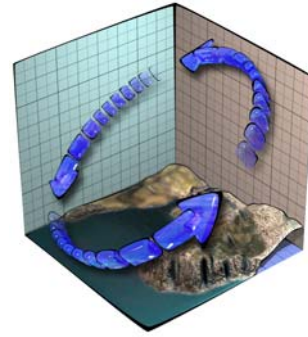
All other trademarks are owned by their respective companies.

MÄK Technologies
10 Fawcett St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revisions TBX-1.0-1-050715



Contents

Preface

How This Manual is Organized	vii
HLA/DIS Toolbox Documentation	viii
MÄK Products	ix
How to Contact Us	xi
Document Conventions	xii
Hardware and Operating System Naming Conventions	xii
Mouse Button Naming Conventions	xiii

Chapter 1 Introduction

1.1. Introduction	1-2
1.2. The HLA/DIS Toolbox and VR-Link	1-3
1.3. The HLA/DIS Toolbox Supports Multiple Protocols	1-4
1.3.1. HLA	1-4
1.3.2. The DIS Protocol	1-5
1.3.3. Obtaining Further Information about HLA and DIS	1-5

Chapter 2 Installing the HLA/DIS Toolbox

2.1. Installing the HLA/DIS Toolbox	2-2
2.1.1. Installing the HLA/DIS Toolbox from a CD	2-2
2.1.2. Installing the HLA/DIS Toolbox from a Downloaded File	2-3
2.2. HLA/DIS Toolbox Files Installed	2-4
2.3. Setting Up and Using the FLEXlm License Manager	2-5
2.3.1. The FLEXlm Client-Server Architecture	2-5
2.3.2. Installing and Configuring the License Manager	2-6
2.3.3. Installing the License Manager Software	2-6
2.3.4. Requesting a License File	2-7
2.3.5. Putting the License File in the <i>flexlm</i> Directory	2-7
2.3.6. Setting the MAKLMGRD_LICENSE_FILE Variable	2-8

2.3.7. Adding Additional License Files	2-9
2.3.8. Running the License Server	2-9
2.3.9. Running FLEXlm as a Service on Windows	2-10
2.3.10. Troubleshooting the FLEXlm License Manager	2-10
2.4. Installing an RTI	2-13
2.4.1. Installing the MÄK RTI	2-13

Chapter 3 HLA/DIS Toolbox Concepts

3.1. VR-Link Concepts	3-2
3.2. Connecting to an Exercise	3-2
3.2.1. Configuring Exercise Connections	3-3
3.3. Managing Entities	3-4
3.4. Managing Interactions	3-4
3.4.1. Using Callbacks	3-5
3.5. Identifying Objects	3-5
3.6. Coordinate Systems	3-5
3.7. Dead-Reckoning and Smoothing	3-6

Chapter 4 Creating Simulations in MATLAB

4.1. Using the HLA/DIS Toolbox with MATLAB	4-3
4.2. HLA/DIS Toolbox Examples	4-3
4.2.1. A Listen-Only Example	4-4
4.2.2. A Send-Only Example	4-7
4.2.3. Running and Testing the Examples and Your Applications	4-9
4.3. Connecting to an Exercise	4-11
4.3.1. Getting the Attributes of an Exercise Connection	4-12
4.3.2. Setting an Exercise Connection's Attributes	4-13
4.3.3. Destroying an Exercise Connection	4-13
4.4. Using Real Time or Simulation Time	4-13
4.5. Draining Input from the Network	4-14
4.6. Publishing Entity Data	4-14
4.6.1. Creating an Entity Publisher	4-14
4.6.2. Getting Entity Publisher Properties and Entity Attributes	4-14
4.6.3. Setting Entity Publisher Properties and Entity Attributes	4-16
4.6.4. Sending Entity Update Messages	4-17
4.6.5. Destroying an Entity Publisher	4-17
4.7. Learning About Remote Entities	4-17
4.7.1. Creating the Reflected Entity List	4-17
4.7.2. Getting Entity State from a Reflected Entity List	4-18
4.7.3. Getting Properties of a Reflected Entity List	4-19
4.7.4. Setting Properties for a Reflected Entity List	4-19
4.7.5. Destroying a Reflected Entity List	4-19
4.8. Sending and Receiving Interactions	4-19
4.8.1. Creating Interactions	4-19
4.8.2. Sending Interactions	4-20

4.8.3. Adding Interaction Callbacks	4-20
4.8.4. Removing Interaction Callbacks	4-20
4.8.5. Using Interaction Translators (HLA Only)	4-21
4.9. Using HLA Time Management	4-23
4.9.1. Instantiating the Time Manager	4-23
4.9.2. Advancing Time	4-23
4.9.3. Getting and Setting Time Manager Properties	4-24
4.9.4. Starting and Stopping Time Regulation	4-24
4.9.5. Starting and Stopping Time Constraint	4-25
4.10. HLA/DIS Toolbox Library Functions	4-26
4.11. Coordinate Systems and Coordinate Conversions	4-27
4.11.1. Geocentric Coordinates	4-27
4.11.2. Geodetic Coordinates	4-28
4.11.3. Topographic Coordinates	4-29
4.11.4. Setting the UTM Reference Point	4-30
4.11.5. Differences Between UTM and Topographic Coordinates ...	4-31
4.11.6. Converting Between Coordinate Systems	4-32

Chapter 5 Using the HLA/DIS Toolbox with Simulink

5.1. Introduction	5-3
5.2. The HLA/DIS Toolbox Library	5-3
5.2.1. Blockset Design	5-4
5.3. Example Applications for Simulink	5-5
5.4. Publishing a Simulink Model	5-5
5.4.1. Create an Exercise Connection	5-6
5.4.2. Add a Simple Entity Publisher	5-6
5.4.3. Add the Entity Model	5-6
5.4.4. Add a Coordinate Conversion Block	5-7
5.4.5. Add a Wall Clock Timer	5-7
5.4.6. Add the Final Connections	5-7
5.5. Listening for Remote Entities in a Simulink Model	5-8
5.5.1. Create an Exercise Connection	5-8
5.5.2. Add a Reflected Entity List	5-9
5.5.3. Add a Reflected Entity Selector	5-9
5.5.4. Add a Simple Reflected Entity	5-10
5.5.5. Add a Wall Clock Timer	5-10
5.6. Blockset Descriptions	5-11
5.6.1. Exercise Connection	5-11
5.6.2. Entity Publisher	5-12
5.6.3. Simple Entity Publisher	5-13
5.6.4. Reflected Entity List	5-14
5.6.5. Reflected Entity	5-14
5.6.6. Simple Reflected Entity	5-15
5.6.7. Reflected Entity Selector	5-16
5.6.8. Interaction Publisher	5-16

Contents

5.6.9. Received Interaction	5-16
5.6.10. Received Interaction List	5-17
5.6.11. Received Interaction Selector	5-17

Chapter 6 Using MATLAB to Analyze Logger Data

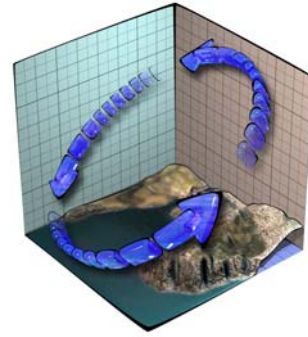
6.1. Importing Logger Data into MATLAB	6-2
6.2. Database Schema Used by the Logger SQL API	6-3
6.2.1. The entities Table	6-3
6.2.2. The aggregates Table	6-4
6.2.3. The aggregate_subordinates Table	6-7
6.2.4. The detonations Table	6-7
6.2.5. The fires Table	6-9

Chapter 7 FOM Agility

7.1. FOM Agility and FOM Mapping	7-2
--	-----

MÄK Products Glossary

Index



Preface

This manual is written for persons who will develop software using the HLA/DIS Toolbox for MATLAB and Simulink. The manual assumes that you are familiar with your operating system, the MATLAB and Simulink development environments, and C++.

Please see release documentation for updates to this manual and the latest information about your version of HLA/DIS Toolbox.

How This Manual is Organized

The chapters are organized as follows:

Chapter 1, *Introduction* introduces you to the HLA/DIS Toolbox.

Chapter 2, *Installing the HLA/DIS Toolbox* explains how to install the HLA/DIS Toolbox, VR-Link, the MÄK RTI, and license management software.

Chapter 3, *HLA/DIS Toolbox Concepts* explains VR-Link concepts for simulations and explains how they are implemented in the HLA/DIS Toolbox.

Chapter 4, *Creating Simulations in MATLAB* explains how to create MATLAB applications using the HLA/DIS Toolbox.

Chapter 5, *Using the HLA/DIS Toolbox with Simulink* explains how to create Simulink models using the HLA/DIS Toolbox.

Chapter 6, *Using MATLAB to Analyze Logger Data* explains how to import simulation data that has been saved to a SQL database by the MÄK Data Logger.

Chapter 7, *FOM Agility* briefly describes FOM agility and FOM Mappers.

The *MÄK Products Glossary* defines common terms used in simulations and MÄK products.

HLA/DIS Toolbox Documentation

The HLA/DIS Toolbox documentation set is as follows:

- ♦ *MÄK HLA/DIS Toolbox for MATLAB and Simulink Developer's Guide* describes how to install and use the HLA/DIS Toolbox. It is available in printed format and as an online manual in Adobe Portable Document Format (PDF).
- ♦ Release documentation consists of release notes provided as a PDF file, text file, and in printed format.
- ♦ Online help viewable in the MATLAB help browser and integrated with Simulink blockset masks.

The PDF files are in the *matlabroot/help/hladistbx.x* directory.

Online Manuals

The online manuals use hypertext links for the entries in the table of contents, for all index entries, and for cross references. These files also support full text searching.

To view and print PDF documents, you need a copy of the Adobe Acrobat Reader software (version 5.0 or later). It is available for free from Adobe. To obtain a copy, visit www.adobe.com.

MÄK Products

The HLA/DIS Toolbox for MATLAB and Simulink is a member of the MÄK Technologies® line of software products designed to streamline the process of developing and using networked simulated environments. The MÄK Technologies product line includes the following:

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

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

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, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface for the MÄK RTI rtiexec.
- ♦ **VR-Forces®.** VR-Forces is a computer generated forces application and toolkit. It provides an application with a graphical user interface, that gives you a two dimensional view of a terrain database.

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

- ♦ **MÄK Stealth and StealthXR.** The Stealth displays a realistic, three-dimensional view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway.

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

- ♦ **MÄK Data Logger.** The Data Logger, also called the Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal, or in reverse, and locate areas of interest quickly. The Logger has a graphical user interface and a command line interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application.

The Logger comes with a set of utility programs that let you inspect, combine, and edit Logger recordings.

- ♦ **MÄK Plan View Display.** The Plan View Display (PVD) provides a two-dimensional, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display.
- ♦ **MÄK Gateway.** The Gateway is a translator application that allows DIS exercises to operate as members of HLA federation executions that use the RPR FOM, thereby preserving the value of simulation code written to support DIS. The MÄK Gateway is a standalone application and does not require you to write additional code for its use.
- ♦ **MÄK HLA/DIS Toolbox for MATLAB and Simulink.** The HLA/DIS Toolbox is a wrapper around VR-Link that allows MATLAB and Simulink users to analyze data from live or recorded HLA or DIS exercises. It also allows Simulink users to build models that can participate in HLA or DIS exercises.
- ♦ **MÄK Game-Link** is a toolkit that allows games developed using Epic Games’ Unreal Engine to interoperate with DIS and HLA simulations and trainers.

How to Contact Us

For HLA/DIS Toolbox 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.
	Indicates supplemental or clarifying information.
	Indicates additional information that you must observe to ensure the success of a procedure or other task.

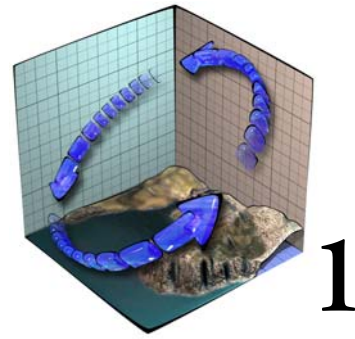
Directory names are preceded with dot and slash characters that show their position with respect to the HLA/DIS Toolbox home directory. For example, the directory */doc* appears in the text as *./doc*.

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 explains what the HLA/DIS Toolbox does and provides some general background information.

Introduction	1-2
The HLA/DIS Toolbox and VR-Link	1-3
The HLA/DIS Toolbox Supports Multiple Protocols	1-4
HLA	1-4
The DIS Protocol	1-5
Obtaining Further Information about HLA and DIS	1-5

1.1. Introduction

The HLA/DIS Toolbox provides an interface between MATLAB and Simulink and distributed simulation environments. It allows you to import live or recorded HLA or DIS data into MATLAB for analysis, or integrate MATLAB models into an HLA or DIS environment and it also allows Simulink models to communicate with HLA or DIS applications.

The HLA/DIS Toolbox is a wrapper around VR-Link that allows a MATLAB application or a Simulink model to act as a full-fledged HLA federate or to participate in a DIS exercise. It provides an interface (via M-code classes and a Simulink library) to common VR-Link functionality, so that you can easily join a federation, subscribe to various object and interaction classes, and receive data through the HLA RTI, or from the DIS network.

You can connect the outputs of your Simulink models (location, velocity, and so on) to new Simulink blocks that publish your object on the network. On the receiving side, new Simulink blocks generate outputs based on updates received from the network. Your MATLAB or Simulink models become natively HLA or DIS compliant, without writing any C++ code.

Here are a few examples of what you can do with the HLA/DIS Toolbox:

- ♦ A UAV simulation written in C++ might want to use a high-fidelity sensor model that is written in MATLAB. Using the HLA/DIS Toolbox, your sensor model can respond to HLA queries from the UAV simulation, by providing information about which targets are visible.
- ♦ A surface-to-air missile (SAM) simulation might be written within MATLAB, and you may want to test your control system against a target aircraft being generated by an external man-in-the-loop flight simulator. The aircraft publishes its state information through HLA, and it is received and made available to the SAM simulator through the HLA/DIS Toolbox. The SAM flyout is performed within MATLAB, and data is published through HLA/DIS, so that the pilot can see it approaching on his screen in the flight simulator. The SAM simulation calculates that the missile has hit the target, and publishes an HLA detonation interaction using the HLA/DIS Toolbox. The flight simulator receives the interaction, and tells the pilot that he has been hit.
- ♦ A tank simulation is constructed by connecting various components in the Simulink model development environment (for example, path planning model, ground dynamics model, damage model, and so on). To spread the computational load, you want each component to run on a separate machine. The HLA/DIS Toolbox Library for Simulink allows you to connect these various Simulink components using HLA or DIS. In addition letting applications communicate among the components, the use of a standard protocol like HLA and DIS allows you to log the data using existing third party tools.

1.2. The HLA/DIS Toolbox and VR-Link

As noted previously, the HLA/DIS Toolbox is a wrapper around VR-Link. VR-Link has a protocol-independent API that lets programmers create simulation applications largely without regard to the specifics of HLA or DIS. The HLA/DIS Toolbox abstracts from these simulation standards even further. With the exception of some protocol-specific initialization data and some features supported only by HLA, any M-code that you write with the HLA/DIS Toolbox can easily be run using any of the supported protocols. [Figure 1-1](#) illustrates the interaction of MATLAB, the HLA/DIS Toolbox, and VR-Link.

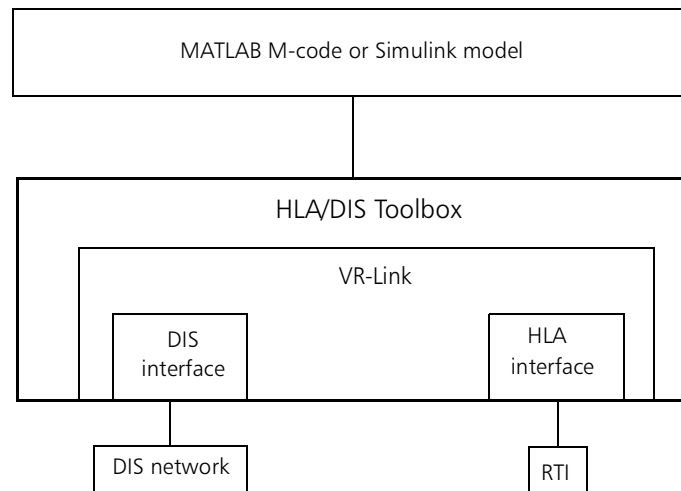


Figure 1-1. Structure of a typical HLA/DIS Toolbox application

The Simulink blocksets are protocol-specific. However, their components use identical naming conventions and any model that you create using the DIS blockset, for example, can easily be replicated using the same components in either of the HLA blocksets.

1.3. The HLA/DIS Toolbox Supports Multiple Protocols

The HLA/DIS Toolbox supports the HLA 1.3 and 1516 specifications and DIS. Please refer to your release documentation for a list of the versions of these simulation standards supported by your release of the HLA/DIS Toolbox.

It is beyond the scope of this manual to fully describe DIS and HLA. However, in this section we briefly describe them as a foundation for the concepts and terms discussed in the rest of this manual. You may also want to consult the [MÄK Products Glossary](#).

1.3.1. HLA

HLA is required by the Department of Defense for use in all DoD simulations. In HLA, the types and formats of data to be exchanged among simulators are not standardized, as they are in DIS. Groups of simulators that want to play together must use the same Federation Object Model (FOM) and Run-Time Infrastructure (RTI). FOMs and RTIs are described in the next two sections.

The Federation Object Model (FOM)

Every federation execution requires a Federation Object Model (FOM), which defines the data model for the federation execution. Members of a federation execution can develop a FOM themselves, use an existing reference FOM, or build a FOM by starting with a reference FOM and modifying it.

A FOM is implemented through a Federation Execution Data (FED) file, which contains a subset of the FOM that is required by the RTI. The HLA/DIS Toolbox has built-in support for the Realtime-Platform Reference FOM (RPR FOM). More specifically, the HLA/DIS Toolbox supports the FOM defined within the FED file called *VR-Link.fed*, which is in the HLA/DIS Toolbox home directory (*matlabroot/toolbox/hladistbx.x*). *VR-Link.fed* represents the RPR FOM, plus a few extra classes that MÄK has added.

The RPR FOM is a reference FOM developed by the SISO-sanctioned RPR FOM Standards Development Group, which consists of representatives from many companies that used the DIS protocol in the pre-HLA era.

The goal of the RPR FOM is to facilitate a priori interoperability (to about the extent available in DIS) among HLA simulations that choose to use it. In other words, if you use the RPR FOM (either as is, or with your own extensions), then you know that you will be able to interoperate with anyone else who chooses to use this FOM. An advantage of using the RPR FOM is that many simulation tools, including most MÄK Technologies products, support this FOM.

The Run-Time Infrastructure (RTI)

An RTI is an implementation of the HLA interface specification, that is, the services that make up the standard RTI application programmer's interface (API). Applications written with the HLA/DIS Toolbox access RTI function calls automatically through the toolbox. The HLA rules specify that federates must use the RTI to exchange all simulation data.



VR-Link, and therefore, the HLA/DIS Toolbox, supports the RTI 1.3 specification and the SISO Standard DLC C++ API for IEEE 1516 specification, an implementation of the IEEE 1516 specification that supports link compatibility. Therefore, you can run HLA/DIS Toolbox applications with RTIs that support one of these RTI specifications. However, the HLA/DIS Toolbox only supports RTIs compiled using Microsoft Visual C++ 7.1.

1.3.2. The DIS Protocol

The DIS protocol is a set of standards that govern how participating applications share information about a 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.



DIS 2.0.4 and IEEE 1278.1-1995 are equivalent, as are DIS 2.1.4 and IEEE 1278.1a-1998.

1.3.3. Obtaining Further Information about HLA and DIS

For information about HLA rules and specifications, visit the HLA Web site at <http://www.dmsomil/public/transition/hla/>

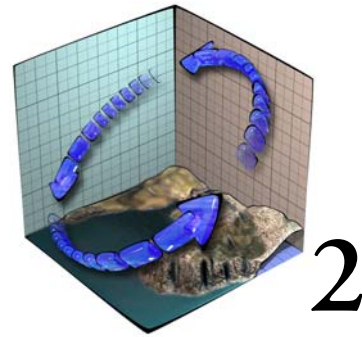
You can find general DIS information in the document library at the Simulation Interoperability Standards Organization (SISO) home page <http://www.sisostds.org/doclib/>

The IEEE documentation of IEEE 1278.1-1995 and 1278.1a-1998 has superseded documentation of DIS 2.0.4 and DIS 2.1.4. For documentation of the IEEE 1278.1-1995 or 1278.1a-1998 versions of DIS, you can obtain the publications, *IEEE Standard for Distributed Interactive Simulation, Application Protocols and Enumeration and Bit Encoded Values for Use with Protocols for DIS Applications* at the IEEE World Wide Web site: <http://shop.ieee.org/store/>

Enumeration and Bit Encoded Values for Use with Protocols for DIS Applications (DIS enumerations document) is also available at the SISO site.



The URLs listed in this section were accurate at the time the manual was released. We cannot guarantee that they will be maintained in the future.



Installing the HLA/DIS Toolbox

This chapter explains how to install the HLA/DIS Toolbox, license management software, and the MÄK RTI.

Installing the HLA/DIS Toolbox.....	2-2
Installing the HLA/DIS Toolbox from a CD	2-2
Installing the HLA/DIS Toolbox from a Downloaded File	2-3
HLA/DIS Toolbox Files Installed.....	2-4
Setting Up and Using the FLEXlm License Manager	2-5
The FLEXlm Client-Server Architecture	2-5
Installing and Configuring the License Manager.....	2-6
Installing the License Manager Software	2-6
Requesting a License File.....	2-7
Putting the License File in the flexlm Directory.....	2-7
Setting the MAKLMGRD_LICENSE_FILE Variable.....	2-8
Adding Additional License Files.....	2-9
Running the License Server	2-9
Running FLEXlm as a Service on Windows	2-10
Troubleshooting the FLEXlm License Manager	2-10
Installing an RTI.....	2-13
Installing the MÄK RTI	2-13

2.1. Installing the HLA/DIS Toolbox

This section explains how to install the HLA/DIS Toolbox from CD-ROM or from an ftp download.

The HLA/DIS Toolbox CD contains the installers for the HLA/DIS Toolbox, VR-Link, and the MÄK RTI. If you plan to create simulations for use with HLA, you must install an RTI that is link compatible with the HLA 1.3 specification or the SISO Standard DLC C++ API for IEEE 1516 specification. The MÄK RTI is compatible with both of these specifications. Since the HLA/DIS Toolbox is essentially an interface to VR-Link, you may want to install VR-Link to take advantage of its supporting utilities and to consult the documentation. If you want to write a FOM Mapper, you must install VR-Link (see Chapter 7, [FOM Agility](#)).



The MÄK RTI is licensed separately from the HLA/DIS Toolbox. An HLA/DIS Toolbox license includes a license to use VR-Link through the HLA/DIS Toolbox. It does not allow you to build VR-Link applications independently of the HLA/DIS Toolbox.

Before you install the HLA/DIS Toolbox, read the *Release Notes* to see if there are any special instructions for installation.

2.1.1. Installing the HLA/DIS Toolbox from a CD

The HLA/DIS Toolbox installer is designed to install on top of an existing installation of MATLAB. It does not modify existing files.



You will need write access to the MATLAB directory to install.

The HLA/DIS Toolbox installation wizard asks you if you want to install Flexlm 9.2 (license management software - see “[Setting Up and Using the FLEXlm License Manager](#),” on page 2-5) and run the MATLAB install script.

To add the necessary paths to MATLAB, the HLA/DIS Toolbox ships a MATLAB script called MAK_Install that automatically adds the directories to the permanent MATLAB path. The installer will run MATLAB with a startup parameter to run the MAK_Install script then quit.



This MAK_Install script will fail if the installation is not done on top of MATLAB.

To install the HLA/DIS Toolbox:

1. Insert the HLA/DIS Toolbox CD into your CD-ROM drive. If 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.
2. In the MÄK PC Products Installation window, click HLA/DIS Toolbox.
3. Follow the instructions of the installation program, noting the following.
 - When you select the installation directory, choose the root MATLAB directory, for example, *C:\MATLAB701*.
 - If you already have FLEXlm installed for other MÄK products, you do not need to install it again for the HLA/DIS Toolbox.

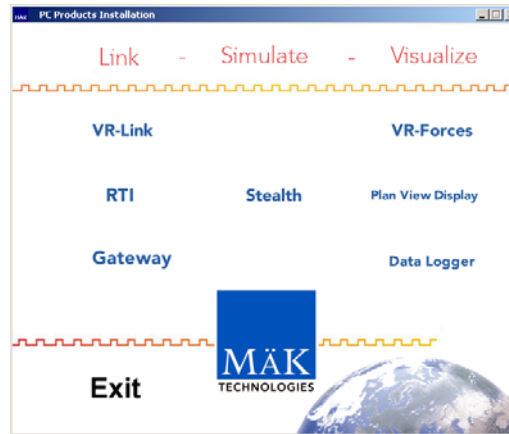


Figure 2-1. MÄK PC Products Installation window

4. To install VR-Link or the MÄK RTI, click its icon on the PC Products Installation window and follow the prompts..

2.1.2. Installing the HLA/DIS Toolbox from a Downloaded File

If you obtained the HLA/DIS Toolbox via ftp, you received a compressed file. It may be a zip file or an executable file.

To install the HLA/DIS Toolbox 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.

2.2. HLA/DIS Toolbox Files Installed

The default installation process installs the HLA/DIS Toolbox in the MATLAB directory structure. [Figure 2-2](#) illustrates the directory structure.

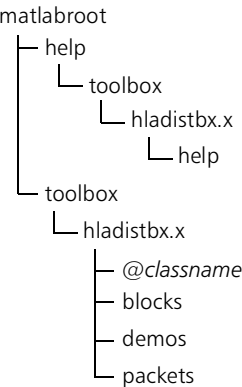


Figure 2-2. HLA/DIS Toolbox directory structure

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

Table 2-1: Contents of HLA/DIS Toolbox directories

Directory	Directory contents
<i>matlabroot/help/toolbox/hladistbx.x</i>	HLA/DIS Toolbox documentation in PDF format and online help.
<i>matlabroot/toolbox/hladistbx.x/@...</i>	M-code class files
<i>matlabroot/toolbox/hladistbx.x/blocks</i>	HLA/DIS Toolbox libraries for Simulink blocksets.
<i>matlabroot/toolbox/hladistbx.x/demos</i>	Source files for all example applications and models.
<i>matlabroot/toolbox/hladistbx.x/packets</i>	DIS freshening header file.

2.3. Setting Up and Using the FLEXlm License Manager

MÄK products use the FLEXlm license manager. Before you can use a MÄK product, you must obtain a valid license file and configure the license server and client machines. (For general information about FLEXlm, see <http://www.macrovision.com/products/flexlm/index.shtml>.) Contact the MÄK Technologies sales department for information about special licensing agreements.

2.3.1. The FLEXlm Client-Server Architecture

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

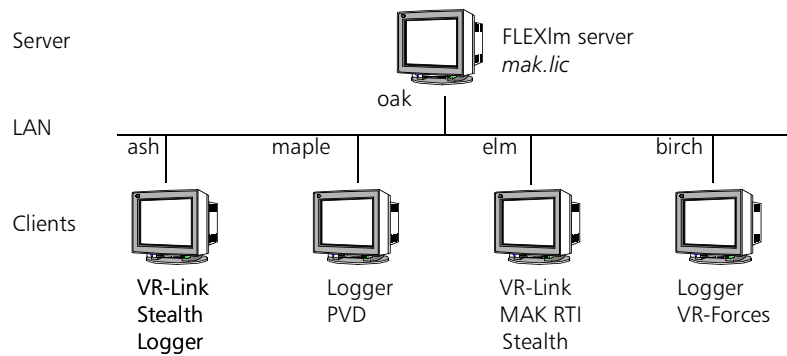
The license server executes vendor daemons (for MÄK Technologies, *maklmgrd*) and uses one or more license files, provided by individual vendors. The license file identifies the products for which you have licenses. On each client machine, the `MAKLMGRD_LICENSE_FILE` environment variable (which you must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

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



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

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



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

Figure 2-3. FLEXlm example architecture

2.3.2. Installing and Configuring the License Manager

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

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

If you have MÄK Technologies products installed and you purchase additional licenses or products, you must get a license file for the new products or licenses. For this procedure, see [“Adding Additional License Files,”](#) on page 2-9.

2.3.3. Installing the License Manager Software

As part of the installation of the HLA/DIS Toolbox, the FLEXlm software is installed in *C:\MAK\flexlm92*.

1. Install your MÄK products on the machines on which you want to run them.
2. Copy the files in *flexlm92* to *C:\flexlm* on the server machine.



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/s1ss10p1.php> and complete the license key request form. MÄK will e-mail you a license file.

Identifying the Host ID and License Server Name in Windows

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

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

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 2-9.

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.

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 2-7.

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 *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. Check the Task Manager. (To open the Task Manager, press Ctrl-Alt-Delete and click Task Manager.)
2. If there are old processes present that may be using a license, kill them. 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 an RTI other than the MÄK RTI and you get the unsupported product message, it is possible that your application is finding the DLLs for an unlicensed version of the MÄK RTI before it finds the other RTI. To fix this problem do the following:

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

2.4. Installing an RTI

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

i

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

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

For the most recent information about the RTI versions supported by MÄK products, see the release notes for your MÄK application or the Product Versions page on the MÄK Technologies web site at: http://www.mak.com/product_version.php.

2.4.1. Installing the MÄK RTI

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

Configuring Your System to Use the MÄK RTI

The RTI dynamic libraries need to be located somewhere on your dynamic library search path. The path is specified in the PATH environment variable.

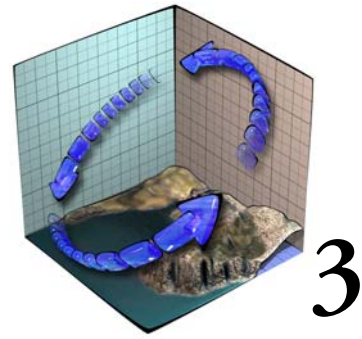
The MÄK RTI needs to know where to find the following configuration files:

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

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

Running Applications with the MÄK RTI

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



HLA/DIS Toolbox Concepts

This chapter explains concepts that apply to use of the HLA/DIS Toolbox with both MATLAB and Simulink.

VR-Link Concepts.....	3-2
Connecting to an Exercise.....	3-2
Configuring Exercise Connections.....	3-3
Managing Entities.....	3-4
Managing Interactions	3-4
Using Callbacks.....	3-5
Identifying Objects	3-5
Coordinate Systems.....	3-5
Dead-Reckoning and Smoothing	3-6

3.1. VR-Link Concepts

Since the HLA/DIS Toolbox is a wrapper around VR-Link, its classes and objects mirror the basic VR-Link classes and implement VR-Link concepts. A VR-Link application performs several tasks in order to interact with other players in a virtual world. These tasks typically include:

- ♦ Connecting to an exercise¹. The application must create an object through which it can send and receive simulation data. That object should provide an efficient way to direct incoming data to other parts of the application. VR-Link uses an exercise connection to connect to an exercise.
- ♦ Managing state information of local entities. The application must inform other exercise participants about the states of its locally generated objects, such as entities, aggregates, and so on. It does this using entity publishers.
- ♦ Managing state information of remote entities. The application typically must receive, process, and manage state information about remote objects. VR-Link uses a reflected entity list to track the participants in an exercise, which are known as reflected entities.
- ♦ Managing information about events. The application must inform other participants about interactions such as fire events or detonations involving a locally simulated entity, and receive and process events and interactions sent by other applications. The HLA/DIS Toolbox has built-in support for fire and detonate interactions.

For a detailed discussion of VR-Link concepts, see *VR-Link Developer's Guide*.

3.2. Connecting to an Exercise

HLA/DIS Toolbox applications connect to an exercise through an exercise connection. The HLA/DIS Toolbox uses the exercise connection to send and receive state information and other data. The exercise connection allows an application to do the following:

- ♦ Send interactions to the exercise
- ♦ Read (drain) input from the network
- ♦ Generate event IDs
- ♦ In M-code, register callback functions that are executed each time the HLA/DIS Toolbox receives interactions from the network
- ♦ Manage the simulation clock.

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

3.2.1. Configuring Exercise Connections

An exercise connection takes different input parameters depending on whether it is for a DIS exercise or HLA federation. This section provides general descriptions of them. See the descriptions of exercise connections in Chapter 4, [Creating Simulations in MATLAB](#) and Chapter 5, [Using the HLA/DIS Toolbox with Simulink](#) for the specific syntax for specifying the parameters.

The parameters for HLA are:

- ♦ Exercise name – Specifies the federation execution.
- ♦ Federate name – Specifies the name of this federate.
- ♦ FED file – Specifies the FED file. If the FED file is not specified, it is assumed to be *exerciseName.fed*. For example, if the exercise name is VR-Link, the FED file is assumed to be *VR-Link.fed*. By default, VR-Link looks for the FED file in the directory from which you are running an application. Then it looks in the directory specified by the RTI_CONFIG environment variable. We recommend that you set this environment variable or specify the path of the FED file when you configure an exercise connection.
- ♦ FOM Version – Specifies the version of the RPR FOM being used. The HLA/DIS Toolbox has built-in support for the following versions of the RPR FOM:
 - RPR FOM 0.5
 - RPR FOM 0.7
 - RPR FOM 0.8
 - RPR FOM 1.0
 - RPR FOM 2.0 draft 6
 - RPR FOM 2.0 draft 14
 - RPR FOM 2.0 draft 17.

Default: RPR FOM 1.0.

If you specify a RPR FOM version other than the default, you must also specify the appropriate FED file. The names of the FED files provided with the HLA/DIS Toolbox correspond to the RPR FOM versions. For example, the FED file for RPR FOM 2.0, draft 17 is *VR-Link20017.fed*. The FED files are installed in *matlabroot/toolbox/hladistbx.x*.

- ♦ FOM Mapper – Specifies the name of a VR-Link FOM Mapper DLL. The FOM Mapper must be built with the same compiler as the HLA/DIS Toolbox. For information about FOM Mappers, see Chapter 7, [FOM Agility](#).

The parameters for DIS are:

- ♦ Port – Specifies the UDP port for DIS traffic.
- ♦ Exercise ID – Specifies the DIS exercise ID.
- ♦ Site ID – Specifies the DIS site ID.
- ♦ Application number – Specifies a unique identifier for the DIS simulation.

3.3. Managing Entities

The HLA/DIS Toolbox uses an entity publisher to send entity state updates to the network. It manages remote entities by maintaining a reflected entity list. In your application, you set entity attributes in a MATLAB structure. The entity publisher sends them to the network through the exercise connection.

The HLA/DIS Toolbox maintains a list of remote (reflected) entities. Applications can query the list to get information about specific remote entities.

Figure 3-1 illustrates how entities are managed.

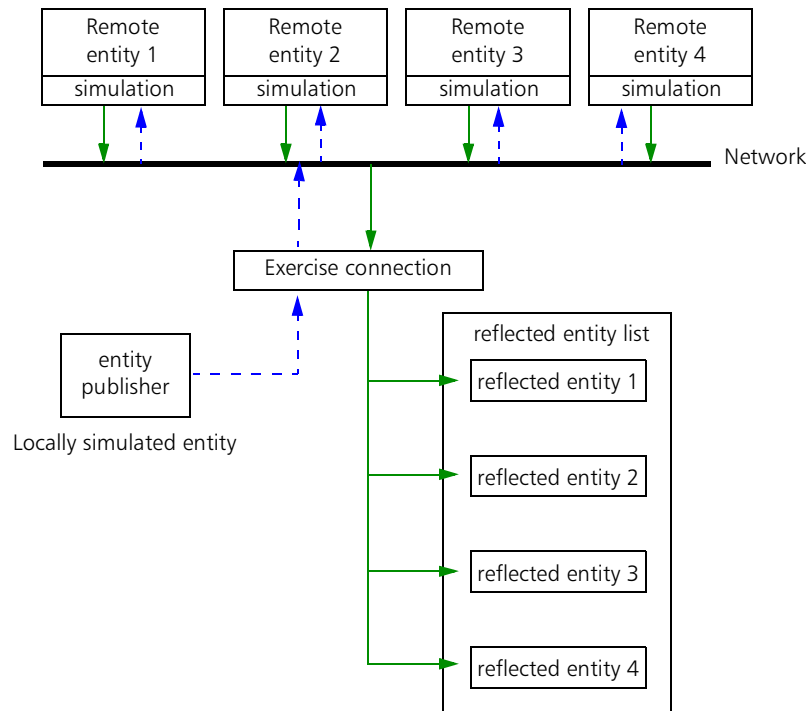


Figure 3-1. Managing state information

3.4. Managing Interactions

The HLA/DIS Toolbox can generate interactions and send them to the network through the exercise connection. In M-code, it uses the `generateInteraction()` function to create interactions. In Simulink, it uses an Interaction Publisher block.

Most simulation applications must be able to respond to interactions generated by remote entities. The HLA/DIS Toolbox handles interactions in MATLAB through the use of callbacks. It handles them in Simulink with interaction lists. The Simulink model iterates through the list of interactions and responds to them appropriately.

3.4.1. Using Callbacks

Callback functions are functions that you write that enable your application to respond to events. You register a callback with the exercise connection. Then the application calls your function in response to a specified event.

For MATLAB applications, the HLA/DIS Toolbox uses callbacks to process incoming interactions. This is accomplished using *DtExerciseConn*'s `addInteractionCallback()` function. Subsequently, whenever the application reads that type of interaction from the network, it calls your callback function, passing it a copy of the relevant interaction. The callback gets called from within `drainInput()` – the HLA/DIS Toolbox function that causes input to be read from the exercise.

3.5. Identifying Objects

One concept that HLA and DIS handle differently is object identification.

DIS only

In DIS, entities are identified by a triplet (site, application, entity) known as an Entity Identifier. Other types of objects are usually identified using the IDs of their host entities plus a single additional ID number.

HLA only

In HLA, objects have several different identifiers, including:

- An Object Handle, which is used within a federate to identify a particular object in an RTI service invocation
- An Object Name, which is used within interactions and attribute updates exchanged among federates for the RPR FOM.

3.6. Coordinate Systems

By default, entity location is represented with geocentric coordinates, as specified by DIS and the RPR FOM. The HLA/DIS Toolbox has coordinate conversion routines that let you convert incoming and outgoing entity information to other coordinates systems. See [“Coordinate Systems and Coordinate Conversions,”](#) on page 4-27 for details about coordinate systems and coordinate conversions.

3.7. Dead-Reckoning and Smoothing

To represent the behavior of entities in between state updates, the HLA/DIS Toolbox uses dead-reckoning, a process of estimating the location of an entity based on its acceleration and velocity. When updated location information arrives, the HLA/DIS Toolbox must move the simulated entity to its actual position. To ensure that transitions from an entity's dead-reckoned position to its actual position are not so abrupt as to be visually disconcerting, the HLA/DIS Toolbox implements smoothing. [Figure 3-2](#) illustrates the difference between a smoothed transition and a non-smoothed transition.

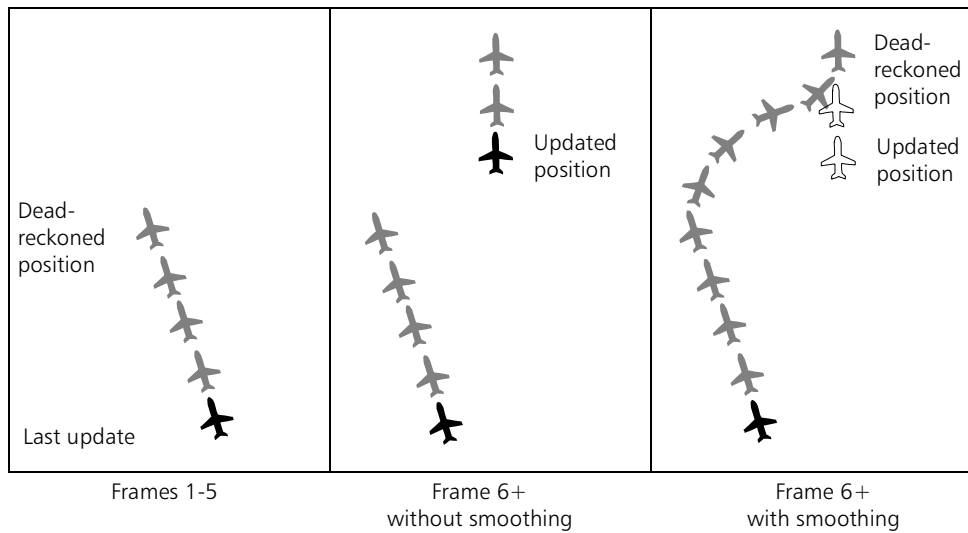
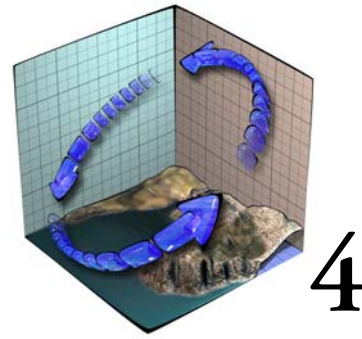


Figure 3-2. Dead-reckoning and smoothing



Creating Simulations in MATLAB

This chapter explains how to create and manage simulations using M-code.

Using the HLA/DIS Toolbox with MATLAB	4-3
HLA/DIS Toolbox Examples	4-3
A Listen-Only Example	4-4
A Send-Only Example	4-7
Running and Testing the Examples and Your Applications	4-9
Connecting to an Exercise.....	4-11
Getting the Attributes of an Exercise Connection	4-12
Setting an Exercise Connection's Attributes	4-13
Destroying an Exercise Connection	4-13
Using Real Time or Simulation Time	4-13
Draining Input from the Network	4-14
Publishing Entity Data.....	4-14
Creating an Entity Publisher.....	4-14
Getting Entity Publisher Properties and Entity Attributes	4-14
Setting Entity Publisher Properties and Entity Attributes.....	4-16
Sending Entity Update Messages	4-17
Destroying an Entity Publisher.....	4-17
Learning About Remote Entities	4-17
Creating the Reflected Entity List.....	4-17
Getting Entity State from a Reflected Entity List.....	4-18
Getting Properties of a Reflected Entity List.....	4-19
Setting Properties for a Reflected Entity List.....	4-19
Destroying a Reflected Entity List	4-19
Sending and Receiving Interactions.....	4-19

Creating Interactions.....	4-19
Sending Interactions.....	4-20
Adding Interaction Callbacks	4-20
Using Interaction Translators (HLA Only)	4-21
Using HLA Time Management	4-23
Instantiating the Time Manager	4-23
Advancing Time	4-23
Getting and Setting Time Manager Properties.....	4-24
Starting and Stopping Time Regulation	4-24
Starting and Stopping Time Constraint.....	4-25
HLA/DIS Toolbox Library Functions	4-26
Coordinate Systems and Coordinate Conversions	4-27
Geocentric Coordinates.....	4-27
Geodetic Coordinates.....	4-28
Topographic Coordinates	4-29
Setting the UTM Reference Point	4-30
Differences Between UTM and Topographic Coordinates.....	4-31

4.1. Using the HLA/DIS Toolbox with MATLAB

The HLA/DIS Toolbox provides an M-code interface to VR-Link. You use it to send entity state updates and interactions to the network and to receive updates and interactions from the network. Figure 4-1 illustrates the relationship between user M-code, the HLA/DIS Toolbox, and VR-Link.

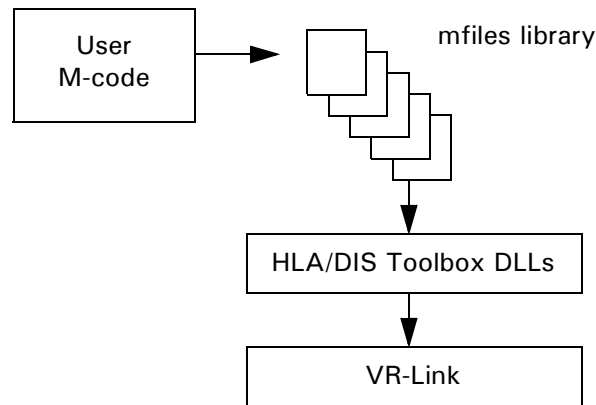


Figure 4-1. Relationship of MATLAB and VR-Link

4.2. HLA/DIS Toolbox Examples

The HLA/DIS Toolbox includes examples (in `matlabroot\toolbox\hladistbx.x\demos`) that demonstrate how to use the HLA/DIS Toolbox with MATLAB. The M-code files provided are:

- ♦ *Berkeley3D.m* – loads the Berkeley terrain and the *MAK_SimpleStealth.m* example.
- ♦ *berkeleydemo.m* – runs the Berkeley database demonstration.
- ♦ *defineWeaponFire.m* – demonstrates how to write interaction translators. This example is a translator for a WeaponFire RPR FOM 1.0 interaction.
- ♦ *DtNullOp.m* – a function used for timing purposes.
- ♦ *f18DBdemo.m* – an example of displaying data from a database.
- ♦ *listen.m* – creates a reflected entity list and listens to the network. This application is launched by *MAK_LISTEN.m* or *MAK_TALK_AND_LISTEN.m* and can run simultaneously with *talk.m*. By default, it only runs for 10 seconds.
- ♦ *loadMap.m* – loads a simple 3D map
- ♦ *MAK_DB.m* – runs the database examples, *f18DBdemo.m* and *Berkeleydemo.m*.
- ♦ *MAK_LISTEN.m* – runs the *listen.m* example.
- ♦ *MAK_MCodeLauncher.m* – a function for launching iterative call applications, such as *talk.m* and *listen.m*.
- ♦ *MAK_SimpleStealth.m* – runs a simple 3D simulation view in the HLA 1.3 protocol.

- *MAK_TALK.m* – runs the *talk.m* example.
- *MAK_TALK_AND_LISTEN.m* – runs *talk.m* and *listen.m* together.
- *MAK_VIEW3D.m* – If you have the MÄK Data Logger, you can run the *berkeley-demo.lgr* file and *MAK_VIEW3D.m* and see the output in a 3D view in MATLAB.
- *simpleListen.m* – creates a reflected entity list and listens to the network.
- *simpleTalk.m* – sends update messages for an entity.
- *talk.m* – sends update messages for an entity. This application is launched by *MAK_TALK.m* or *MAK_TALK_AND_LISTEN.m* and can run simultaneously with *listen.m*. By default, it runs for 10 seconds.
- *talkTM.m* – same as *talk.m*, but uses time management.

The demos directory also includes a file, *berkeleydemotape.exe*, that runs a MÄK Logger recording that you can use as input for your simulations. The version of the MÄK Logger included in this demo cannot be used to record simulations or play other Logger recordings.

This section lists and describes two simple HLA/DIS Toolbox applications to demonstrate how to send and receive entity data. The first is a listen-only application (*simpleListen.m*) that observes an exercise without simulating any entities. The second is a write-only program (*simpleTalk.m*) that simulates an entity and sends update message to the network.

4.2.1. A Listen-Only Example

Example 4.1 illustrates a simple, listen-only application (*simpleListen.m*). This application can connect to a DIS or HLA exercise. The only protocol-specific code is contained in creation of the exercise connection.

With each iteration of the loop, the program prints a reflected entity's updated, dead-reckoned position in topographic coordinates, based on information received from the network. In addition, if a fire PDU or interaction is detected on the network, the program prints a message showing the entity ID of the attacker.



The example code in this manual may be formatted somewhat differently than the actual code.

Example 4.1

```

1  function simpleListen(protocol,time)
2      % Define a callback to process fire interactions.
3      function fireCB(exerciseConn,interaction)
4          disp(['Connection : ' num2str(get(exerciseConn,'handle')) ...
5              ' Received Fire Interaction from : ' interaction.attackerId]);
6      end
7  try
8      %connection parameters
9      execName      = 'VR-Link';

```

```

10     fedName      = 'Matlab listen';
11     port         = 3000;
12     exerciseId   = 1;
13     siteId       = 1;
14     applicationNum = 16;
15     % Create a connection to the exercise or federation execution.
16     if ~isempty(strfind(protocol,'HLA'))
17         exConn = DtExerciseConn(protocol,execName,fedName);
18     elseif strcmp(protocol,'DIS')
19         exConn = DtExerciseConn(protocol,
20             port,exerciseId,siteId,applicationNum);
21     end
22     % Register a callback to handle fire interactions.
23     cb = addInteractionCallback(exConn,'FireInteraction',@fireCB);
24     % Create an object to manage entities that we hear about
25     % on the network.
26     entityList = DtReflectedEntityList(exConn,true,false);
27     simTime = 0.0;
28     while simTime <= time
29         % Tell VR-Link the current value of simulation time.
30         simTime = get(exConn,'elapsedRealTime');
31         set(exConn,'simTime',simTime);
32         simTime = get(exConn,'elapsedRealTime');
33         drainInput(exConn);
34         % Get first entity
35         reflectedEntity = getEntityByIndex(entityList,0);
36         if ~isempty(reflectedEntity)
37             loc = reflectedEntity.Location;
38             vel = reflectedEntity.Velocity;
39             acc = reflectedEntity.Acceleration;
40             disp(['Position: ' num2str(loc(1)) ' ' num2str(loc(2)) ...
41                 ' ' num2str(loc(3))]);
42             disp(['Velocity: ' num2str(vel(1)) ' ' num2str(vel(2)) ...
43                 ' ' num2str(vel(3))]);
44             disp(['Accelera: ' num2str(acc(1)) ' ' num2str(acc(2)) ...
45                 ' ' num2str(acc(3))]);
46         end
47         display(['Current Sim Time: ' num2str(simTime)]);
48         % Wait a bit
49         pause(0.1);
50     end
51     %clean up entity memory
52     destroy(entityList);
53     %clean up simulation memory
54     removeInteractionCallback(exConn,cb);
55     disconnect(exConn);
56     destroy(exConn);
57 catch
58     %clean up entity memory
59     if exist('entityList','var')
60         destroy(entityList);
61     end
62     %clean up simulation memory
63     if exist('exConn','var')
64         if exist('cb','var')
65             removeInteractionCallback(exConn,cb);
66         end
67         disconnect(exConn);
68         destroy(exConn);
69     end

```

```
69     error(lasterr);  
70 end  
71 end
```

Connecting to An Exercise

In lines 15 - 20, the program creates an exercise connection using the `DtExerciseConn()` function. This connection serves as the program's interface to an exercise. A different constructor is used depending on whether the program is for use in HLA or DIS. If you know that you will only be using one of these simulation standards, you can eliminate the alternate constructor in your code.

If you connect to an HLA federation, the constructor takes a federation execution and federate name. If you connect to a DIS exercise, the constructor takes a port number and the three-part exercise:site:application ID. The variables are defined in lines 8 - 14. This is the only protocol-specific part of the example.

Managing State and Interaction Information

HLA/DIS Toolbox applications use callbacks to handle incoming interactions. The example defines a callback called `fireCb` in lines 2 - 6. The callback is registered with the exercise connection in line 22, using the `addInteractionCallback()` function. This callback prints a message containing the attacker ID. It executes whenever the exercise connection receives a Fire PDU or interaction during a call to `drainInput()`.

Tracking Entities

We create a reflected entity list in line 25 to keep track of entities found on the network. The entity list tracks the arrival and departure of entities, performs dead reckoning, manages time-outs, and performs other entity-tracking tasks.

Managing Time

On line 26 we initialize the simulation time. In successive iterations of the loop, we update the simulation time (lines 65 - 67).

Listening to the Network

The `drainInput()` call (line 53) reads and processes any messages arriving through the exercise connection. This call triggers the execution, if needed, of any callbacks you have registered for that exercise connection.

In line 34, the program finds the first entity in the entity list. In lines 36 - 38 it retrieves its position, velocity, and acceleration data. Then it prints the information to the display (lines 39 - 44).

Cleanup

The rest of the example destroys the reflected entity list, removes the interaction callback, and destroys the exercise connection.

4.2.2. A Send-Only Example

Example 4.2 illustrates a simple send-only application (*simpleTalk.m*) that simulates the flight of an F16 aircraft. The program begins by sending a fire PDU or HLA fire interaction. Thereafter, the F16 flies north for 10 seconds, updating its position by sending DIS entity state PDUs or HLA attribute updates.

Example 4.2

```

1  function simpleTalk(protocol,time)
2  try
3      %connection parameters
4      execName      = 'VR-Link';
5      fedName       = 'VR-Link talk';
6      port          = 3000;
7      exerciseId    = 1;
8      siteId        = 1;
9      applicationNum = 16;
10     % Create a connection to the exercise or federation execution.
11     if ~isempty(strfind(protocol,'HLA'))
12         entityId = 'F16';
13         exConn = DtExerciseConn(protocol,execName,fedName);
14     elseif strcmp(protocol,'DIS')
15         entityId = 33;
16         exConn = DtExerciseConn(protocol,
17             port,exerciseId,siteId,applicationNum);
17     end
18     % Create the entity type
19     lib      = get(exConn,'lib');
20     % Get enumerations from the library
21     kind     = DtGetEnum('DtEntityKind','DtPlatform');
22     domain   = DtGetEnum('DtPlatformDomain','DtPlatformDomainAir');
23     country  = DtGetEnum('DtCountryCode','DtUnitedStates');
24     category = DtGetEnum('DtPlatformAir','DtFighter');
25     subCategory = DtGetEnum('DtPlatformAirUSFighter','DtF16');
26     specific = DtGetEnum('DtF16','DtBasicF16');
27     extra    = 0;
28     type=[kind,domain,country,category,subCategory,specific,extra];
29     % Create the entity
30     entity = DtEntityPublisher(exConn,type,entityId);
31     % Create a topographic view on the state repository, so we
32     % can set location information in topographic coordinates.
33     refLatitude = 10.0086 * pi/180;
34     refLongitude = 60.0136 * pi/180;
35     transform = DtTopoToGeocTransform(refLatitude,refLongitude);
36     % We can use the ESR to set state.
37     set(entity,'MarkingText','VR-Link');
38     % Lets set the starting location and velocity
39     location    = [0,0,-250];
40     velocity    = [20,0,0];
41     orientation = [0,0,0];
42     % Send a Fire Interaction.
43     fire = generateInteraction(exConn,'FireInteraction');
44     fire.attackerId = get(entity,'id');
45     send(exConn,'FireInteraction',fire);
46     % Main loop
47     dt      = 0.1;
48     simTime = get(exConn,'simTime');
```

```

49     while simTime <= time
50         % Set the current sim time
51         set(exConn,'simTime',simTime);
52         % Get input and process messages
53         drainInput(exConn);
54         % Set the current location information.
55         [loc, vel,null,ori] =
            DtConvertCoord('topographic','geocentric',...
56             location,velocity,[],orientation,transform);
57         set(entity,'location',loc);
58         set(entity,'velocity',vel);
59         set(entity,'orientation',ori);
60         % Call tick, which insures that any data that needs to be
61         % updated is sent.
62         tick(entity);
63         % Set up for next iteration.
64         location(1) = location(1) + (velocity(1) * dt);
65         simTime = simTime + dt;
66         % Wait till real time equals simulation time of next step.
67         realTime = get(exConn,'elapsedRealTime');
68         display(['Current Sim Time: ' num2str(simTime)]);
69         pause(simTime - realTime);
70     end
71     %clean up entity memory
72     destroy(entity);
73     entity = [];
74     %clean up simulation memory
75     disconnect(exConn);
76     destroy(exConn);
77     exConn = [];
78 catch
79     %clean up entity memory
80     if exist('entity','var')
81         destroy(entity);
82     end
83     %clean up simulation memory
84     if exist('exConn','var')
85         if get(exConn,'isConnected')
86             disconnect(exConn);
87         end
88         destroy(exConn);
89     end
90     error(lasterr);
91 end
92 end

```

Connecting to An Exercise

Like the listen-only example, this program creates an exercise connection to provide an interface to the RTI or DIS network (lines 10 through 17).

Managing Entities

Lines 18 - 28 define the entity type the F16 will use. The entity is created by creating an entity publisher, created in line 30. The entity publisher manages the generation of messages for this particular entity. It provides a tick() function, which causes state information to be sent to the network, if necessary. After the publisher is created, the entity's location, velocity, and orientation are specified.

Sending Interactions

The application generates and sends a fire interaction in lines 42-45.

Sending State Messages

As in the listen-only example, this program sets simulation time at the start of each iteration (line 51).

i

While this program's main purpose is to demonstrate the sending of data to the network, it is not a true send-only application. Incoming data is also processed with the drainInput() call on line 53. This call is required for HLA, because this is where we tick the RTI.

The program updates the F16's positional data in lines 54 - 59, and ticks the entity publisher in line 62 to send the updated data onto the network.

Cleanup

The rest of the example destroys the entity and destroys the exercise connection.

4.2.3. Running and Testing the Examples and Your Applications

The talk and listen examples, as well as some of the other examples provided, are only one half of the simulation equation. You cannot tell if talk is talking unless you have another application to listen to its output. Similarly, the listen example will not produce any output unless it has something to listen to. So, to verify that the examples work, as well as for simple testing of the applications you run, you can use the utilities included with VR-Link. If you are developing for HLA, the trial mode of the MÄK RTI allows you to run two federates, which is adequate for the simple tests described in this section.

In all cases, the MATLAB application and the external application must be properly configured for the protocol you are using.

Testing the Output of Example Applications

To test an example that generates output, such as *simpleTalk.m*, run the appropriate netdump application (*netdump* for DIS, *hlaNetdump* for HLA.) If you are connected to the network, netdump will display the update messages that the talk application is publishing.

If you are running an HLA example, make sure you run *vrlinkx.x/binHLA13/hlaNetdump*, since the examples are all for HLA 13. You do not need to run the rtiexec for talk or listen.

If you want to run *talkTM.m*, which uses time management, you must run the rtiexec.

Testing Examples that Need Input

To test an example that requires input, such as *simpleListen.m*, run the VR-Link f18 application or a MÄK Logger recording. The listen application will display the data it is receiving.

4.3. Connecting to an Exercise

You connect to an exercise through the exercise connection (*DtExerciseConn*). The *DtExerciseConn*() constructor automatically connects to the exercise. It takes different arguments for HLA and DIS.

To connect to an HLA13 or HLA1516 exercise, use the *DtExerciseConn*() function as follows:

```
DtExerciseConn('protocol',exerciseName,federateName, [args])
```

where:

- ♦ *protocol* – Specifies the HLA specification being used, either HLA13 or HLA1516.
- ♦ *exerciseName* – A string that specifies the federation execution.
- ♦ *federateName* – A string that specifies the name of this federate.
- ♦ *fedFile* – Optionally, specifies the FED file. If the FED file is not specified, it is assumed to be *exerciseName.fed*.
- ♦ *fomVersion* – Optionally, specifies the version of the RPR FOM being used, as follows:
 - .5 = RPR FOM 0.5
 - .7 = RPR FOM 0.7
 - .8 = RPR FOM 0.8
 - 1.0 = RPR FOM 1.0
 - 2.006 = RPR FOM 2.0 draft 6
 - 2.0014 = RPR FOM 2.0 draft 14
 - 2.0017 = RPR FOM 2.0 draft 17.

Default: RPR FOM 1.0.

- ♦ *fomMapper* – Optionally, specifies the name of a VR-Link FOM Mapper DLL (built with the same compiler as the HLA/DIS Toolbox.)

```
% Specifies an exercise connection using default values
exConn = DtExerciseConn('HLA13','VR-Link','Matlab Federate');
% Specifies an exercise connection using optional arguments
exConn = DtExerciseConn('HLA1516','VR-Link','ML',
    'VR-Link20014.fed',2.0014);
```

To connect to a DIS exercise, use *DtExerciseConn*() as follows:

```
DtExerciseConn('protocol',port,exerciseId,siteId,appNumber)
```

where:

- ♦ *port* – Specifies the UDP port for DIS traffic.
- ♦ *exerciseId* – Specifies the DIS exercise ID number.
- ♦ *siteId* – Specifies the DIS site ID number.
- ♦ *appNumber* – Specifies a unique identifier for the DIS simulation, for example:

```
% DIS exercise connection
exConn = DtExerciseConn('DIS',3000,1,2,3);
```



The *DtExerciseConn* constructor automatically calls the *connect()* function. However you can use it explicitly if you need to reconnect following a disconnect, using the same object:

```
% Create, disconnect, then reconnect
exConn = DtExerciseConn('HLA13','VR-Link','MLTest');
disconnect(exConn);
connect(exConn);
```

4.3.1. Getting the Attributes of an Exercise Connection

- To get the attributes of an exercise connection, use the *get()* function, in the format:

```
get(DtExerciseConn,'attributeName')
```

where:

- ♦ *DtExerciseConn* is the exercise connection
- ♦ *attributeName* can be:
 - *protocol* – a string indicating the protocol (DIS, HLA13, HLA1516)
 - *lib* – a string specifying a native library
 - *isConnected* – the status of the connection (true or false)
 - *handle* – the internal handle
 - *elapsedRealTime* – the elapsed real time, in seconds, since the last drain input
 - *simTime* – the internal simulation time
 - *nextEventId* – a unique Event Number
 - *mappedInteractions* – a list of the mapped native interaction classes.

HLA only

The following attributes apply only to HLA:

- ♦ *timeManager* – a *DtTimeManager* object. This object is transient and may be created when needed. It does not contain any native data, but is only valid during the life of the *DtExerciseConn* that created it.
- ♦ *fedName* – the federate name.

- ♦ `execName` – the exercise name.
- ♦ `fomInteractions` – a list of FOM interaction classes.

DIS only

The following attributes apply only to DIS:

- ♦ `port` – the DIS port
- ♦ `exerciseId` – the exercise id [host site appNum].

4.3.2. Setting an Exercise Connection's Attributes

- To set an exercise connection's attributes, use the `set()` function, as follows:

```
set(DtExerciseConn, 'attributeName', value)
```

The only attribute supported by this function is simulation time (`simTime`). It sets the internal simulation time of the `DtExerciseConn`. The simulation time is used when calculating dead-reckoning and sending messages.

4.3.3. Destroying an Exercise Connection

You must destroy an exercise connection to clean up resources and memory in the native libraries.



The `destroy()` function always calls `disconnect()`. You can use `disconnect()` explicitly, but you still must destroy the exercise connection.

- To destroy an exercise connection, use the `destroy()` function, as follows:

```
%example to destroy exConn
exConn = DtExerciseConn('HLA13', 'VR-Link', 'MLTest');
destroy(exConn);
```

4.4. Using Real Time or Simulation Time

The HLA/DIS Toolbox needs to know the simulation time to make its calculations. You can set the simulation time explicitly (as is done in the `simpleListen` and `simpleTalk` examples), or you can let MATLAB set the simulation time implicitly based on real time.

By default, HLA/DIS Toolbox expects you to set the simulation time explicitly (line 51 in `simpleTalk.m`) by setting it in the exercise connection. If you want MATLAB to set the simulation time, call the `useRealTime(DtExerciseConn)` function. If you do this, you should not set the simulation time explicitly. (If you set simulation time explicitly, it will be ignored.)

To return to explicitly specifying the simulation time, call the `useSimTime(DtExerciseConn)` function.

4.5. Draining Input from the Network

- To get update messages from the network, use `drainInput(DtExerciseConn)`. It processes all input messages from the network, and calls any callbacks that are registered.

```
% Example
exConn = DtExerciseConn('HLA13','VR-Link','MLTest');
drainInput(exConn);
```

4.6. Publishing Entity Data

The entity publisher sends entity state data to the network. The publisher also controls dead-reckoning and heartbeats (for DIS).

4.6.1. Creating an Entity Publisher

- To create an entity publisher, use the `DtEntityPublisher()` function, as follows:

```
DtEntityPublisher(DtExerciseConn,type,nameId)
```

where:

- ♦ *DtExerciseConn* is the exercise connection
- ♦ *type* is a seven-element entity enumeration
- ♦ *nameId* is protocol-dependent, as follows:
 - DIS: *nameId* is an integer (entity number)
 - HLA: *nameId* is a string.

```
% example for HLA entity publisher
publisher = DtEntityPublisher(exConn,[1,3,4,3,2,1,2],'EntityName');
```

```
% example for DIS entity publisher
publisher = DtEntityPublisher(exConn,[1,3,4,3,2,1,2],33);
```

4.6.2. Getting Entity Publisher Properties and Entity Attributes

You can get the properties of an entity publisher, including the attributes of the entity.

- To get the properties of an entity publisher, use the `get()` function, as follows:

```
get(DtEntityPublisher,'property')
```

where:

- ♦ *DtEntityPublisher* is the entity publisher
- ♦ *property* can be one of the following:
 - *id* – the entity ID. In DIS it is a number; in HLA it is the name.
 - *handle* – the internal handle.
 - *exConn* – the DtExerciseConn of the publisher.
 - *attributes* – gets all the attributes, in a MATLAB structure.

The attributes that can be in the structure are:

- ♦ GlobalId
- ♦ Site
- ♦ Host
- ♦ EntityId
- ♦ TypeKind
- ♦ TypeDomain
- ♦ TypeCountry
- ♦ TypeCategory
- ♦ TypeSubCategory
- ♦ TypeSpecific
- ♦ TypeExtra
- ♦ Location
- ♦ Velocity
- ♦ Acceleration
- ♦ Orientation
- ♦ RotationalVelocity
- ♦ TimeStamp
- ♦ TimeStampType
- ♦ Appearance
- ♦ Capabilities
- ♦ Algorithm
- ♦ MarkingText
- ♦ ForceId



All attribute names are case-insensitive.

```
% Get the entity ID
id = get(publisher,'id');
```

4.6.3. Setting Entity Publisher Properties and Entity Attributes

You can set the attributes on an entity through its entity publisher.

- To set entity publisher properties, use the `set()` function, as follows:

```
set(DtEntityPublisher,'property',value);
```

The only property supported is the `attributes` property. To set the attributes, they must be in a MATLAB structure that has the same format as that returned by `get(DtEntityPublisher,attributes)`.

```
% Get the entity attributes
attrs = get(publisher,'attributes');
% Set the location attribute
attrs.Location = [1,2,3];
set(publisher,'attributes',attrs);
% Another way to set location
set(publisher,'Location',[1,2,3]);
```

Table 4-1 lists entity attributes and their data types.

Table 4-1: Entity publisher attributes and data types

Attribute	Datatype
TypeKind	INT32
TypeDomain	INT32
TypeCountry	INT32
TypeCategory	INT32
TypeSubCategory	INT32
TypeSpecific	INT32
TypeExtra	INT32
Location	3 doubles
Velocity	3 doubles
Acceleration	3 doubles
Orientation	3 doubles
RotationalVelocity	3 doubles
TimeStamp	double
TimeStampType	INT32
Appearance	INT32
Capabilities	INT32
Algorithm	INT32
MarkingText	String
Forceld	INT32

4.6.4. Sending Entity Update Messages

To send entity update messages to the network, you must tick the entity publisher. In the `tick()` function, the HLA/DIS Toolbox determines whether or not it needs to send an update message for an entity according to the rules of the RPR FOM or DIS.

- To tick the entity publisher, use the `tick()` function, as follows:

```
tick(publisher);
```

4.6.5. Destroying an Entity Publisher

You must destroy an entity publisher before you destroy or disconnect from the exercise connection.

- To destroy an entity publisher, use the `destroy()` function, as follows:

```
destroy(publisher);
```

4.7. Learning About Remote Entities

A simulation learns about remote entities when it drains input every tick. A list of remote entities is maintained in a reflected entity list (*DtReflectedEntity*). The state of an entity is maintained as a MATLAB structure.

4.7.1. Creating the Reflected Entity List

- To create a *DtReflectedEntityList*, use the `DtReflectedEntityList()` constructor, as follows:

```
DtReflectedEntityList(DtExerciseConn,discoverAfterType,useSmoothing)
```

where:

- ♦ *DtExerciseConn* is the exercise connection for the simulation.
- ♦ *discoverAfterType* specifies whether or not the simulation should wait to discover an entity until after the type attribute is received.
- ♦ *useSmoothing* specifies whether or not to use smoothing.

```
reflEntList = DtReflectedEntityList(exConn,false,true);
```

4.7.2. Getting Entity State from a Reflected Entity List

You can get the state of an entity by its index or by its name. When you get entity state, the HLA/DIS Toolbox returns an entity state encoded into a MATLAB structure. If the index or name is invalid, the HLA/DIS Toolbox returns an empty structure.

- To get entity state by index, use the `getEntityByIndex()` function, as follows:

```
getEntityByIndex(DtReflectedEntityList,index)
```

where `DtReflectedEntityList` is the reflected entity list and `index` is the index for the list element to return.

```
% To iterate through all entities
for index = 0:get(reflEntList,'count')-1
    entityState = getEntityByIndex(reflEntList,index);
end
```



The index is 0-based.

- To get an entity state by entity name, use the `getEntityByName()` function, as follows:

```
getEntityByName(DtReflectedEntityList,name)
```

For HLA, `name` is the name of the entity. For DIS, `name` is the entity ID encoded as a string, for example, 1:2:3.

The following example code gets the entity object, then gets some attributes:

```
% Get the first entity in the list
reflectedEntity = getEntityByIndex(entityList,0);
% Get the entity attributes you are interested in
if ~isempty(reflectedEntity)
    loc = reflectedEntity.Location;
    vel = reflectedEntity.Velocity;
    acc = reflectedEntity.Acceleration;
    disp(['Position: ' num2str(loc(1)) ' ' num2str(loc(2)) ...
        ' ' num2str(loc(3))]);
    disp(['Velocity: ' num2str(vel(1)) ' ' num2str(vel(2)) ...
        ' ' num2str(vel(3))]);
    disp(['Accelera: ' num2str(acc(1)) ' ' num2str(acc(2)) ...
        ' ' num2str(acc(3))]);
end
```

4.7.3. Getting Properties of a Reflected Entity List

- To get the properties of a reflected entity list, use the `get()` function, as follows:

```
get(DtReflectedEntityList, 'property');
```

where *property* can be:

- ♦ **handle** – A handle to the list
- ♦ **exConn** – The exercise connection to which the list is bound
- ♦ **count** – The number of reflected entities in the list
- ♦ **timeoutrate** – The time-out rate of the list (DIS only).

4.7.4. Setting Properties for a Reflected Entity List

- To set a property of the Reflected Entity list, use the `set()` function, as follows:

```
set(DtReflectedEntityList, 'property', value)
```

The only property you can set is the DIS **timeoutrate**.

4.7.5. Destroying a Reflected Entity List

When a simulation is complete, you must destroy it to deallocate memory. You should destroy the reflected entity list before you destroy the exercise connection.

- To destroy an exercise connection, use the `destroy()` function.

```
destroy(reflEntList)
```

4.8. Sending and Receiving Interactions

Interactions are transient events such as firing a munition or the detonation of a munition. Simulations can create interactions and send them over the network, and they can receive interactions from other simulations.

4.8.1. Creating Interactions

The HLA/DIS Toolbox has built-in support for fire and detonate interactions. For HLA simulations, if you want to use additional interactions, you can write encoders and decoders for them. For details, see [“Using Interaction Translators \(HLA Only\),”](#) on page 4-21.

- To create an interaction, use the `generateInteraction()` function.

```
generateInteraction(DtExerciseConn, 'interactionClass')
```

The `generateInteraction()` function generates a MATLAB structure that contains the parameters of the interaction class. The structure can be used in the `send()` function. For interactions that have built-in support, each field contains a default value.

4.8.2. Sending Interactions

- To send an interaction, use the `send()` function, as follows:

```
send(DtExerciseConn,'interactionClass',struct)
```

where:

- ♦ `DtExerciseConn` is the exercise connection.
- ♦ `interactionClass` is the name of the interaction: `FireInteraction` or `DetonateInteraction`.
- ♦ `struct` is a MATLAB structure that holds the parameters of the interaction class.

The HLA/DIS Toolbox sends a time stamped interaction. Empty parameters are not sent.

```
% Send a fire interaction
interaction = generateInteraction(exConn,'FireInteraction');
send(exConn,'FireInteraction',interaction);
```

4.8.3. Adding Interaction Callbacks

- To add an interaction callback, use the `addInteractionCallback()` function, as follows:

```
addInteractionCallback(DtExerciseConn,'interactionClass',
    @function_handle)
```

where:

- ♦ `DtExerciseConn` is the exercise connection.
- ♦ `interactionClass` is the name of the interaction class for which you want to add a callback.
- ♦ `function_handle` is a handle to the function that is to be called after `drainInput()` for each interaction received. The callback signature is `function(exConn,interactionStruct)`. The handle returned is used to remove the callback.

```
% Defines a callback, then adds it
function printInt(exConn,interaction)
    interaction
end
handle = addInteractionCallback(exConn,'WeaponFire',@printInt);
```

4.8.4. Removing Interaction Callbacks

- To remove a callback use the `removeInteractionCallback()` function, as follows:

```
removeInteractionCallback(DtExerciseConn,callbackHandle)
```

```
% removes a callback
removeInteractionCallback(exConn,handle);
```

4.8.5. Using Interaction Translators (HLA Only)

The HLA/DIS Toolbox includes built-in support for fire and detonate interactions. If you want to receive or send other kinds of interactions, you must write translators for them. The translators decode received interactions into a MATLAB structure and encode MATLAB arrays to an outgoing interaction. The default decoding is a blank array (`[]`). The default encoding is the MATLAB type used.

- To add a translator for an interaction attribute, use the `addTranslator()` function, in one of the following formats:

```
addTranslator(DtExerciseConn, 'fomInteractionClass', 'parameterName',
             mxClassId, size)
addTranslator(DtExerciseConn, 'fomInteractionClass', 'parameterName',
             {mxClassId, size, ...})
```

where:

- ♦ *fomInteractionClass* is the name of the interaction.
- ♦ *parameterName* is the name of the parameter being added by the function.
- ♦ *mxClassId* can be:
 - `mxLOGICAL_CLASS` – translated as a unsigned 8bit integer
 - `mxSINGLE_CLASS` – translated as a 32bit floating point number
 - `mxDOUBLE_CLASS` – translated as a 64bit floating point number
 - `mxINT8_CLASS` – translated as a signed 8bit integer
 - `mxINT16_CLASS` – translated as a signed 16bit integer
 - `mxINT32_CLASS` – translated as a signed 32bit integer
 - `mxINT64_CLASS` – translated as a signed 64bit integer
 - `mxUINT8_CLASS` – translated as a unsigned 8bit integer
 - `mxUINT16_CLASS` – translated as a unsigned 16bit integer
 - `mxUINT32_CLASS` – translated as a unsigned 32bit integer
 - `mxUINT64_CLASS` – translated as a unsigned 64bit integer
 - `mxCHAR_CLASS` – translated as a ASCII String.
- ♦ *size* is the size of the attribute.

Translators can be single types or packed types depending on the data in the attribute. Some attributes are single values, and thus only one type and size need to be specified.

```
addTranslator(exConn, 'WeaponFire', 'FireMissionIndex', 'mxUINT32_CLASS',
             1);
```

Some attributes are multiple values of the same type.

```
addTranslator(exConn, 'WeaponFire', 'FiringLocation', 'mxDOUBLE_CLASS', 3);
```

Other attributes are multiple types packed together. The `addTranslator()` function is passed a cell array containing "type, size" pairs to set the packed attribute.

```
addTranslator(exConn, 'WeaponFire', 'EventIdentifier',  
             {'mxINT16_CLASS', 1, 'mxCHAR_CLASS', -1});
```

The *defineWeaponFire.m* example (in *matlabroot/toolbox/hladistbx.x/demos*) shows how to add translators for all of the attributes of a `WeaponFire` interaction. You must add each attribute separately.

i

All strings are dynamically sized, and thus the size argument is ignored (an acceptable value is -1). If a fixed length string is desired, it must be translated as a `mxINT8_CLASS` with the size, then converted in MATLAB to and from a string. Also, since strings are dynamically sized, they can only be used as the last type of a packed attribute.

Removing Interaction Translators

- To remove a translator, use `removeTranslator()`, as follows:

```
removeTranslator(DtExerciseConn, 'fomInteractionClass', 'parameterName').
```

The following example shows how to add a translator and then remove it:

```
% Add a translator  
success = addTranslator(exConn,  
    'WeaponFire', 'FuseType', 'mxUINT32_CLASS', 1);  
% Remove a translator  
removeTranslator(exConn, 'WeaponFire', 'FuseType');
```

4.9. Using HLA Time Management

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

4.9.1. Instantiating the Time Manager

The HLA/DIS Toolbox uses a *DtTimeManager* to control HLA time management. The following examples have the exact same effect.

- To create the Time Manager explicitly, use the `DtTimeManager()` function.

```
timeManager = DtTimeManager(exConn);
```

- To create the Time Manager implicitly, use the `get()` function.

```
timeManager = get(exConn, 'TimeManager');
```

4.9.2. Advancing Time

- To request and process time advance RTI calls, use the `advanceTime()` function.

```
advanceTime(DtTimeManager, timeStep)
```

where *timeStep* is the time step to advance. If the application cannot advance, it returns zero.



The result of a successful time advance may not match the *timeStep*, because time is determined by the RTI. You should ask the Time Manager for the current federation time when you set simulation time.

```
% example  
success = advanceTime(timeManager, .5);
```

4.9.3. Getting and Setting Time Manager Properties

You can get and set the following Time Manager properties:

- ♦ **lookAhead** – The current lookahead for the Time Manager; only used when Time Regulating.
- ♦ **fedTime** – The current federation time. Use this value to set simulation time when managing time.
- To get the properties of the Time Manager, use the `get()` function, as follows:

```
get(DtTimeManager, 'property')
```

where *property* is **lookAhead** or **fedTime**.

- To set Time Manager properties, use the `set()` function, as follows:

```
set(DtTimeManager, 'property', value)
```

where *property* is **lookAhead** or **fedTime**, and *value* is the value specified for the property.



The resulting federation time may not match the value set.

4.9.4. Starting and Stopping Time Regulation

When you use HLA time management, you can specify that a federate is a time regulating federate. You can also remove this designation.

- To specify that a federate is a time regulating federate, use the `startRegulating()` function, as follows:

```
startRegulating(DtTimeManager, lookahead)
```

where *lookahead* specifies the lookahead value for the federate.

```
% example  
startRegulating(timeManager, .5);
```

- To stop time regulation, use the `stopRegulating()` function, for example:

```
stopRegulating(timeManager);
```

4.9.5. Starting and Stopping Time Constraint

In HLA time management, you can specify that a federate be a time-constrained federate. Its advancement is constrained by the regulating federates.

- To specify that a federate be time-constrained, use the `startConstraining()` function, for example:

```
startConstraining(timeManager);
```

- To specify that a federate no longer be time-constrained, use the `stopConstraining()` function, for example:

```
stopConstraining(timeManager);
```

4.10. HLA/DIS Toolbox Library Functions

This section lists the functions you can use to load, unload, and verify libraries.



These libraries are called automatically by the various HLA/DIS Toolbox classes and functions. You do not have to use them unless you have a specific reason for doing so.

`DtLoadLib(protocol | '')` loads the library for the specified protocol or the shared utility library.

```
% Load the hla13 library
libStr = DtLoadLib('HLA13');
% Load the shared utility library
utilLib = DtLoadLib('')
```

`DtUnloadLib(protocol)` Unloads the library from MATLAB.

```
% Example to unload hla13 library
DtUnloadLib('HLA13');
```

`DtDebug()` returns true if you load debug libraries. The default is false, because debug libraries may not be available on platforms. You do not need to call this function.

`DtSupportedProtocols()` returns a cell array of the available protocol libraries. The supported libraries for HLA/DIS Toolbox 1.0 are DIS, HLA13, and HLA1516.

```
% Example to display all protocols
disp(DtSupportedProtocols());
```

`DtValidProtocol('protocol')` returns true if the specified protocol is supported.

```
% Example to test protocol string
if DtValidProtocol('invalid')
    disp('Invalid Protocol');
end
```

`DtLoadEnums('library')` returns arrays of functions, structures, and enumerations used by the specified library.

```
% Get all enumerations in the shared library
[functs,structs,enums] = DtLoadEnums(DtLoadLib(''));
disp(enums);
```

4.11. Coordinate Systems and Coordinate Conversions

The HLA/DIS Toolbox supports several coordinate systems and provides functions for converting locations, vectors, and orientations from one system to another.

4.11.1. Geocentric Coordinates

DIS, and the RPR FOM in HLA, specify that world locations, velocities, accelerations and orientations be represented with respect to a right-hand geocentric Cartesian coordinate system. The origin of this 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 (Figure 4-2).

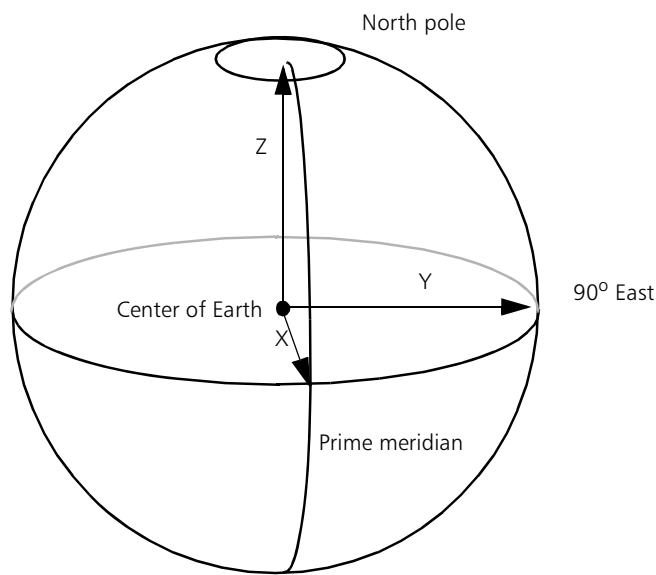


Figure 4-2. Geocentric coordinate system

4.11.2. Geodetic Coordinates

Geodetic coordinates are another means of specifying location in the world. In VR-Link, geodetic coordinates consist of latitude and longitude in radians, and altitude in meters above a certain reference ellipsoid, for example, the surface of the earth at sea level.

Negative latitudes are south latitudes; negative longitudes are west longitudes.

Choosing a Reference Ellipsoid

The default reference ellipsoid used for geocentric to geodetic conversions is WGS84.

- To configure the reference ellipsoid, use the `DtInitGeodSphere()` function, as follows:

```
DtInitGeodSphere(name)
```

where *name* is one of the following:

- ♦ DtWGS84 (default)
- ♦ DtED50
- ♦ DtNASANAD27
- ♦ DtNASBNAD27
- ♦ DtCONUSNAD27.

```
% Example  
DtInitGeodSphere('DtWGS84');
```



The name values are case-insensitive.

4.11.3. Topographic Coordinates

HLA/DIS Toolbox defines a topographic coordinate system as 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 (Figure 4-3). Obviously, there are an infinite number of topographic coordinate systems – one for each point on the earth's surface.

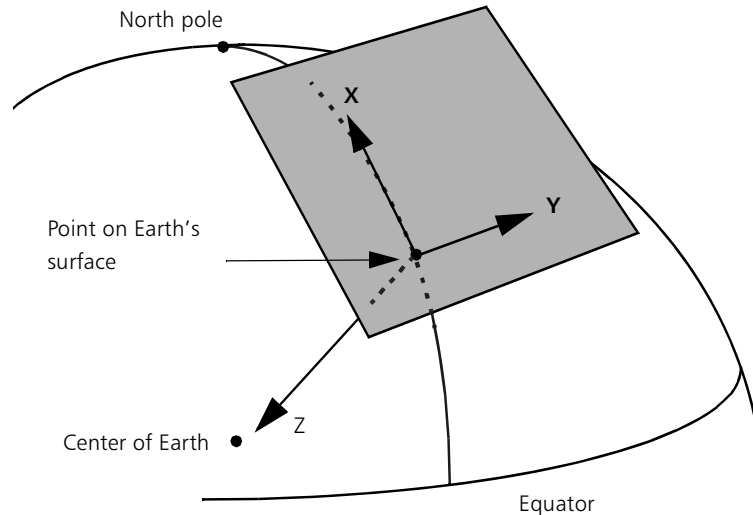


Figure 4-3. Topographic coordinate system

Converting Between Topographic and Geocentric Coordinate Systems

If you want to convert to or from topographic coordinates, you must specify a transformation matrix.

- To specify a geocentric to topographic transformation matrix, use the `DtGeocToTopoTransform()` function, as follows:

```
DtGeocToTopoTransform(refLat,refLong)
```

where:

- ♦ *refLat* = the latitude (in Radians) of the topographic reference point
 - ♦ *refLong* = the longitude (in Radians) of the topographic reference point.
- To specify a topographic to geocentric transformation matrix, use the `DtTopoToGeocTransform()` function, as follows:

```
DtTopoToGeocTransform(refLat,refLong)
```

where *refLat* and *refLong* have the same meaning as in the `DtGeocToTopoTransform()` function.

```
% Geocentric to topographic
deg2rad = pi/180;
geoc2Topo = DtGeocToTopoTransform(35 * deg2rad,122 * deg2rad);

% Topographic to geocentric
deg2rad = pi/180;
topo2Geoc = DtTopoToGeocTransform(35 * deg2rad,122 * deg2rad);
```

4.11.4. Setting the UTM Reference Point

Locations in the world can be specified using a UTM coordinate system. UTM coordinates are mapped to a reference ellipsoid approximating the surface of the earth using a Universal Transverse Mercator projection. The coordinates consist of easting and northing from an origin or reference location, and an altitude above a reference ellipsoid, all in meters.

In a true UTM coordinate system, the origin or reference point (0,0,0) is often a point on the equator at the center of a particular UTM zone. (The world is longitudinally divided into 60 UTM zones, each centered on an odd multiple of 3 degrees). However, you can define an “offset UTM” coordinate system by passing any arbitrary reference point, to the function `DtInitUtm()`.

You must establish the world location of the origin of your UTM attribute system before you use any of the UTM coordinate conversion functions. The HLA/DIS Toolbox supports the use of only one UTM reference point at a time. However, you can set the origin multiple times in your application to define the UTM origin you want to use for the each set of conversions.

- To initialize the UTM reference point, use the `DtInitUTM()` function, as follows:

```
DtInitUTM(refLat,latDir,refLong,longDir,onlyPositive)
```

where:

- ♦ *refLat* = is the latitude in [degrees, minutes, seconds]
- ♦ *latDir* = is the latitude direction: North, South, East, West
- ♦ *refLong* = is the longitude in [degrees, minutes, seconds]
- ♦ *longDir* = is the longitude direction: North, South, East, West
- ♦ *onlyPositive* - specifies whether or not offset UTM coordinates are always positive.

```
% example
DtInitUTM([35 0 0], 'North', [122 0 ,0], 'West', true);
```

4.11.5. Differences Between UTM and Topographic Coordinates

Trouble often arises when programmers do not understand the differences between a UTM coordinate system and a topographic coordinate system that share the same origin.

Although the coordinates of a point near the origin may be similar in the two systems, they are different in several ways.

- In a UTM system, Z represents altitude above the earth's surface. In the topographic frame, $-Z$ represents height above a plane tangent to the earth's surface, which deviates from altitude above the earth's surface itself as one moves away from the origin (Figure 4-4).

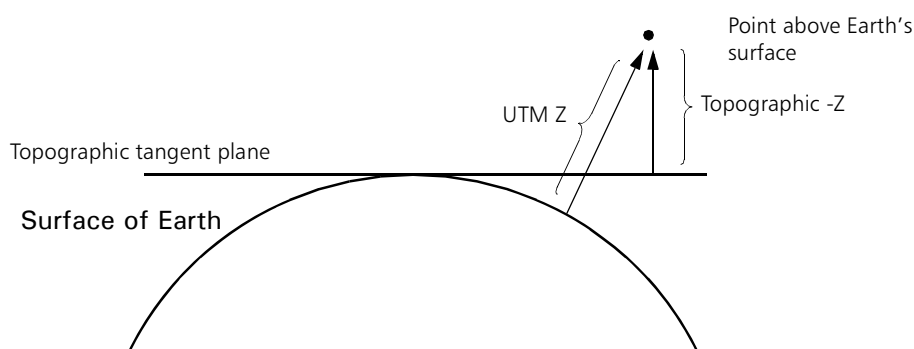


Figure 4-4. Comparison of UTM Z and topographic $-Z$

- Neglecting the small angle between northing and north, UTM coordinates X , Y , and Z correspond to east, north, and up, while the topographic X , Y , and Z correspond to north, east, and down (Figure 4-5).

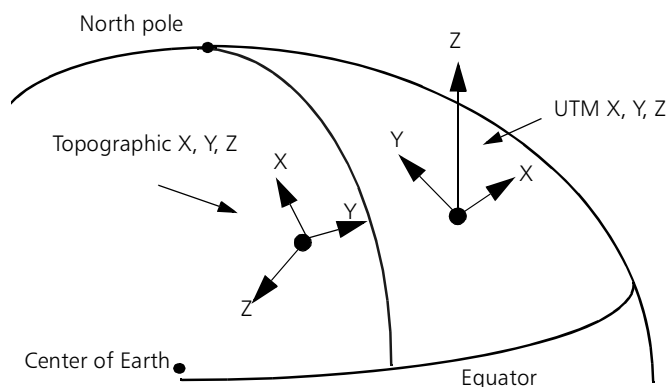


Figure 4-5. Topographic X , Y , and Z compared to UTM

- ♦ Although lines of constant easting in UTM coordinates run very close to north/south, they do not run truly north/south as lines of longitude do. This means that two points with equal easting values in their UTM coordinates, such as {100, 100, 0} and {200, 100, 0} do not lie exactly on the same line of longitude. The second point is not due north of the first. Depending on location within a UTM zone, the deviation can be plus or minus a couple of degrees.

4.11.6. Converting Between Coordinate Systems

- To convert from one coordinate system to another, use the `DtConvertCoord()` function, as follows:

```
DtConvertCoord(coordSystemIn, coordSystemOut, coordIn, vecIn, accIn, oriIn,  
              varargin)
```

where:

- ♦ *coordSystemIn* specifies the input coordinate system, as follows:
 - `geocentric`
 - `topographic`
 - `geodetic`
 - `UTM`.
- ♦ *coordSystemOut* specifies the output coordinate system, as follows:
 - `geocentric`
 - `topographic`
 - `geodetic`
 - `UTM`.
- ♦ *coordIn* specifies a set of X, Y, and Z coordinates in the *coordSystemIn* coordinate system.
- ♦ *vecIn* specifies a three-element vector *coordSystemIn* coordinates. This is usually velocity. Use `[]` (empty value) if there is no vector to convert.
- ♦ *accIn* specifies a three-element vector in *coordSystemIn* coordinates. This is usually acceleration. Use `[]` if there is no vector to convert.
- ♦ *oriIn* specifies a TaitBryan in *coordSystemIn* space expects three doubles. Typically orientation uses `[]` if there is no TaitBryan to convert.
- ♦ *varargin* specifies options specific to the coordinate systems being converted.

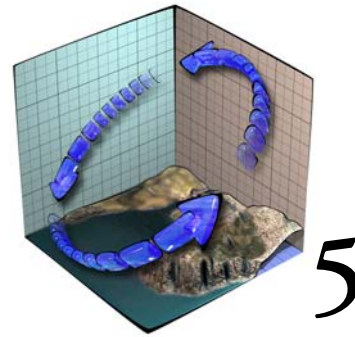
Notes

- ♦ For topographic conversions, you must pass in a transformation matrix created with the DtTopoToGeocTransform() or DtGeocToTopoTransform() functions.
 - If you are converting topographic to any other system, use DtTopoToGeocTransform(), because geocentric is an intermediary stage.
 - If you are converting to topographic from any other system, use DtGeocToTopoTransform(), because geocentric is an intermediary stage.
- ♦ Geodetic space is only available for coordinates, so no vector or orientation conversion is supported.

```
% Convert Topographic location, velocity and orientation to Geocentric
transform = DtTopoToGeocTransform(refLatitude,refLongitude);
[loc, vel,null,ori] = DtConvertCoord('topographic','geocentric',...
location,velocity,[],orientation,transform);
```

```
% Convert UTM location velocity and acceleration to Topographic
DtInitUTM([35 0 0],'North',[122,0,0],'West',true);
transform = DtGeocToTopoTransform(refLatitude,refLongitude);
[loc,vel,acc,null] = DtConvertCoord('UTM','topographic',location,
velocity,acceleration,[],transform);
```

```
% Convert Geocentric coordinates to Geodetic Lat,Long,Alt
% lat & long will be in radians)
DtInitGeodSphere('DtWGS84');
[latLongAlt,null,null,null] = DtConvertCoord('geocentric',
'geodetic',location,[],[],[]);
```

Using the HLA/DIS Toolbox with Simulink

This chapter explains how to use the HLA/DIS Toolbox to create HLA or DIS simulations in Simulink.

Introduction	5-3
The HLA/DIS Toolbox Library	5-3
Blockset Design	5-4
Example Applications for Simulink	5-5
Publishing a Simulink Model	5-5
Create an Exercise Connection	5-6
Add a Simple Entity Publisher	5-6
Add the Entity Model	5-6
Add a Coordinate Conversion Block	5-7
Add a Wall Clock Timer	5-7
Add the Final Connections	5-7
Listening for Remote Entities in a Simulink Model	5-8
Create an Exercise Connection	5-8
Add a Reflected Entity List	5-9
Add a Reflected Entity Selector	5-9
Add a Simple Reflected Entity	5-10
Add a Wall Clock Timer	5-10
Blockset Descriptions	5-11
Exercise Connection	5-11
Entity Publisher	5-12
Simple Entity Publisher	5-13

Reflected Entity List.....	5-14
Reflected Entity.....	5-14
Simple Reflected Entity.....	5-15
Reflected Entity Selector	5-16
Interaction Publisher	5-16
Received Interaction.....	5-16
Received Interaction List	5-17
Received Interaction Selector.....	5-17

5.1. Introduction

The HLA/DIS Toolbox has a library of blocksets that you can use to create simulation models and connect simulation models to the DIS network or HLA. [Figure 5-1](#) illustrates the relationship between Simulink and VR-Link.

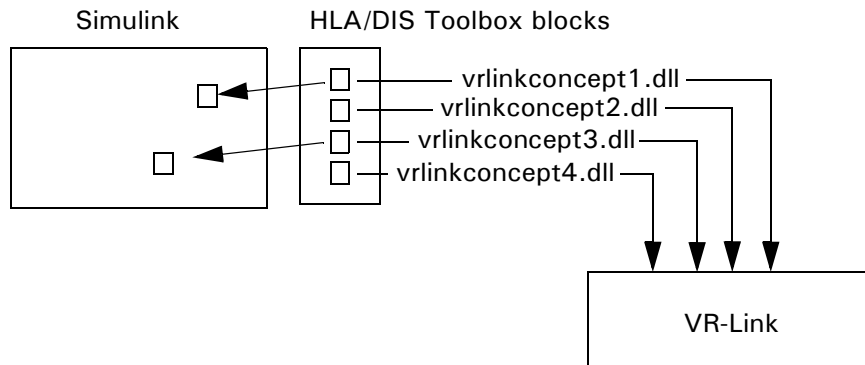


Figure 5-1. Simulink relationship with VR-Link

Each block in a HLA/DIS Toolbox library is an S-Function block implemented in C that represents a VR-Link concept, such as an exercise connection, an entity publisher, or a detonation interaction. The S-Function calls the C++ VR-Link library that it represents.

5.2. The HLA/DIS Toolbox Library

Each of the three protocol-specific blocksets in the HLA/DIS Toolbox library have the following blocks:

- ♦ Exercise Connection
- ♦ Entity Publisher
- ♦ Simple Entity Publisher
- ♦ Reflected Entity
- ♦ Reflected Entity List
- ♦ Reflected Entity Selector
- ♦ Simple Reflected Entity
- ♦ Interaction Publisher
- ♦ Received Interaction List
- ♦ Received Interaction Selector
- ♦ Received Interaction.

The blocksets include a block, the Received Interaction List, that does not directly map to functions in the M-code interface. The received interaction list is needed because you cannot use callbacks in the Simulink environment.

The Utility blockset has the following blocks:

- ♦ 3D Vector
- ♦ Constant Vector
- ♦ Coordinate Conversion
- ♦ Geodetic to Degrees
- ♦ String Comparison
- ♦ String Constant
- ♦ String Output
- ♦ String Parameter
- ♦ Wall Clock Timer.



Although the blocks in the DIS and HLA libraries have similar names, you cannot connect DIS blocks to HLA blocks.

5.2.1. Blockset Design

Each block in a library is a functionally complete module. However, most modules must interact with objects from other modules. To do this they need the memory address of the object. In Simulink this address is passed as a handle, which is stored in a global table that maps to a memory address. The handle is a 32 bit integer.

Each data type has its own handle table, so handles are only unique in their datatype. Therefore, it is possible that different blocks could have the same handle. It is up to you to ensure that the correct handle types are connected. The port names indicate which type of handle they expect.

The HLA/DIS Toolbox library uses the following data types:

- ♦ Enum – unsigned 32 bit integer.
- ♦ Handle – unsigned 32bit Integer.
- ♦ ID – protocol dependent ID. In HLA it is a string; in DIS it is three 16 bit integers.
- ♦ String – stored as fixed length eight bit integers. Strings do not need to be zero-terminated, but a zero-terminated string can be stored as a larger fixed string.
- ♦ TBryan – TaitBryan angles, stored in three doubles.
- ♦ Type – Entity type structure stored as 7 unsigned 16 bit integers, which is an enumeration of DtKind, DtDomain, DtCountry, DtCategory, DtSubCategory, DtSpecific, and DtExtra.
- ♦ Vector – 3D Vector stored in three doubles.

Each simulink block mask labels each port with the data type and a descriptive name.

5.3. Example Applications for Simulink

The HLA/DIS Toolbox library has the following sample Simulink applications:

- Coordinate Conversions (protocol independent) gives examples of the different coordinate conversions supported by the HLA/DIS Toolbox.
- Publish Entity (for DIS, HLA 13, and HLA 1516) publishes state updates for a simple entity model.
- Reflected Entity (for DIS, HLA 13, and HLA 1516) creates a reflected entity list, displays the number of entities discovered, and displays state information for the first entity in the list.
- Simulation Monitor (for DIS, HLA 13, and HLA 1516) creates a monitor that displays entities and interactions over time.

For details about how to test the sample applications or your own applications, see [“Running and Testing the Examples and Your Applications,”](#) on page 4-9.

5.4. Publishing a Simulink Model

This section describes Published Entity HLA 13 ([Figure 5-2](#)), which shows how to send data from a Simulink model to the network. This example is the Simulink equivalent to the *simpleTalk.m* example. This example demonstrates use of:

- An Exercise Connection block
- A Coordinate Conversion block
- A Simple Entity Publisher block
- A Wall Clock Timer block.

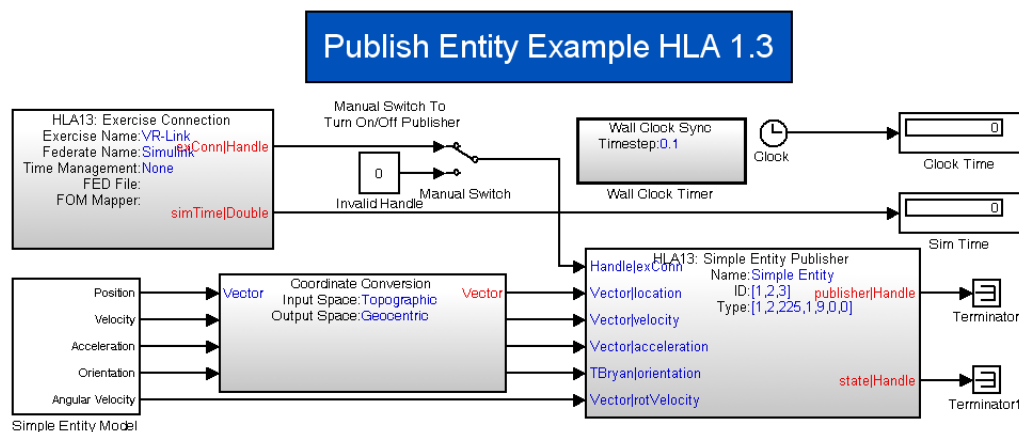


Figure 5-2. Published Entity HLA 13 example

5.4.1. Create an Exercise Connection

For any simulation to connect to the HLA or DIS network, you need an exercise connection.

To create an exercise connection:

1. Drag an Exercise Connection block from the HLA 13 Blockset into your model.
2. Double-click the block to display the block parameters dialog box.
3. For an HLA exercise connection, you must specify the exercise name and federate name. You can also specify use of time management, a FED file, and a FOM Mapper.



The parameters for a DIS exercise are different from HLA. Click the Help button on the block parameters dialog box for a list of parameters and their descriptions.

5.4.2. Add a Simple Entity Publisher

An entity publisher publishes entity state information to the network through the exercise connection. This example uses a Simple Entity Publisher, which manages fewer attributes than the Entity Publisher block, but is in all other ways identical in function.

To add a Simple Entity Publisher:

1. Drag a Simple Entity Publisher block from the HLA 13 Blockset into the model.
2. Connect the exConn output port from the Exercise Connection to the exConn input port of the Simple Entity Publisher. The example adds an on/off switch to this connection, but it is not required.

5.4.3. Add the Entity Model

This example has a simple entity model that generates position, velocity, acceleration, orientation, and angular velocity attributes. We want to publish its state to the network. It could be any entity model that you have developed and want to test.

5.4.4. Add a Coordinate Conversion Block

The simple entity model generates its location data in topographic coordinates. The data sent over the network must be in geocentric coordinates, so we need to convert the coordinate data using a Coordinate Conversion Block.

To add a Coordinate Conversion Block:

1. Drag a Coordinate Conversion Block from the HLA 13 Blockset into the model.
2. Double-click the Coordinate Conversion Block to display its function block parameter dialog box.
3. Specify the input space as topographic and the output space as geocentric.
4. Set the other parameters to the appropriate values for the exercise.
5. Connect the position, velocity, acceleration, and orientation output ports from the entity model to the input port of the Coordinate Conversion Block.
6. Connect the output ports of the Coordinate Conversion Block to the input ports of the Simple Entity Publisher.
7. Connect the angular velocity output from the entity model to the Simple Entity Publisher input port.

5.4.5. Add a Wall Clock Timer

You need to synchronize the simulation time with wall clock time.

To add a Wall Clock Timer block:

1. Drag a Wall Clock Timer block from the Utility Blockset.
2. Double-click the Wall Clock Timer block to display its parameters dialog box.
3. Set the Time Step to match the time step for the simulation.

5.4.6. Add the Final Connections

To finish the model, you need to make some final connections.

To complete the model:

1. Add some terminators to the Simple Entity Publisher output ports. These prevent warnings due to unconnected ports.
2. Connect the Exercise Connection simTime output port to a Display sink.
3. Add a Display sink for the Wall Clock.

5.5. Listening for Remote Entities in a Simulink Model

This section describes Reflected Entity HLA 13 (Figure 5-3), which shows how to listen for remote entities in a Simulink model. This example is the Simulink equivalent to the *simpleListen.m* example. This example demonstrates use of:

- An Exercise Connection block
- A Reflected Entity List block
- A Reflected Entity Selector block
- A Simple Reflected Entity block
- A Wall Clock Timer block.

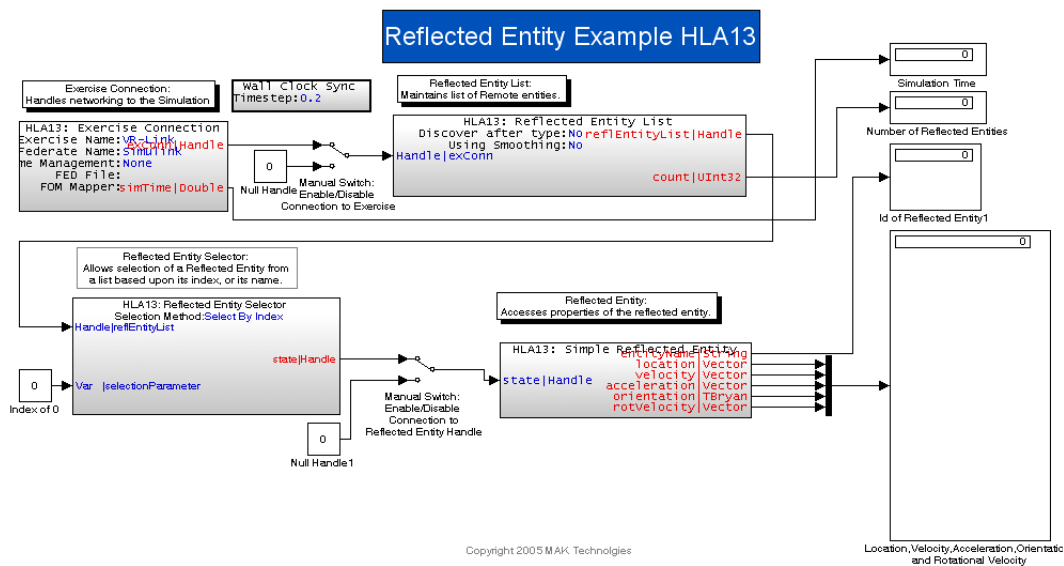


Figure 5-3. Reflected Entity HLA 13 example

5.5.1. Create an Exercise Connection

For any simulation to connect to the HLA or DIS network, you need an exercise connection.

To create an exercise connection:

1. Drag an Exercise Connection block from the HLA 13 Blockset into your model.
2. Double-click the block to display the block parameters dialog box.
3. For an HLA exercise connection, you must specify the exercise name and federate name. You can also specify use of time management, a FED file, and a FOM Mapper.
4. Add a Display Sink to the model and connect the Exercise Connection's *simTime* output port to it.



The parameters for a DIS exercise are different from HLA. Click the Help button on the block parameters dialog box for a list of parameters and their descriptions.

5.5.2. Add a Reflected Entity List

A reflected entity list tracks all of the remote entities discovered on the network. To find out about a remote entity, you must select it from the list and get its attributes.

To add a Reflected Entity List block:

1. Drag a Reflected Entity List block from the HLA 13 Blockset into the model.
2. Optionally, enable or disable the following parameters in the parameter block dialog box:
 - Smoothing – enable or disable.
 - Discover after entity type is known.
3. Connect the Exercise Connection's exConn output port to the Reflected Entity List's exConn input port.
4. Add a Display sink to the model and connect the Reflected Entity List's count output port to it. It displays the number of entities in the reflected entity list.

5.5.3. Add a Reflected Entity Selector

You select entities from the Reflected Entity List with the Reflected Entity Selector. You need to input the selection parameter, either an index or a string.

To add a Reflected Entity Selector block to the model:

1. Drag a Reflected Entity Selector block from the HLA 13 Blockset into the model.
2. In the block's parameter dialog box, specify the selection criteria, index or name. The example uses Select by Index.
3. Connect the Reflected Entity List's reflEntityList output port to the Reflected Entity Selector's reflEntityList input port.
4. Add a Source Block to the model. This block sets the index value for the selection criteria.
5. Connect the Source Block to the Reflected Entity Selector's selectionParameter input port.

5.5.4. Add a Simple Reflected Entity

The reflected entity stores the entity state information. This example uses a Simple Reflected Entity, which manages fewer attributes than the Reflected Entity block, but is in all other ways identical in function.

To add a Simple Reflected Entity:

1. Drag a Simple Reflected Entity block from the HLA 13 Blockset into the model.
2. Connect the Reflected Entity Selector's state output port to the Simple Reflected Entity's state input port.
3. Add Display sinks to display the entity's state and connect the Simple Reflected Entity's output ports to the displays.

5.5.5. Add a Wall Clock Timer

You need to synchronize the simulation time with wall clock time.

To add a Wall Clock Timer block:

1. Drag a Wall Clock Timer block from the Utility Blockset.
2. Double-click the Wall Clock Timer block to display its parameters dialog box.
3. Set the Time Step to match the time step for the simulation.

5.6. Blockset Descriptions

This section describes the purpose of each blockset and lists its input and output ports and parameters.

5.6.1. Exercise Connection

All interaction between your Simulink project and the HLA or DIS network goes through the exercise connection. It is required for any simulation. [Table 5-1](#) lists the input ports, output ports, and parameters for the exercise connection blockset.

Table 5-1: Exercise Connection blockset components

Type	Name	Description
Input Ports	None.	
Output Ports	exConn	Provides a handle to an exercise connection.
	simTime	Uses the simulation time.
HLA Parameters	Exercise name	The name of the HLA federation. Default: VR-Link.
	Federate name	The name of this federate. Default: Simulink.
	Time Management	Specifies whether or not time management is being used.
	FED file	The FED file for this exercise.
DIS Parameters	FOM Mapper	The FOM being used for the exercise.
	UDP Port	The port through which the exercise participants communicate. Default: 3000.
	Exercise number	The exercise number for this simulation.
		 The exercise number:site ID:application ID triplet uniquely identify a participant in a DIS exercise.
	Site ID	The site ID for this simulation.
	Application ID	The application ID for this simulation.

5.6.2. Entity Publisher

An entity publisher maintains information about an entity and sends update messages to the network from its `tick()` function. [Table 5-2](#) lists the input ports, output ports, and parameters for the entity publisher blockset.

Table 5-2: Entity Publisher blockset components (Sheet 1 of 2)

Type	Name	Description
Input Ports	Exercise Connection	Provides a handle to an exercise connection.
	entityName	A string containing the name of the entity.
	entityType	An entity type enumeration.
	entityId	The entity ID.
	location	Vector
	velocity	Vector
	acceleration	Vector
	orientation	TaitBryan
	rotVelocity	Vector
	deadReckonAlg	An enum that represents the dead-reckoning algorithm being used.
	relLocation	Vector
	relVelocity	Vector
	relAcceleration	Vector
	relOrientation	TaitBryan
	relRotVelocity	Vector
	relFrozen	Bool
	relDeadReckonAlg	Enum
	forceld	Enum
	guise	Type
	markingCharSet	Int32
	markingText	The marking text for this entity (a string.)
	appearance	UInt32
	capabilities	UInt32
	measuredSpeed	UInt16
	infraRedSigReplIndex	UInt16
	acousticSigReplIndex	UInt16
	radarSigReplIndex	UInt16

Table 5-2: Entity Publisher blockset components (Sheet 2 of 2)

Type	Name	Description
Output Ports	publisher	A handle to the entity being published.
	state	
Parameters	None.	

5.6.3. Simple Entity Publisher

A simple entity publisher publishes a subset of entity attributes. [Table 5-3](#) lists the input ports, output ports, and parameters for the simple entity publisher blockset.

Table 5-3: Simple Entity Publisher blockset components

Type	Name	Description
Input Ports	Exercise Connection	Provides a handle to an exercise connection.
	entityType	EntityType
	entityId	ID
	location	Vector
	velocity	Vector
	acceleration	Vector
	orientation	TaitBryan
	rotVelocity	Vector
Output Ports	publisher	A handle to an entity publisher.
	state	A handle to the entity state.
Parameters	Name	The name of the entity.
	ID	The entity ID.
	Type	The entity type enumeration.

5.6.4. Reflected Entity List

The reflected entity list is a list of all entities discovered over the network. [Table 5-4](#) lists the input ports, output ports, and parameters for the reflected entity list blockset.

Table 5-4: Reflected Entity List blockset components

Type	Name	Description
Input Ports	state	A handle to the entity state information sent by the publisher.
	reflEntityList	A handle to a reflected entity list.
Output Ports	count	The number of reflected entities.
	Use Smoothing	Specifies use of smoothing (Yes or No.)
Parameters	Discover after entity type is known	Specifies whether or not to wait until the entity type is known before requesting data about this entity.

5.6.5. Reflected Entity

A reflected entity maintains the state of an entity that has been discovered on the network. [Table 5-5](#) lists the input ports, output ports, and parameters for the reflected entity blockset.

Table 5-5: Reflected Entity blockset components (Sheet 1 of 2)

Type	Name	Description
Input Ports	state	Handle
Output Ports	entityName	Name
	entityType	Type
	entityId	Id
	location	Vector
	velocity	Vector
	acceleration	Vector
	orientation	TaitBryan
	rotVelocity	Vector
	deadReckonAlg	Enum
	relLocation	Vector
	relVelocity	Vector
	relAcceleration	Vector
	relOrientation	TaitBryan

Table 5-5: Reflected Entity blockset components (Sheet 2 of 2)

Type	Name	Description
	relRotVelocity	Vector
	relFrozen	Bool
	relDeadReckonAlg	Enum
	forceld	Enum
	guise	Type
	markingCharSet	Int32
	markingText	String
	appearance	UInt32
	capabilities	UInt32
	measuredSpeed	UInt16
	infraRedSigRepIndex	UInt16
	acousticSigRepIndex	UInt16
	radarSigRepIndex	UInt16
Parameters	None.	

5.6.6. Simple Reflected Entity

A simple reflected entity maintains the state of a subset of attributes for an entity that has been discovered on the network. [Table 5-6](#) lists the input ports, output ports, and parameters for the Simple Reflected Entity blockset.

Table 5-6: Simple Reflected Entity blockset components

Type	Name	Description
Input Ports	state	Handle
Output Ports	None.	
Parameters	entityName	HLA only: String
	entityId	DIS only: The entity ID
	location	Vector
	velocity	Vector
	acceleration	Vector
	orientation	TaitBryan
	rotVelocity	Vector

5.6.7. Reflected Entity Selector

The reflected entity selector is the mechanism through which you can select an entity from the reflected entity list. [Table 5-7](#) lists the input ports, output ports, and parameters for the reflected entity selector blockset.

Table 5-7: Reflected Entity Selector blockset components

Type	Name	Description
Input Ports	reflEntityList	A handle to a reflected entity list.
	selectionParameter	An index or string to use to select and entity from the reflected entity list.
Output Ports	state	A handle to the state of the entity selected.
Parameters	Selection Method	Specifies use of and index (Int) value or entity name (String) to select the entity.

5.6.8. Interaction Publisher

An interaction publisher sends interactions over the network. [Table 5-8](#) lists the input ports, output ports, and parameters for the interaction publisher blockset.

Table 5-8: Interaction Publisher blockset components

Type	Name	Description
Input Ports		
Output Ports	None defined.	The inputs depend on the interaction being sent.
Parameters	None.	

5.6.9. Received Interaction

A received interaction is an interaction received from the network. [Table 5-9](#) lists the input ports, output ports, and parameters for the received interaction blockset.

Table 5-9: Received Interaction blockset components

Type	Name	Description
Input Ports	interaction	A handle to the received interaction.
Output Ports	None defined.	The outputs depend on the interaction received.
Parameters	Interaction Type	The type of interaction, for example, fire or detonate.

5.6.10. Received Interaction List

The received interaction list is a list of interactions received from the network. [Table 5-10](#) lists the input ports, output ports, and parameters for the received interaction list blockset.

Table 5-10: Received Interaction List blockset components

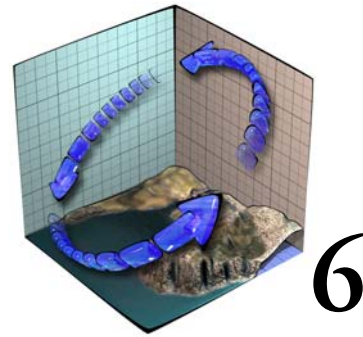
Type	Name	Description
Input Ports	exConn	A handle to an exercise connection through which to receive interactions.
Output Ports	recvIntList	A handle to the interaction list.
	count	A unsigned 32 bit integer that contains the number of interactions in the list.
Parameters	Interaction Type	The type of interaction, for example, fire or detonate.

5.6.11. Received Interaction Selector

The received interaction selector is the mechanism through which an application selects an interaction from the received interaction list. [Table 5-11](#) lists the input ports, output ports, and parameters for the received interaction selector blockset.

Table 5-11: Received Interaction Selector blockset components

Type	Name	Description
Input Ports	recvIntList	A handle to the received interaction list.
	index	An unsigned 32 bit integer with the index value to be used to select an interaction.
Output Ports	interaction	A handle to the interaction chosen from the list.
Parameters	None.	



Using MATLAB to Analyze Logger Data

This chapter explains how to import data saved by the MÄK Data Logger into MATLAB for analysis.

Importing Logger Data into MATLAB	6-2
Database Schema Used by the Logger SQL API.....	6-3
The entities Table.....	6-3
The aggregates Table	6-4
The aggregate_subordinates Table	6-7
The detonations Table.....	6-7
The fires Table.....	6-9

6.1. Importing Logger Data into MATLAB

Previous chapters have shown how to use the HLA/DIS Toolbox to enable MATLAB applications to participate in real-time simulations. You can also use MATLAB to analyze HLA or DIS simulation data that has been saved by data logging programs.

The HLA/DIS Toolbox includes two databases – *f18.sql* and *berkeley.sql* as examples of HLA and DIS data exported from the MÄK Data Logger. It also includes two example MATLAB programs, *f18DBdemo.m* and *berkeleydemo.m* as examples of importing Logger data into MATLAB programs. You can run both demonstration applications from *MAK_DB.m*.

“Database Schema Used by the Logger SQL API,” on page 6-3 describes the format of the tables created by the Logger.

i

To import data that has been exported from the MÄK Data Logger to a SQL database, you need the Mathworks Database Toolbox.

To import data into a MATLAB program:

1. Make a connection:

```
% Connect to database named 'f18'
conn = database('f18','root','');
ping(conn);
```

2. Query some data using SQL commands:

```
% Query entity attributes for all the entity entries.
entities = exec(conn,['select id,host,site,timestmp,longitude,
    latitude,altitude,position_x,position_y,position_z from entities']);
entities = fetch(entities);
close(conn);
```

3. Select the data you want to analyze:

```
% Grab individual data sets from the returned data
col = 1;
id      = entities.Data(:,col);col=col+1;
host    = entities.Data(:,col);col=col+1;
site    = entities.Data(:,col);col=col+1;
times   = entities.Data(:,col);col=col+1;
longitudes = entities.Data(:,col);col=col+1;
latitudes = entities.Data(:,col);col=col+1;
altitudes = entities.Data(:,col);col=col+1;
pos_x    = entities.Data(:,col);col=col+1;
pos_y    = entities.Data(:,col);col=col+1;
pos_z    = entities.Data(:,col);col=col+1;
```

4. Perform your analysis on the data.

6.2. Database Schema Used by the Logger SQL API

The data exported to a Logger database is structured as five independent tables, named entities, aggregates, aggregate_subordinates, detonations, and fires. This section describes the contents of these tables.

6.2.1. 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 6-1: 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	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.)
18	altitude	
19	velocity_x	
20	velocity_y	
21	velocity_z	
22	velocity_north	
23	velocity_east	
24	velocity_down	

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

Column	Name	Definition
25	acceleration_x	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 Topo graphic reference frame attached to the entity.)
26	acceleration_y	
27	acceleration_z	
28	acceleration_north	
29	acceleration_east	
30	acceleration_down	
31	orient_psi	Orientation. The first three entries are the TaitBryan angles (in radians). The next three entries are the conventional Heading, Pitch, Roll (in degrees).
32	orient_theta	
33	orient_phi	
34	heading_deg	
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.

6.2.2. 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 6-2: Contents of aggregates table (Sheet 1 of 3)

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.

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

Column	Name	Definition
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	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).
21	type_extra	
22	position_x	
23	position_y	
24	position_z	
25	latitude	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.)
26	longitude	
27	altitude	
28	velocity_x	
29	velocity_y	
30	velocity_z	
31	velocity_north	
32	velocity_east	
33	velocity_down	

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

Column	Name	Definition
34	acceleration_x	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.)
35	acceleration_y	
36	acceleration_z	
37	acceleration_north	
38	acceleration_east	
39	acceleration_down	Orientation. The orient entries are the TaitBryan angles (in radians). The next three entries are the conventional Heading, Pitch, Roll (in degrees).
40	orient_psi	
41	orient_theta	
42	orient_phi	
43	heading_deg	
44	pitch_deg	Angular velocity (rate of rotation). The three entries are the x, y, z components of the vector in Bodyreference frame.
45	roll_deg	
46	ang_velocity_x	
47	ang_velocity_y	Arrival time of the message.
48	ang_velocity_z	
49	timestamp	
50	timestamp_type	Timestamp type.
51	appearance	OR-ed entity appearance enumerators.
52	dr_algorithm	Dead-reckoning algorithm enumerator.
53	marking	Marking text.

6.2.3. 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 6-3: 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.

6.2.4. 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 6-4: 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 6-4: 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 down components 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	Warhead type enumerator.
38	munition_type_extra	
39	warhead_type	
40	fuse_type	Fuse type enumerator.

Table 6-4: 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.

6.2.5. The fires Table

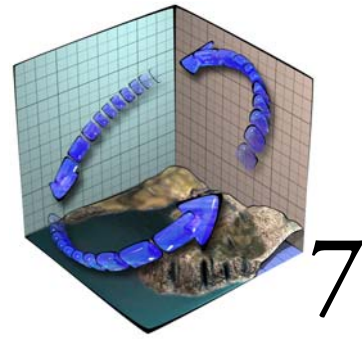
In HLA the fires table contains simulation data corresponding to the RPR FOM 1.0 WeaponFire interactions (or to any of the subclasses of WeaponFire). In DIS this table contains simulation data corresponding to Fire PDUs.

Table 6-5: 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 6-5: 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.



FOM Agility

This chapter describes FOM agility and FOM mapping.

FOM Agility and FOM Mapping	7-2
-----------------------------------	-----

7.1. FOM Agility and FOM Mapping

In addition to its built-in support for the RPR FOM, the HLA/DIS Toolbox can be configured to work with other FOMs, by using VR-Link FOM Mappers to define mappings between VR-Link's protocol-independent interface and the objects, interactions, parameters, and attributes defined in your FOM. This ability to work with a variety of FOMs is called FOM agility.

Figure 7-1 illustrates VR-Link's basic architecture for FOM agility.

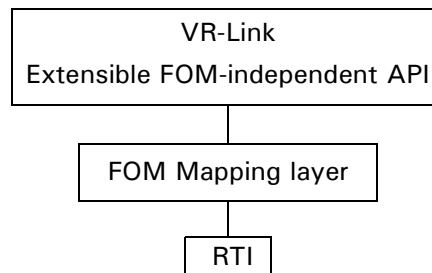


Figure 7-1. FOM mapping architecture

The top-level FOM-independent API lets you set and inspect the state of local and remote objects and the data contained in locally generated or received interactions, without regard to how the data is actually represented in a FOM. This top-level API includes the object publishers, reflected objects, reflected object lists, state repositories, and interaction classes.

The FOM Mapping layer routes and converts data between the FOM-independent API and the FOM representations you are using for a particular federation execution.

There are two broad categories of FOM modifications that developers might implement:

- ♦ FOMs that contain only those concepts that are already present in VR-Link's top-level API, but which may represent the concepts differently from the way they are represented in the HLA/DIS Toolbox API.
- ♦ FOMs that include concepts that are not present in VR-Link's API.

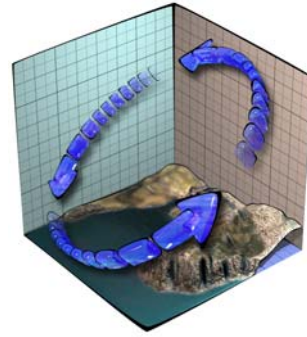
If you are implementing FOMs that have the same concepts as VR-Link's API, you can switch from FOM to FOM just by modifying the FOM Mapping layer, which you can do without affecting the top-level API. This means that application code written to the top-level API does not need to change when switching from FOM to FOM.

For example, in one FOM (such as the RPR FOM), position data may be represented in geocentric coordinates in an attribute named "Position". In another, it might be represented in topographic coordinates in an attribute named "Location". In either case, position data is available to application code through VR-Link in the manner dictated by its FOM-independent API – as geocentric data. FOM Mapping code takes care of converting from the various FOMs' representations to the representation chosen by VR-Link.

If you are using FOMs that contain concepts not found in the VR-Link API, you must follow a two-step FOM-configuration process.

1. Extend the top-level VR-Link API, by deriving new classes from our base state repository, publisher, reflected object, or reflected object list classes to include your new concepts. Once you have done this, your code is part of the FOM-independent API.
2. Create mappings from arbitrary FOM representations of these concepts to your new API extensions, just as if your extensions were part of the standard VR-Link API. (Typically, you will choose representations for your API that closely match the representation in a particular FOM, meaning that the mapping code will be fairly straightforward).

For more information about FOM Mapping and details about how to write a FOM Mapper, see *VR-Link Reference Manual*. The HLA/DIS Toolbox license entitles you to use VR-Link to create FOM Mappers.



MÄK Products Glossary

This glossary defines terms used in MÄK product documentation.

absolute dead reckoning

A method of [dead-reckoning](#) in which an application uses the time stamps contained within state updates if the sender is using absolute time stamping. Contrast with [relative dead reckoning](#).

aggregate

The combination of individual entities, aggregates, or both into a single object. For example, an organizational group such as a platoon.

API

Application Programming Interface.

articulated part

A part of an entity that is capable of movement relative to the entity, such as a turret or gun.

attached part

An object that is attached to another object, such as a rocket attached to a jet.

body coordinate frame

A coordinate framework centered on an entity. To represent the orientation of an entity, an application uses Euler angles or a rotation matrix that rotates from a reference frame to the body coordinate frame.

In the 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 an application clips objects.

culling

The process of discarding database objects which are not within view, and therefore need not be rendered and displayed.

Cartesian coordinates

A system for indicating location by means of three planes intersecting at right angles to one another at the origin.

dead-reckoning

A process by which an application calculates the expected location of an entity during periods between state updates, based on velocity, acceleration, and rotational velocity.

Defense Modeling and Simulation Office

The executive secretariat for the Executive Council on Modeling and Simulation (EXCIMS), which provides a full-time focal point for information concerning Department of Defense modeling and simulation (M&S) activities. Currently the DMSO promulgates M&S policy, initiatives, and guidance to promote cooperation among DoD components to maximize efficiency and effectiveness.

DIS

[Distributed Interactive Simulation](#).

DIS data packets

See [Protocol Data Unit](#).

Distributed Interactive Simulation

The DIS protocol is a set of standards that govern how participating applications share information about the virtual world. The protocol specifies a set of packets, called protocol data units (PDUs), that communicate this information. Each PDU identifies the sender and contains other information depending on the PDU type. The DIS protocol also specifies when and how frequently PDUs are sent.

The DIS protocol has been superseded by the [High-Level Architecture](#) for use by the Department of Defense.

DMSO

[Defense Modeling and Simulation Office.](#)

emitter

A simulated device on an entity that emits electromagnetic radiation, for example, radar.

entity

An element in a simulation, such as a vehicle or a person, that is represented in the simulation through the issuance of state messages.

entity maintenance

A process by which the 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 HLA/DIS Toolbox 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. HLA/DIS Toolbox 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 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 an application approximates the location of an object based on the local time of receipt of the last state update message.

RTI Initialization Data (RID)

The initialization data required by the RTI for operation.

Data required by an RTI during initialization, independent of the FOM being used. RID data is usually dependent on a specific implementation of the RTI.

RPR FOM

[Realtime Platform Reference FOM](#).

RTI

[Run-Time Infrastructure](#).

Run-Time Infrastructure

A library and other supporting software that implements the HLA interface specification. All federates communicate with one another in an HLA environment through RTI functions.

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

In VR-Forces, simulation time is used in dead-reckoning of remote entities and thresholding of local entities. Typically, simulation time is set once during each iteration of the application's main simulation loop so that all entities are dead-reckoned based on the same value of current time.

SISO

Simulation Interoperability Standards Organization

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

UDP port

A network channel through which an application sends and receives data for DIS exercises.

Universal Transverse Mercator

In general, a non-cartesian coordinate system in which the X, Y, and Z correspond to easting (nearly east), northing (nearly north), and height above an ellipsoid that approximates the surface of the earth. When in UTM mode (the default), the Stealth uses an offset UTM coordinate system, one in which coordinates are expressed with respect to an origin located at the latitude and longitude specified in a configuration file. Typically, this will be the location of the origin of your terrain database.

UTM coordinates

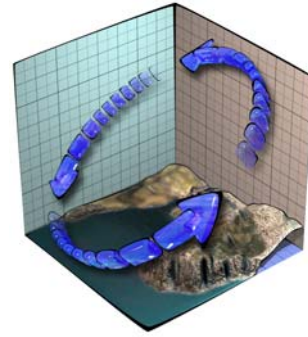
[Universal Transverse Mercator](#) coordinates.

view control messages

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

view floor

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

A

- adding
 - callback 4-20
 - new license file 2-9
 - translators 4-21
- addInteractionCallback() function 5, 6, 20
- addTranslator() function 21
- Adobe Acrobat 1-viii
- advanceTime() function 23
- advancing
 - time 4-23
- aggregate_subordinates table 6-7
- aggregates table 6-4
- application
 - example 4-4
- application number 3-3, 4-12
- attribute
 - entity 3-4
 - data types 4-16
 - list of 4-15
 - exercise connectionn 4-12
 - getting
 - entity 4-14
 - publishing in Simulink 5-5
 - setting
 - entity 4-16

B

- block
 - Coordinate Conversion 5-7
 - Entity Publisher 5-6
 - Exercise Connection 5-6
 - Reflected Entity 5-10

- Reflected Entity List 5-9
- Reflected Entity Selector 5-9
- Simple Entity Publisher 5-6
- Simple Reflected Entity 5-10
- Wall Clock Timer 5-7, 5-10

blockset

- description
 - entity publisher 5-12
 - exercise connection 5-11
 - interaction publisher 5-16
 - received interaction 5-16
 - received interaction list 5-17
 - received interaction selector 5-17
 - reflected entity 5-14
 - reflected entity list 5-14
 - reflected entity selector 5-16
 - simple entity publisher 5-13
 - simple reflected entity 5-15
- list of 5-3
- utility 5-4

C

- callback 3-5, 4-14
 - adding 4-20
 - example 4-6
 - function 3-5
 - removing 4-20
- callback function 3-2
- Cartesian coordinate system 4-27
- class
 - DtExerciseConn* 5, 11, 12
 - DtReflectedEntity* 17
 - DtReflectedEntityList* 17
 - DtTimeManager* 12, 23

- concept
 - callbacks 3-5
 - coordinate systems 3-5
 - dead-reckoning 3-6
 - exercise connection 3-2
 - smoothing 3-6
 - VR-Link 3-2
- configuration file
 - RTI 2-13
- configuring
 - system for MÄK RTI 2-13
- connect() function 12
- connecting
 - exercise to 3-2
 - to exercise 3-2, 4-11
- connection status 4-12
- constructor
 - DtExerciseConn 4-11
- conventions
 - manual 1-xii
- converting
 - coordinate systems 3-5, 4-32
 - coordinates 4-27
 - orientation 4-27
 - vectors 4-27
- coordinate
 - conversion 3-5
 - system 3-5
- Coordinate Conversion block 5-7
- coordinate system
 - Cartesian 4-27
 - concepts 3-5
 - conversion 4-27
 - converting between 4-32
 - geocentric 4-28
 - right-hand 4-27
 - topographic 4-29
 - UTM 4-30
- creating
 - entity publisher in Simulink 5-6
 - exercise connection in Simulink 5-6, 5-8
 - Time Manager 4-23

D

- data
 - importing 6-2
 - publishing entity 4-14
 - sending and receiving 3-2
- data model 1-4

- data type
 - attributes
 - entity 4-16
 - explanation of 5-4
- database
 - importing Logger 6-2
- dead-reckoning 3-6, 4-13
 - concepts 3-6
- decoding
 - interactions 4-21
- definition of 5-4
- destroy() function 13, 17, 19
- destroying
 - entity publisher 4-17
 - exercise connection 4-13
 - reflected entity list 4-19
- detonations table 6-7
- directory
 - tree, VR-Link 2-4
- directory structure 2-4
- DIS 1-2
 - connecting to 4-6
 - connecting to exercise 4-11
 - entity identifier 3-5
 - enumerations document 1-5
 - introduction to 1-5
 - object ID 3-5
- disconnect() function 13
- draining input 3-2, 4-14
- draining input.tick() function 4-17
- drainInput() function 3-5, 6, 9, 14, 20
 - example 6
- DtConvertCoord() function 32
- DtDebug() function 26
- DtEntityPublisher() function 14
- DtExerciseConn* class 5, 11, 12
- DtExerciseConn() function 6, 11
- DtGeocToTopoTransform() function 29, 33
- DtInitGeodSphere() function 28
- DtInitUTM() function 30
- DtLoadEnums() function 26
- DtLoadLib() function 26
- DtReflectedEntity* class 17
- DtReflectedEntityList* class 17
- DtReflectedEntityList() function 4-17
- DtSupportedProtocols() function 26
- DtTimeManager* class 12, 23
- DtTimeManager() function 23
- DtTopoToGeocTransform() function 29, 33
- DtUnLoadLib() function 26

DtValidProtocol() function 26

E

entities table 6-3

entity

attributes 3-4

getting 4-14

list of 4-15

publishing in Simulink 5-5

setting 4-16

dead-reckoning and smoothing 3-6

enumeration 4-14

ID 4-15

identifier 3-5

list of remote 5-9

local 3-2

name 4-14, 4-15

publisher 3-2

publishing data 4-14

reflected 3-4, 4-18

remote 3-2

listening for 5-8

tracking 4-17

selecting from reflected entity list 5-9

entity publisher 3-2, 3-4

attributes

setting 4-16

blockset 5-12

creating in Simulink 5-6

destroying 4-17

exercise connection 4-15

handle 4-15

properties

getting 4-14

simple 5-13

ticking 4-17

Entity Publisher block 5-6

enumeration

document 1-5

environment variable

MAKLMGRD_LICENSE_FILE 2-8

RTI_CONFIG 2-13

event 3-2

event ID 4-12

example

listen 4-4

send 4-7

example application 4-4

exercise 1-2

site

application ID 4-6

exercise connection 3-2, 4-6, 4-11

attributes 4-12

setting 4-13

blockset 5-11

creating in Simulink 5-6, 5-8

destroying 4-13

entity publisher 4-15

specifying for reflected entity list 4-17

Exercise Connection block 5-6

exercise ID 3-3, 4-12

getting 4-13

exercise name

getting 4-12

specifying 3-3, 4-11

F

FDD file 1-4, 2-13

FED file 1-4, 2-13

introduction to 1-4

specifying 3-3, 4-11

federate 1-2, 4-6

specifying 3-3, 4-11

time constrained 4-25

time regulating 4-24

federate name

getting 4-12

federation execution 1-4, 4-6

data model 1-4

specifying name of 3-3, 4-11

Federation Execution Data file 1-4

Federation Object Model 1-4

introduction to 1-4

file

FED 1-4, 2-13

installed 2-4

license 2-7

RID 2-13

rti.mtl 2-13

RTI.RID 2-13

VR-Link.fed 1-4

fires table 6-9

FLEXlm

directory location 2-6

installing 2-6

license file 2-7

license manager 2-5

multiple licenses 2-9

- troubleshooting 2-10
- FOM 1-4, 2-13
 - introduction to 1-4
 - mapping 7-2
 - support 1-4
 - switching between 7-2
- FOM interaction class
 - list of 13
- FOM Mapper
 - specifying in exercise connection 3-3, 4-11
- function
 - addInteractionCallback() 5, 6, 20
 - addTranslator() 21
 - advanceTime() 23
 - callback 3-2, 3-5
 - connect() 12
 - destroy() 13, 17, 19
 - disconnect() 13
 - drainInput() 3-5, 6, 9, 14, 20
 - example 6
 - DtConvertCoord() 32
 - DtDebug() 26
 - DtEntityPublisher() 14
 - DtExerciseConn() 6, 11
 - DtGeocToTopoTransform() 29, 33
 - DtInitGeodSphere() 28
 - DtInitUTM() 30
 - DtLoadEnums() 26
 - DtLoadLib() 26
 - DtReflectedEntityList() 4-17
 - DtSupportedProtocols() 26
 - DtTimeManager() 23
 - DtTopoToGeocTransform() 29, 33
 - DtUnLoadLib() 26
 - DtValidProtocol() 26
 - generateInteraction() 4, 19
 - get() 12, 14, 19, 23, 24
 - getEntityByIndex() 18
 - getEntityByName() 18
 - removeInteractionCallback() 20
 - removeTranslator() 22
 - send() 19, 20
 - set() 13, 16, 19, 24
 - startConstraining() 25
 - startRegulating() 24
 - stopConstraining() 25
 - stopRegulating() 24
 - tick() 17, 12
 - useRealTime() 13
 - useSimTime() 13

G

- generateInteraction() function 4, 19
- geocentric coordinate 3-5
 - right-hand 4-27
- geodetic coordinates 4-28
- get() function 12, 14, 19, 23, 24
- getEntityByIndex() function 18
- getEntityByName() function 18
- getting
 - attributes
 - exercise connection 4-12
 - entity properties 4-14
 - properties
 - reflected entity list 4-19
 - Time Manager 4-24
 - reflected entity state 4-18
 - update messages 4-14

H

- handle 5-4
 - entity publisher 4-15
 - exercise connection 4-12
 - object 3-5
- HLA 1-2, 1-4
 - connecting to 4-6
 - object handle and name 3-5
 - object identifier 3-5
 - time management 4-23
 - time manager 4-12
- HLA 1.3 1-4
- HLA 1516 1-4
- HLA13 4-11
- HLA1516 4-11

I

- ID
 - object 3-5
- identifying
 - objects 3-5
- IEEE
 - 1278.1-1995 1-5
 - Web site 1-5
- importing
 - data 6-2
- index
 - reflected entity list 4-18
- information

- additional 1-5
- input
 - reading network 3-2
- installing
 - FLEXlm software 2-6
 - HLA/DIS Toolbox 2-2
 - RTI 2-13
- interaction 3-2, 4-9
 - adding support for new 4-21
 - callback 4-20
 - list of 13
 - publisher 3-4
 - responding to 3-5
 - sending 4-20
- interaction class
 - list off 4-12
- interaction publisher
 - blockset 5-16
- interoperability
 - simulation tools 1-4

L

- library
 - exercise connection 4-12
 - loading, unloading, verifying 4-26
- license file 2-7
 - adding new 2-9
- license server
 - running 2-9
 - status 2-9
 - stopping 2-9
 - troubleshooting 2-10
- link compatibility 1-5
- listen
 - example application 4-4
- lmdown program 2-9
- lmstat program 2-9
- loading
 - libraries 4-26
- Logger recording
 - importing 6-2

M

- MÄK RTI
 - configuring 2-13
- MAKLMGRD_LICENSE_FILE environment
 - variable 2-8
- manual

- conventions 1-xii
- mapping
 - FOM 7-2
- M-code 4-3
- message
 - update, sending 4-17
- model
 - publishing attributes 5-5
- mxClassId 4-21

N

- name
 - object 3-5
 - reflected entity list 4-18
- network
 - getting updates from 4-14
 - reading 3-2

O

- object
 - handle 3-5
 - identification 3-5
 - identifying 3-5
 - name 3-5
- orientation
 - conversion 4-27

P

- path
 - specifying for RTI 2-13
- PDU
 - described 1-5
- port
 - getting 4-13
 - specifying
 - specifying
 - UDP port 3-3, 4-12
- Portable Document Format (PDF) 1-viii
- position
 - actual and dead-reckoned 3-6
- processing
 - incoming interactions 3-5
- properties
 - getting
 - Time Manager 4-24
 - setting

- Time Manager 4-24
- protocol
 - exercise connection 4-12
 - specifying 4-11
 - supported 1-4
- Protocol Data Unit 1-5
- publisher
 - entity
 - creating in Simulink 5-6
 - interaction 3-4, 5-16
- publishing
 - entity data 4-14

R

- real time 4-12, 4-13
 - using 4-13
- Real-Time Platform Reference FOM 1-4
- received interaction
 - blockset 5-16
- received interaction list
 - blockset 5-17
- received interaction selector
 - blockset 5-17
- receiving
 - data 3-2
- reference ellipsoid 4-28
- reflected entity 3-2, 5-8
 - blockset 5-14
 - getting state 4-18
 - simple 5-15
- Reflected Entity block 5-10
- reflected entity list 3-4, 4-6, 4-17, 5-9
 - blockset 5-14
 - destroying 4-19
 - index and name 4-18
 - properties
 - getting 4-19
 - selecting entities from 5-9
- Reflected Entity List block 5-9
- reflected entity selector
 - blockset 5-16
- Reflected Entity Selector block 5-9
- remote
 - entity 3-2
 - entity list 5-9
- remote entity
 - listening for 5-8
 - tracking 4-17
- removeInteractionCallback() function 20

- removeTranslator() function 22
- removing
 - callback 4-20
 - translators 4-22
- RID file 2-13
- rid.mtl* file 2-13
- right-hand geocentric coordinates 4-27
- RPR FOM 1-4, 3-5
 - description of 1-4
 - specifying in exercise connection 3-3, 4-11
- RTI 1-2, 1-5, 2-13
 - definition of 2-13
 - introduction to 1-5
 - link compatibility 1-5
- RTI_CONFIG 3-3
- RTI_CONFIG environment variable 2-13
- RTI.rid* file 2-13
- runLm program 2-9
- running
 - license server 2-9
- Run-Time Infrastructure
 - introduction to 1-5
 - See also RTI 1-5

S

- selecting
 - entity from reflected entity list 5-9
- send
 - example application 4-7
- send() function 19, 20
- sending
 - data 3-2
 - entity data 4-14
 - interaction 4-20
 - update messages 4-17
- set() function 13, 16, 19, 24
- setting
 - attributes
 - entity 4-16
 - exercise connection 4-13
 - DIS time out rate 4-19
 - properties
 - Time Manager 4-24
 - simulation time 4-13
- shutting down
 - license server 2-9
- simple entity publisher
 - blockset 5-13
- Simple Entity Publisher block 5-6

- simple reflected entity
 - blockset 5-15
 - Simple Reflected Entity block 5-10
 - simpleListen.m 4-4
 - simpleTalk.m 4-7
 - simTime 4-13
 - simulation
 - standard 1-4
 - time. *See* VR-Link
 - simulation time
 - simulation clock 3-2
 - Simulation Interoperability Standards Organization 1-5
 - simulation time 4-6, 4-12, 4-13
 - setting 4-13
 - synchronizing with wall clock time 5-7, 5-10
 - Simulink
 - publishing model 5-5
 - SISO 1-4, 1-5
 - SISO DLC 1516 API 1-5
 - site-application-entity identifier 3-5
 - siteID 3-3, 4-12
 - smoothing 3-6
 - concepts 3-6
 - specifying for reflected entity list 4-17
 - specification
 - HLA 1-4
 - specifying
 - exercise ID, site ID, and application number 3-3, 4-12
 - exercise name 3-3, 4-11
 - FED file in exercise connection 3-3, 4-11
 - federate name in exercise connection 3-3, 4-11
 - FOM Mapper 3-3, 4-11
 - protocol in exercise connection 4-11
 - RPR FOM 3-3, 4-11
 - standards
 - simulation 1-4
 - startConstraining() function 25
 - starting
 - time management
 - constraint 4-25
 - regulation 4-24
 - startRegulating() function 24
 - state information
 - entity 3-2
 - state updates
 - responding to 3-5
 - status
 - license server 2-9
 - stopConstraining() function 25
 - stopping
 - license server 2-9
 - time management
 - constraint 4-25
 - regulation 4-24
 - stopRegulating() function 24
 - supported protocols 1-4
 - switching
 - between FOMs 7-2
 - synchronizing
 - wall clock time and simulation time 5-7, 5-10
- ## T
- talk
 - example 4-7
 - tick() function 9, 17, 12
 - ticking
 - entity publisher 4-17
 - time
 - advancing 4-23
 - simulation 3-2
 - synchronizing wall clock and simulation 5-7, 5-10
 - time management 4-23
 - advancing time 4-23
 - definition of 4-23
 - time constraint
 - starting 4-25
 - stopping 4-25
 - time regulation
 - starting 4-24
 - stopping 4-24
 - Time Manager
 - creating 4-23
 - properties
 - getting 4-24
 - setting 4-24
 - time manager 4-12
 - time out rate
 - setting 4-19
 - topographic coordinate system 4-29
 - and UTM coordinate system, compared 4-31
 - translator
 - interaction 4-21
 - removing 4-22
 - types of 4-21
 - types supported 4-21
 - troubleshooting

FLEXlm 2-10

RTI configuration file 2-13

U

UDP port

specifying 3-3, 4-12

Universal Transverse Mercator 4-30

unloading

libraries 4-26

update

getting 4-14

sending 4-17

update message

sending entity 4-14

useRealTime() function 13

useSimTime() function 13

using

real time 4-13

simulation time 4-13

Utility Blockset 5-7, 5-10

Utility blockset 5-4

UTM coordinate system 4-30

and topographic system, compared 4-31

V

variable

environment 2-8

vector

conversion 4-27

verifying

libraries 4-26

VR-Link 1-2, 1-3, 4-3

concepts 3-2

files 2-4

simulation time 4-9

VR-Link.fed 1-4

VR-Link.fed file 1-4

W

wal clock time 4-13

wall clock time

synchronizing with real time 5-7, 5-10

Wall Clock Timer block 5-7, 5-10

X-Y-Z

XML



Link - Simulate - Visualize

TBX-1.0-1-050715