MAK TECHNOLOGIES

MAK Legion Users Guide

Version 1.1

ST Engineering
North America

MAK Technologies
150 Cambridge Park Drive, 3rd Floor
Cambridge, MA 02140 USA

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

info@mak.com

www.mak.com

Revision LEG-1.1-1-210505

Contents

Chapter 4. Configuring MAK Legion

Chapter 5. MAK Legion Monitor

Chapter 6. The MAK Legion SDK

Chapter 7. Extending the Object Model

Index

Preface

This manual is written for persons who will use or administer MAK Legion.

Please see release documentation for updates to this manual and the latest information about your version of MAK Legion.

How This Manual is Organized

The chapters are organized as follows:

Chapter 1, *Introduction* introduces you to MAK Legion.

Chapter 2, *Installing MAK Legion* explains how to install MAK Legion and related software.

Chapter 3, *Running the MAK Legion Server and Clients* is a brief explanation of how to run the MAK Legion Server and MAK Legion clients.

Chapter 4, *Configuring MAK Legion* explains the MAK Legion configuration file.

Chapter 5, *MAK Legion Monitor* describes the MAK Legion Monitor, an application that gives insights into the simulations that MAK Legion is working with.

Chapter 6, *The MAK Legion SDK* introduces you to the MAK Legion SDK and how to use it to connect to simulations.

Chapter 7, *Extending the Object Model* introduces you to the MAK Legion object model.

MAK Products

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

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

- MAK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MAK RTI is optimized for high performance. It has an API, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface (the RTI Assistant) that helps users with configuration tasks and managing federates and federations.
- VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a GUI, that gives you 2D and 3D views of a simulated environment.

  You can create and view entities, aggregate them into hierarchical units, assign tasks, set state parameters, and create plans that have tasks, set statements, and conditional statements. You can simulate using entity-level modeling, which focuses on the actions of individual people and vehicles, and aggregate-level modeling, which focuses on the interaction of large hierarchical units.

  VR-Forces also functions as a plan view display for viewing remote simulation objects taking part in an exercise. Using the toolkit, you can extend the VR-Forces application or create your own application for use with another user interface.

- VR-Vantage™. VR-Vantage is a line of products designed to meet your simulation visualization needs. It includes three end-user applications (VR-Vantage Stealth, VR-Vantage PVD, and VR-Vantage IG) and the VR-Vantage Toolkit.
  - VR-Vantage Stealth displays a realistic, 3D view of your virtual world, a 2D plan view, and an exaggerated reality (XR) view. Together these views provide both situational awareness and the big picture of the simulated world. You can move your viewpoint to any location in the 3D world and can attach it to simulation objects so that it moves as they do.
  - VR-Vantage IG is a configurable desktop image generator (IG) for out the window (OTW) scenes and remote camera views. It has most of the features of the Stealth, but is optimized for its IG function.

  - VR-Vantage PVD provides a 2D plan view display. It gives you the big picture of the simulated world.
  - SensorFX. SensorFX is an enhanced version of VR-Vantage that uses physics based sensors to view terrain and simulation object models that have been materially classified. It is built in partnership with JRM Technologies.
  - The VR-Vantage Toolkit is a 3D visual application development toolkit. Use it to customize or extend MAK's VR-Vantage applications, or to integrate VR-Vantage capabilities into your custom applications. VR-Vantage is built on top of OpenSceneGraph (OSG). The toolkit includes the OSG version used to build VR-Vantage.

- MAK Data Logger. The Data Logger, also called the Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal and can quickly jump to areas of interest. The Logger has a GUI and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- VR-Exchange™. VR-Exchange allows simulations that use incompatible communications protocols to interoperate. For example, within the HLA world, using VR-Exchange, federations using the HLA RPR FOM 1.0 can interoperate with simulations using RPR FOM 2.0, or federations using different RTIs can interoperate. VR-Exchange supports HLA, TENA, and DIS translation.
- VR-TheWorld™ Server. VR-TheWorld Server is a simple, yet powerful, web-based streaming terrain server, developed in conjunction with Pelican Mapping. Delivered with a global base map, you can also easily populate it with your own custom source data through a web-based interface. The server can be deployed on private, classified networks to provide streaming terrain data to a variety of simulation and visualization applications behind your firewall.
- DI-Guy™. DI-Guy is an SDK that allows you to embed the DI-Guy library in your real-time application and populate your world with lifelike human characters. DI-Guy provides high-fidelity human characters that move realistically, respond to simple high-level commands, and travel throughout the environment as directed by the hosting visual application. It comes with thousands of ready-to-use characters, appearances, and motions. DI-Guy character simulation is an integral part of all MAK's visual applications, including VR-Engage, VR-Forces, and VR-Vantage.
- RadarFX. RadarFX is a client-server application that simulates synthetic-aperture radar (SAR). The server application, which is based on VR-Vantage and SensorFX, loads a terrain database and, optionally, connects to simulations. A client application requests SAR images from the server. RadarFX includes a sample client application.

- VR-Engage. VR-Engage is a multi-role virtual simulator that lets users play the role of a first person human character, a ground vehicle driver, gunner or commander, a sensor operator, or the pilot of a fixed wing aircraft or helicopter. It incorporates the VR-Force simulation engine and the VR-Vantage graphics rendering capabilities.
- WebLVC Server. WebLVC Server implements the server side of the WebLVC protocol so that web-based simulation federates can participate in a distributed simulation. It is part of the WebLVC Suite, which includes the server and several sample applications that work with VR-Forces and VR-Vantage.
- MAK Legion. MAK Legion is a collection of applications and an SDK designed to support simulations with millions of simulation objects.

How to Contact Us

For MAK Legion technical support, information about upgrades, and information about other MAK 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

MAK web site home page:	www.mak.com
License key requests:	www.mak.com/support/get-licenses
Technical support:	www.mak.com/support/incidents
Product version and platform information:	www.mak.com/support/product-versions
For the free, unlicensed MAK RTI:	www.mak.com/support/bonus-material

Post

Send postal correspondence to:	MAK Technologies 150 CambridgePark Drive, 3rd Floor Cambridge, MA, USA 02140

When requesting support, please tell us the product you are using, the version, and the platform on which you are running.

Document Conventions

This manual uses the following typographic conventions:

Monospaced	Indicates commands or values you enter.
Monospaced Bold	Indicates a key on the keyboard.
Monospaced Italic	Indicates command variables that you replace with appropriate values.
Blue text	A hypertext link to another location in this manual or another manual in the documentation set.
{ }	Indicates required arguments.
[ ]	Indicates optional arguments.
|	Separates options in a command where only one option may be chosen at a time.
(|)	In command syntax, indicates equivalent alternatives for a command-line option, for example, **(-h | --help)**.
/	Indicates a directory. Since MAK products run on both Linux 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.
Italic	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**.
	Click the icon to run a tutorial video in the default browser.
	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 MAK Legion home directory. For example, the directory *1.1/doc* appears in the text as *./doc*.

Mouse Button Naming Conventions

An instruction to click the mouse button refers to clicking the primary mouse button, usually the left button for right-handed mice and the right button for left-handed mice. The context-sensitive menu, also called a popup menu or right-click menu, refers to the menu displayed when you click the secondary mouse button, usually the right button on right-handed mice and the left button on left-handed mice.

Third Party Licenses

MAK software products may use code from third parties. This section contains summary license information and where required, specific license documentation required by these third parties.

The following libraries are covered by the GNU Lesser General Public License (LGPL) license:

- CIGI
- DevIL
- GEOS
- GStreamer
- LibIconV
- LibWebSockets
- OPCODE
- OpenAL
- OSG
- OsgEarth
- Pthread
- Qt
- Qwt

The following libraries are covered by the ZLib license:

- Bullet Physics
- LibPNG
- LibSDL
- Minizip
- Zlib

The following libraries are covered by the MIT license:

- Expat
- FreeGLUT
- GDAL
- GeoTiff
- HarfBuzz
- LibCURL
- LibSquish
- LibXML
- LUA
- LuaBind
- Proj
- SMHasher
- TCLAP

The following libraries are covered by the BSD license:

- FreeType
- GLEW
- JPEG
- LibTiff
- Protobuf
- PCRE

The following libraries are covered by the commercial licenses:

- FlexLM
- Gameware
- GLStudio
- JRM Technologies SenSimRT and SigSimRT
- OpenFlight
- SpeedTree
- Sundog Triton
- Sundog SilverLining
- CDB API

The following libraries are covered by custom licenses:

- NVidia CG Toolkit
- FreeImage (FreeImage Public License)
- Boost (Boost Software License)
- Poco (Boost Software License)

COLLADA DOM is covered by the SCEA Shared Source license.

Thread Building Blocks is covered by the GPL license:

SQLLite is in the public domain.

Boost License

VR-Link, and all MAK software that uses VR-Link uses some code that is distributed under the Boost License. All header files that contain Boost code are properly attributed. The Boost web site is: www.boost.org.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libXML and libICONV

VR-Link and all MAK software that uses VR-Link, links in libXML and libICONV. On some platforms the compiled libraries and header files are distributed with MAK Products. MAK has made no modifications to these libraries. For more information about these libraries please see the following web sites:

- The LGPL license is available at: http://www.gnu.org/licenses/lgpl.html.
- Information about IconV is at: http://www.gnu.org/software/libiconv/.
- Information about LibXML is at: http://xmlsoft.org/.

1. Introduction

The MAK Legion toolkit is an SDK and several applications that work together to provide a high-performance interoperability infrastructure capable of support simulations with millions of simulation objects.

1.1. Overview

The MAK Legion toolkit is an SDK and several applications that work together to provide a high-performance interoperability infrastructure. The key components are the Legion Data Store SDK, Legion Network SDK, and the Legion Server. MAK Legion provides a common object model library, but allows users to extend and/or replace the library, which is separate from the framework implementation.

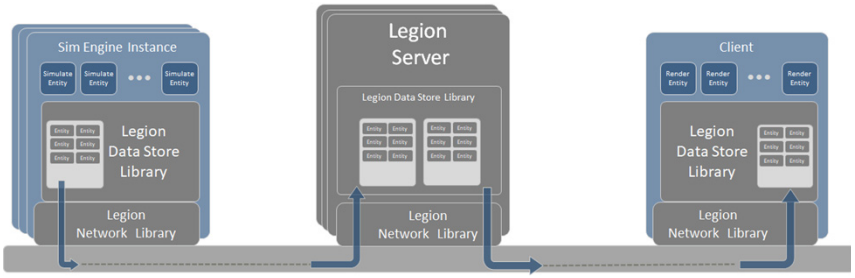

Figure 1-1. MAK Legion topology

The Legion framework has three main components:

- The Data Store Library. The Data Store Library stores all simulation data and provides a thread-safe API for getting and setting the data.
- The Network Library. The Network Library efficiently communicates state updates and events between participating applications.
- The Server. The Server maintains and serves authoritative state for entities based on client interest.

The two libraries (Data Store and Network) are what you would use to link your participating federate into a simulation with potentially millions of entities.

1.1.1. MAK Legion Data Store Library

The Legion Data Store Library addresses an important component of scalability: the ability of simulations to massively increase the number of entities each simulation engine can simulate by parallelizing access to the entity state and interaction data store.

Many traditional simulation engines are not able to take full advantage of today's multi-core hardware. Such systems use a single simulation execution thread which executes one task at a time (Figure 1-2).

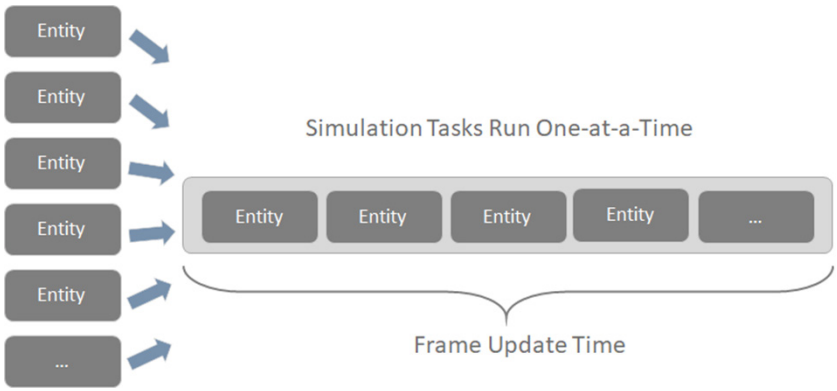

Figure 1-2. Single thread

A more efficient approach is to process entity tasks in parallel, taking better advantage of multiple cores and allowing the application to represent many more entities for each frame.

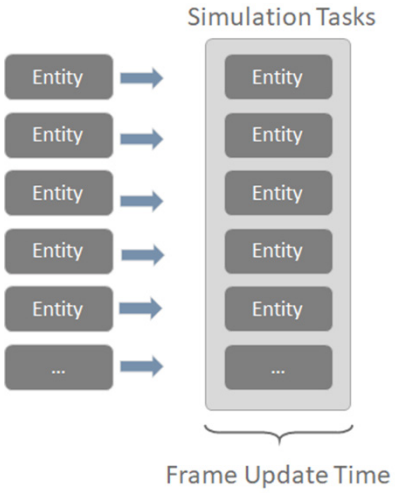

Figure 1-3. Parallel processing

1.1.2. MAK Legion Network Library

The MAK Legion Network Library handles the communication of data from the Data Store to the network and from the network to the Data Store.

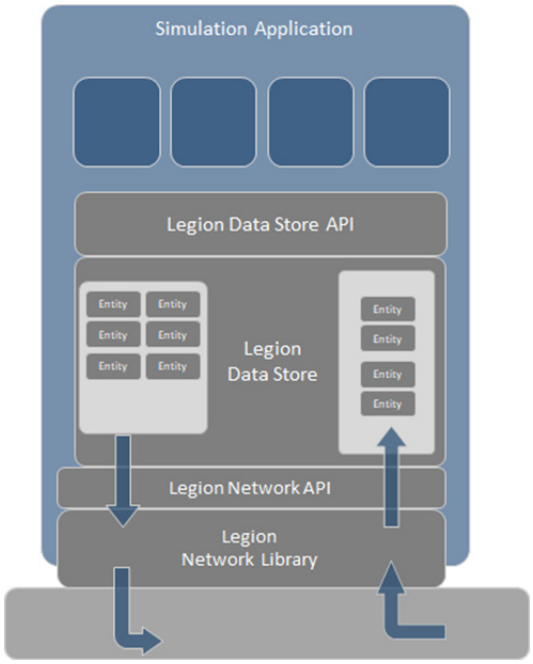

Figure 1-4. Network library

The MAK Legion Network Library is responsible for communicating information between simulation applications. The MAK Legion Network Library's job is to synchronize the application's local Data Store with a mirrored copy in the MAK Legion Server. It pulls the state of locally-simulated entities from the Data Store and sends any required updates to the Server. It writes data to the Data Store based on updates from the Server about remote entities that match the application's interest criteria.

1.1.3. MAK Legion Server

The MAK Legion Server is a central application to any MAK Legion exercise. The server is responsible for determining when client applications have overlapping interest and sharing data between those clients. The server also maintains internal performance statistics and can provide those to interested clients.

Figure 1-5 shows an example of Sim Engine instance A and Sim Engine instance B as clients working with the MAK Legion Server. The server maintains a copy of the data stores from clients A and B as well as for C and D (which represent any arbitrary number of other, not shown, simulation clients). Any Sim Engine instance could be simulating a single entity (as in a first-person simulator) or could be simulating thousands of entities (as in a computer generated forces (CGF) simulation).

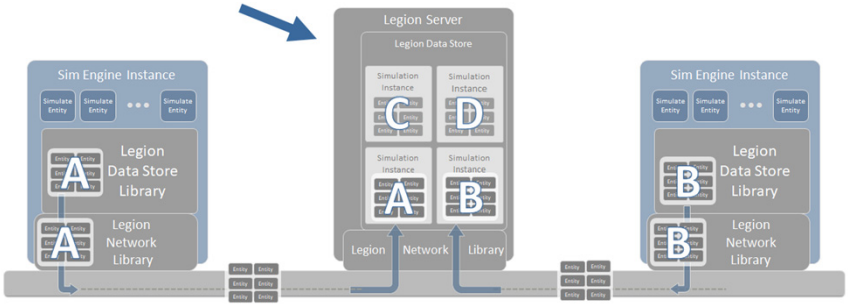

Figure 1-5. MAK Legion Server

1.2. MAK Legion Utilities

The MAK Legion distribution includes the following useful utilities:

- The MAK Legion Monitor application is a MAK Legion client that subscribes to the server statistics and provides insight into the behavior of the server. The monitor provides several different views of all the data that the server provides.For more information about the MAK Legion Monitor, please see Chapter 5, *MAK Legion Monitor*.
- The MAK Legion Logger is a client application that connects to the server and records all client messages and is able to replay the messages back as though they were being generated from the original clients. For details, please see "Running the MAK Legion Logger," on page 3-3.

2. Installing MAK Legion

This chapter explains how to install the MAK Legion and related software.

2.1. Installing MAK Legion

A full MAK Legion installation includes the application, the License Manager software and, for HLA operation, an RTI (or Runtime Infrastructure).

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

2.1.1. Installing MAK Legion on Windows

When you install large applications on Windows, there may be a delay of up to several minutes from the time you try to run the setup program to the time that an installation dialog box is displayed. This is due to how Windows scans setup programs before it executes them. If you experience this problem, turning off User Access Control can reduce or eliminate this delay.

Windows versions of MAK Legion are provided as executable installer files on DVD, or as downloaded files. The installers are named to indicate the compiler used to build that version of MAK Legion

➤ To install MAK Legion, run the installer. Follow the instructions in the installation wizard.

2.1.2. Installing MAK Legion on a Linux System

Linux versions of MAK Legion are provided as compressed tar files on DVD, or as downloaded files.

To install MAK Legion on Linux:

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

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

   where *`application`* is the product release identification.

2.2. The MAK Legion Environment Variable

The MAK Legion installer will try to set the MAK_LEGIONDIR environment variable to the installed runtime location. This environment variable is used to find the MAK Legion runtime libraries and the MAK Legion plugins that client and server applications load. The plugins are used to provide the client and server applications with object model specific optimizations since the MAK Legion framework is object model agnostic.

2.3. The MAK Legion Directory Structure

The MAK Legion installation process creates a directory structure like the one shown in Figure 2-1.

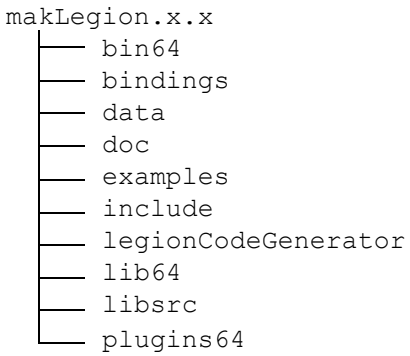

Figure 2-1. Logger directory structure

Table 2-1 describes the contents of the MAK Legion directories.

Table 2-1: Logger directory structure

Directory	Contents
bin64	Executables, including utilities and example applications.
bindings	C#, Java Code examples and bindings.
data	Data used by applications.
doc	Documentation.
examples	Example source code and project files.
include and *lib64*	Header files and libraries for the MAK Legion API.
legionCodeGenerator	Code Generator executables and object model definition.
plugins64	Plugins for client and server applications.

2.4. Installing and Setting Up the MAK License Manager

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

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

The License Manager installer is included on MAK installation media. It is separate from the product installers. You can download the installers from our web site at https://www.mak.com/support/license-support.

Complete installation and configuration instructions are included with the License Manager installer. Instructions are also available at the license support page.

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

2.4.1. Specifying the License Server

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

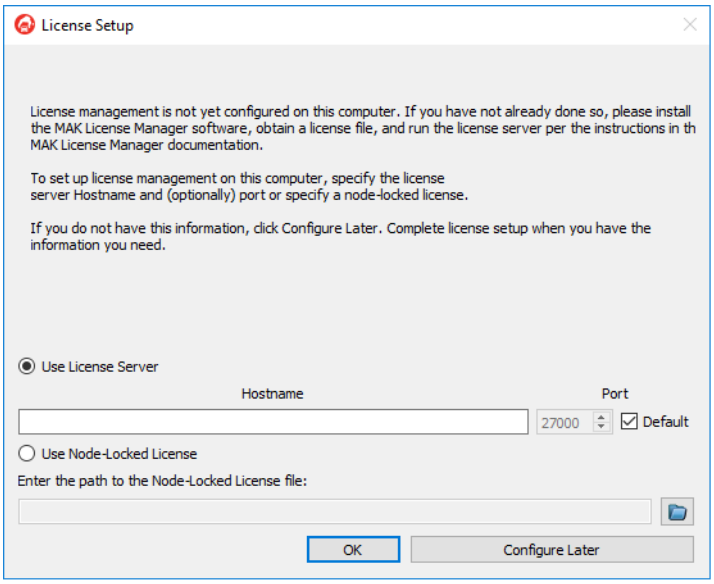

Figure 2-2. License Setup dialog box

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

If you know the hostname, type it in the Hostname box and then click OK. The application will start.

Under limited circumstances, MAK issues node-locked licenses. If you have a node-locked license, select the Use Node Locked License option and enter the path to the license file.

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

The MAKLMGRD_LICENSE_FILE Environment Variable

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

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

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

Windows

To add the MAKLMGRD_LICENSE_FILE in Windows:

1. On the Start menu, click the Settings button. The Settings window opens.
2. In the search box, type environment. A list of choices is displayed.
3. Select Edit the System Environment Variables. The System Properties dialog box opens.
4. Click Environment Variables. The Environment Variables dialog box opens.
5. In the System Variables group box, click New. 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.

Linux

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

```
setenv MAKLMGRD_LICENSE_FILE @oak
```

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

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

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

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

3. Running the MAK Legion Server and Clients

This chapter explains how to run the MAK Legion Server and MAK Legion clients.

3.1. Running MAK Legion Server

The MAK Legion Server is the central application that is required for a MAK Legion Exercise. The server is configured using a configuration file that, at a minimum, must set the network adapter to use and which port to listen on for connections. All other configuration parameters can use the default values.

The default configuration for the Legion Server is to use the loopback device.

To start the Legion Server from the Start menu, choose **MAK Technologies** → **legionServer**. You can also start the Legion Server from the command line by running the *legionServer.exe* application from the *./bin64* directory.

One setting that you may want to adjust for performance reasons is threadPoolSize. The server is a highly parallel application and is designed to take advantage of all the cores available in a CPU. Depending on the machine the server is running on, the configuration should also set the number of threads that the server uses. It is recommended to set threadPoolSize equal N-1, where N is the number of logical cores available on the server CPU. If more than one application is expected to be run on the same machine, then adjusting the number of the threadPoolSize is recommended to prevent CPU starvation for any application.

3.2. Running a MAK Legion Clutter

The MAK Legion Clutter Client is a testing application that generates simple simulated entities and moves them within a region. You use this tool to test connections and performance.

To start the Legion Clutter Client from the Start menu, choose **MAK Technologies** → **legionClutterClient**. You can also start the Legion Clutter Client from the command line by running the *legionClutterClient.exe* application from the *./bin64* directory.

3.3. Running the MAK Legion Monitor

The MAK Legion Monitor is a client application that connects to the server and displays entity, event, and diagnostic information about the server.

To start the Legion Clutter Client from the Start menu, choose **MAK Technologies** → **legionMonitor**. You can also start the Legion Monitor from the command line by running the *legionMonitor.exe* application from the *./bin64* directory.

3.4. Running the MAK Legion Logger

The MAK Legion Logger is a client application that connects to the server and records all client messages and is able to replay the messages back as though they were being generated from the original clients.

- The 'r' key starts recording.
- The 's' key stops recording.
- The 'p' key plays back a recording.

To start the Legion Logger from the command line, run the *legionLogger.exe* application from the *./bin64* directory.

The Logger receives all data from the Server. Due to the large amounts of bandwidth that the Logger can require, this application runs best when co-located on the same machine running the server.

4. Configuring MAK Legion

This chapter explains how to configure MAK Legion.

4.1. The Legion Configuration File

Every Legion application provided with the MAK Legion installer uses a configuration file to set startup parameters. The included applications may also use command line parameters to set configuration properties.

The configuration file is written in the Lua syntax. Configuration parameters are contained in the **`config`** table. Comments are preceded with **`--`**. The following code is an example configuration file:

```
connectionType = "Legion";
config = {
   name = "Legion Example";

   threadPoolSize = 4;

   -- Connection network
   serverAddress = "127.0.0.1";
   deviceAddress = "127.0.0.1";
   port = 6000;
   tcpNoDelay = true;
   asyncIo = false;

   -- Connection process
   responseInterval = 30;
   heartbeatInterval = 5.0;
   timeoutInterval = 30.0;
   connectionRetryCount = 10;

   -- Buffers size in MB
   socketReceiveBufferSize = 20;
   socketSendBufferSize = 20;
   messageReceiveBufferSize = 40;
   messageReceiveBufferPages = 6;
   messageSendBufferSize = 20;
   messageSendBufferPages = 5;
   maxQueueSize = 100;

   -- Logging to file
   logFilename = "legionExample.log";
   logDirectory = ".";
   reuseLogFile = true;
   detachLoggingFromStdOut = 2;

   -- Miscellaneous
   notifyLevel = 2;
   deadReckoning = true;

   -- Compression
   dataCompressionEnabled = false;
   compressionEnabled = true;
   minMsgSizeForCompress = 1000;
   compressorName = "LZ4";
   acceleratorFactor = 2;
   quantizeFloatDelta = false;
   compressBufferSize = 20;
};
```

4.2. Command Line Parameters

Each application provided with the MAK Legion SDK may override the parameters in the configuration file from the command line. To override a value on the command line, use two dashes (**--**), the name of the parameter, and the new value.

For example, to change the number of entities that the Legion Clutter Client will produce:

```
legionClutterClient.exe --maxEntityCount 1000
```

4.3. Printing Help

Each application provided with the MAK Legion SDK will print out a list of the command line options when you enter the **-h** option from the command line.

5. MAK Legion Monitor

This chapter describes the MAK Legion Monitor.

5.1. MAK Legion Monitor

The MAK Legion Monitor application is a MAK Legion client that is useful for debugging and diagnostics. The Legion Monitor client can subscribe to all entities, events, controls, as well as the server statistics to provides insight into the behavior of the server. The monitor provides several different views of all the data that the server provides.

The MAK Legion Monitor requires the use of a modern graphics card in order to render the large entity counts. We recommend a recent Nvidia GeForce card.

5.1.1. Server View

The Server View shows the state of the server, how long the individual phases of a server frame takes and how much bandwidth it is processing each frame. The Server View also shows a list of each connected client and the client's load on the server, as well as the clients interest area and which other clients they have interest in.

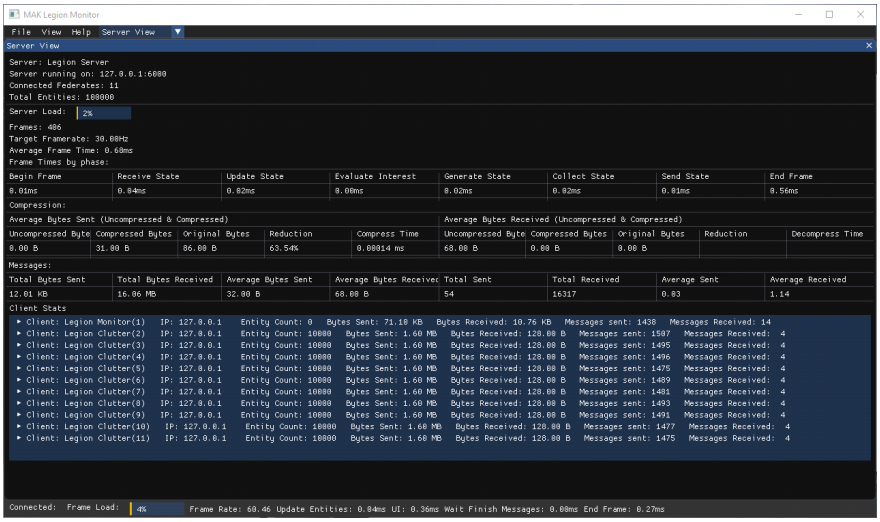

Figure 5-1. Server View

5.1.2. Database View

The Database View shows the state of the database as sent by the server. This view allows the user to see what data a particular client is publishing or which data they are receiving, and from what client. The database view is automatically generated from the registration of attributes with the server. This allows additional attribute data that is added by a client application to be decoded and displayed.

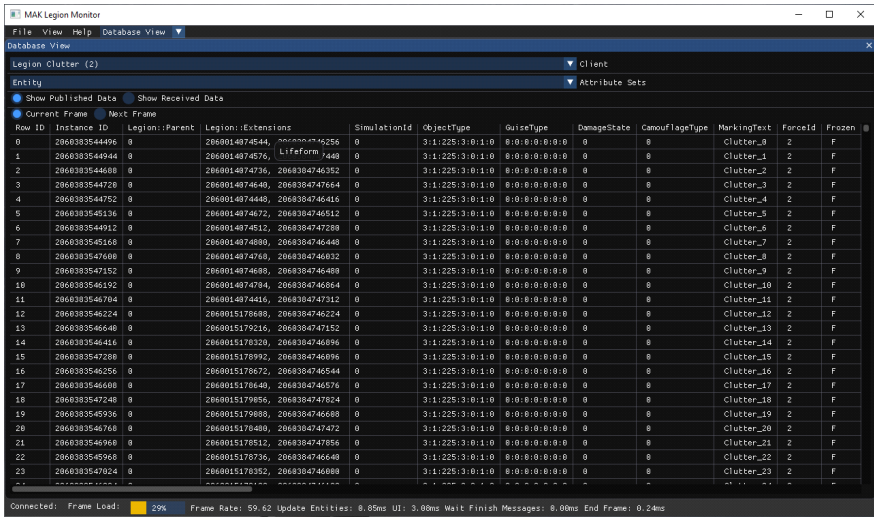

Figure 5-2. Database View

Mousing over an extension or any field that is an instanceID displays a tooltip that shows what the type is. Right-clicking the instanceID lets you to open a window to decode that specific instance.

You can switch between displaying the data the client is publishing and displaying what the client is receiving by using the check box “Show Published Data” and “Show Received Data”.

5.1.3. Event View

The Event View shows the events received and decodes the data for viewing. You can filter the event list by type or by sending client ID.

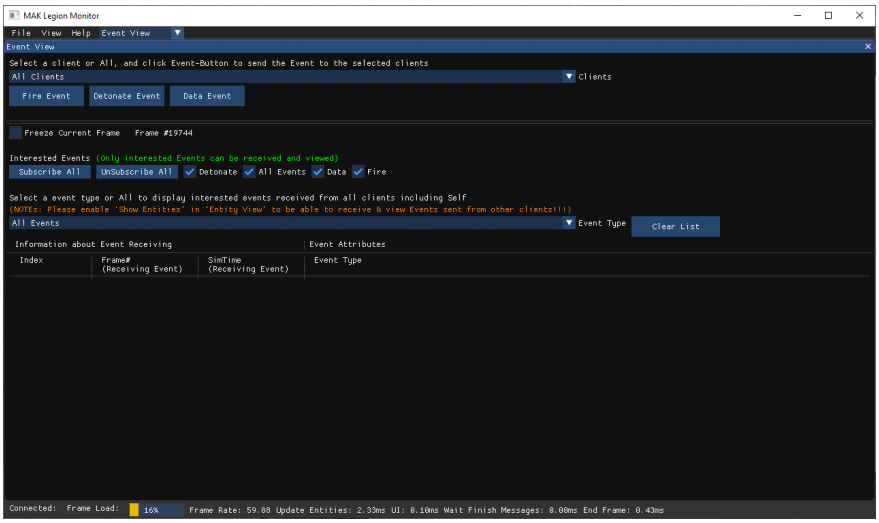

Figure 5-3. Event View

5.1.4. Control View

The Control View shows the controls received and decodes the data for viewing. You can filter the list by type or sending client ID.

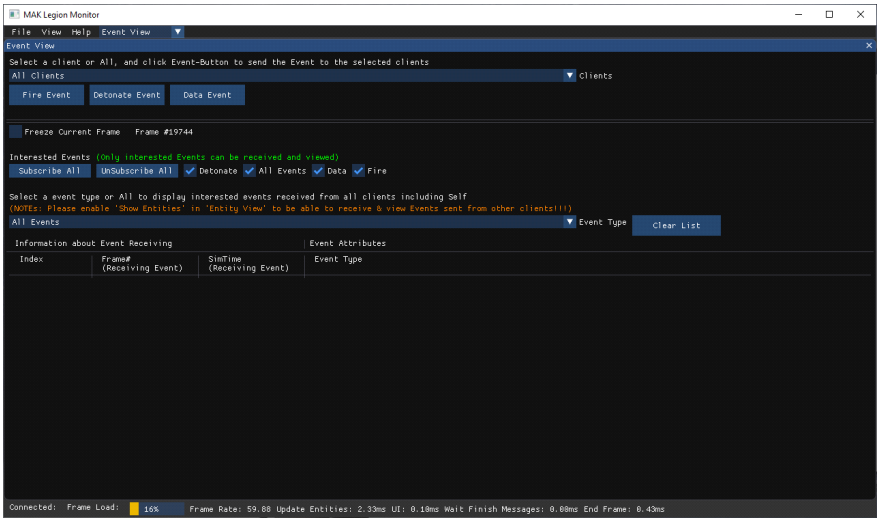

Figure 5-4. Control View

5.1.5. Entity View

The Entity View displays a 2D map of the world and lets you see the location of entities and the interest regions of clients. You can turn entities on and off and turn individual client interest regions on and off.

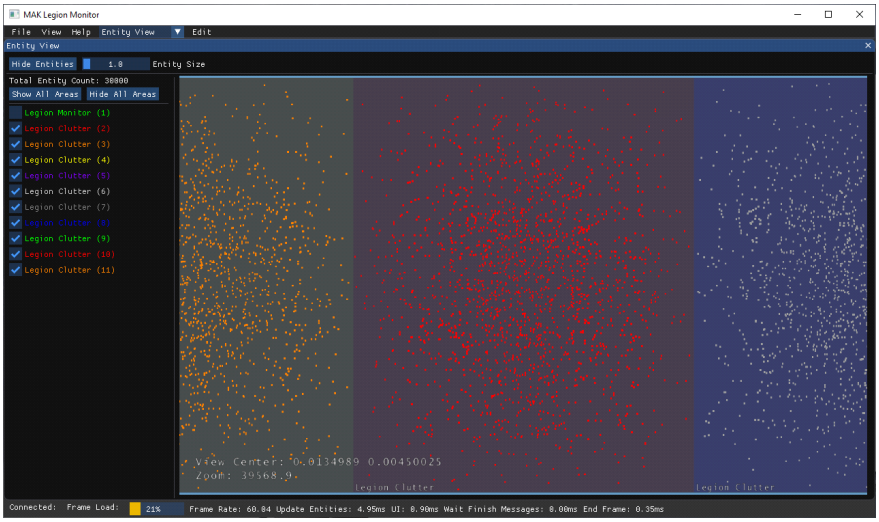

Figure 5-5. Entity View

6. The MAK Legion SDK

This chapter describes the MAK Legion object model and how to extend it.

6.1. MAK Legion SDK Concepts

This section explains important concepts used by MAK Legion.

6.1.1. Simulation Database

The central object of the data server is the Simulation Database. The Simulation Database stores all simulation data and creates/sends all events and controls. The simulation database also is responsible for creating views of the database.

6.1.2. Attributes and Attribute Sets

Persistent simulation objects are defined as a collection of named attributes called an Attribute Set. Each simulation object may be defined by one or more attribute sets. All data in the attribute set is only accessible through the MAK Legion Data Store API. Attribute sets may also be extended one of two ways:

- Add additional attributes to the set. Every instance of the simulation object has these new attributes.
- Attach a dependent attribute set to a given instance of a simulation object to provide instance specific extensions.

6.1.3. Instances

Instances of persistent simulation objects are accessed through an instance repository interface. Instance data can be accessed through a WriteStateInstance or a ReadStateInstance, or through a more specific simulation class derived from them. WriteStateInstances and ReadStateInstance (and their derived classes) are very lightweight objects similar to pointers that are references to the data within the Legion Data Store and may be copied by value rather than requiring you to store a reference or pointer to them.

Instance interfaces are thread safe and can be used to read/write data from multiple threads. WriteStateInstances are created for local entities that the application has created. You can update the fields of the object through this class. ReadStateInstances are typically returned from a View (see section 6.1.6) of the Legion Data Store and may not be used to write data. Typically, ReadStateInstances represent remote entities that the application does not own.

6.1.4. Simulation Frame

A key concept of the MAK Legion Data Store Architecture is that there is a beginFrame() and endFrame() call on the database that denotes a frame of simulation. Between beginFrame() and endFrame(), the simulation objects may be updated by the user if they are owned by the user, or the network if they are reflected. All events and controls need to be created and sent between beginFrame() and endFrame(). Processing of simulation messages from the Replication Layer also needs to be done between begin and end frame. It is an error (an exception is thrown) to attempt certain data store functions outside of a frame. Simulation object state is guaranteed to be in a consistent state between beginFrame() and endFrame() when viewed. Updates to simulation objects with a frame will not be visible until the next frame. All simulation object data is double buffered, which provides for thread safety when reading and writing simulation object data allowing for multiple threads to update the simulation data.

6.1.5. Events

Simulation events are a transient transaction that occurs during a single frame that describe something that occurred within the simulation. Common examples of events are the fire and detonate events describing weapon fire that occurred during simulation. An event of a given type is created through the Event Manager, which is discovered from the Simulation Database. A user creates an event and then fills out the fields of the event. Once the event data is set, the user then adds the event to the event manager. This will queue the event for distribution to anybody that registered a callback for the event type.

6.1.6. Controls

Simulation controls are very similar to simulation events except they define a transient transaction that controls the simulation application. Examples of simulation controls are Scenario Start/Stop/Pause, Load Scenario, and transfer of ownership. All controls have a sender and receiver field. These fields are for the client ID of the application sending the control and the client ID of the intended recipient. A wildcard may be used for the recipient and the control will be distributed to all clients.

6.1.7. Interest Management

MAK Legion clients when connected to a server need to tell the server a geographic region that the client intends to publish data into and a geographic region that it is interested in receiving data from. These are the intent region and the interest region. The intent and interest regions are typically the same geographic area, but not necessarily required to be so.

6.1.8. Ownership Transfer

The Legion Framework allows for the transfer of ownership of simulation objects between applications. A common use case is for multiple simulation applications to each be responsible for non-overlapping regions in which they control the entities. If an entity moves from one area to another area, a transfer of ownership of that entity can be initiated to transfer the entity from one simulation application to the other.

6.1.9. Views

A View of the database is an object that is updated every frame to provide a list of instances that meet a specific criteria. An example view is the AllEntity view, which is a list of all the entity instances in the database, regardless of whether they are local or remote. It is possible to register callbacks on the view for when instances are added or removed from the view. It is possible to create a view with a custom predicate function to get user generated views.

6.2. Using the SDK

A typical MAK Legion application must make several API calls, using the C++ API (C# and Java mirror this same API). The key steps are:

1. Initialize the configuration object.
2. Create the simulation database.
3. Initialize the Object Model.
4. Create the connection.
5. Set the Interest Region.
6. Create Entity Instances.
7. Run a Frame of Simulation.
8. Shut down.

6.2.1. Initializing the Configuration Object

Create a configuration object as follows:

```
theConfiguration = new Legion::Configuration();
```

You can pass in the name of the configuration file that contains parameters as a construction parameter. The configuration object may also be passed the command line arguments to find parameters to set. Please see Chapter 7, *Extending the Object Model* for information about how to add additional parameters to the configuration object.

6.2.2. Creating the Simulation Database

After you create a configuration object, you must create the simulation database:

```
theSimDb = new Legion::SimulationDatabase(*theConfiguration)
```

You should check that the returned simulation database is valid before you initialized the object model.

6.2.3. Initializing the Object Model

The MAK Legion framework is object model agnostic. The MAK Legion SDK provides the Legion Standard Object Model, which is derived from the RPR and DIS object models of HLA and DIS. To initialize the Legion Data Store with the default object model, call the object model initialization function:

```
LOM::initObjectModel(*theSimDb)
```

If there are extensions to the object model, such as the VR-Forces extension that is provided with the MAK Legion SDK, you must initialize the extensions to populate the database with their data:

```
VRF::initObjectModel(*theSimDb)
```

6.2.4. Creating the Connection

Once the simulation database is created and the object models initialized, create and initialize the MAK Legion connection:

```
theConnection = new Legion::Connection(*theSimDb, *theConfiguration);
```

Creating the connection object will automatically attempt to connect to the server. The application can test if there is a valid connection with the isConnected() function:

```
if ( !theConnection || theConnection->isConnected() == false )
  {
    LOG_WARN("Example") << "Error creating talk exercise." << std::endl;
    return -1;
  }
```

6.2.5. Setting the Interest Region

Once a connection has been made, the application should set the interest and intent regions with the connection so that the server will send the application pertinent information. This is done by creating a bounds object:

```
Legion::BoundarySpatialPlanar bounds;
bounds.setLowerBound(minBounds);
bounds.setUpperBound(maxBounds);
```

The bounds object defines the area with a minimum and maximum latitude and longitude coordinates, in degrees. The bounds object is then used with the connection to be sent to the server:

```
theConnection->setIntent(bounds);
theConnection->setInterest(bounds);
```

6.2.6. Creating Entity Instances

The application needs to use the simulation database to create an instance of an entity:

```
LOM::WriteEntityRepository entity;
Entity = theSimDb->createInstance<LOM::WriteEntityRepository>();
```

This creates an instance of the entity in the simulation database and returns the WriteEntityRepository, which is the interface to that entity in the database. All data will be set and read through this interface.

6.2.7. Receiving Entity Data

To receive entity data, the application needs to create an EntityView of the database. The Simulation Database provides access to the View Manager, which is responsible for creating and destroying views of the database:

```
LOM::EntityView* theEntityView =
  LOM::EntityView *)theSimDb->
  viewManager().createView(LOM::EntityView::creationName(),
  Legion::ViewScope::Remote);
```

The entity view provides a list of all the instances that are received. The application can iterate over the list of all received entities in the current frame by getting the instance list from the view:

```
std::vector<LOM::ReadEntityRepository>& entities = theEntityView->
  instances();
```

These are ReadEntityRepositories, which are very similar to the WriteEntityRepository except that they are not able to set attribute values on the entity and are read only.

6.2.8. Running a Frame of Simulation

An application must update the simulation periodically to advance the simulation. In a loop, the first thing that must happen is to tell the Legion Data Store that it is time to start a new frame of simulation and to set the new simulation time:

```
theSimDb->beginFrame(simTime);
```

Next, the connection should be told to start processing received messages:

```
theConnection->startProcessingMessages(0.01);
```

Processing of remote messages is an asynchronous operation and this call will return immediately, allowing the application to then update the local entities at the same time the remote entities are being updated by the connection.

The entity simulation step should then be run and the entity updated through the WriteEntityRepository object:

```
// Run the simulation
position += velocity * deltaTime;

// Update the repository values
entity.setLocation(position);
entity.setVelocity(velocity);
```

When all the simulation for local entities this frame has been completed, the application needs to tell the connection that it is ending the frame and to stop processing messages:

```
theConnection->waitUntilDoneProcessingMessages();
```

This is a blocking call. When the connection finishes the waitUntilDoneProcessingMessages() call, the simulation database must be notified that the frame is over using the endFrame call():

```
theSimDb->endFrame();
```

The endFrame() call notifies the database that no more entities will be updated and the frame of simulation is complete. The simulation database does its work and bookkeeping during the beginFrame() and endFrame() calls to identify data that has changed, generate network messages and send them. All simulation updates should happen between a beginFrame() and endFrame() call.

6.2.9. Events

Creating and sending events is accomplished using the event manager, which is accessible through the simulation database object:

```
theEventManager = &theSimDb->eventManager();
```

To create and send an event, the application needs to use the event manager to create the event:

```
LOM::FireEvent fire = theEventManager->createEvent<LOM::FireEvent>();
```

The application then needs to fill out the event fields with the data it wants to send, then call the addEvent() function on the event manager with their event data. This adds the event to the outgoing event queue and it is sent to other interested simulations:

```
fire.setSourceId(entity.globalId());
fire.setMunitionType(Legion::ObjectType(2,1,1,0,0,0,0));
theEventManager->addEvent(fire);
```

To receive an event the application must register a callback for the event with the event manager:

```
theEventManager->
  addEventCallback<LOM::FireEvent>(LOM::
  FireEventCallback(fireEventCallback));
```

The event callback will be called for all events of registered type that were received that frame.

6.2.10. Shutting Down

When the application is shutting down, delete the configuration, simulation database, and connection objects.

7. Extending the Object Model

This chapter describes the MAK Legion object model and how to extend it.

7.1. Introduction

The MAK Legion infrastructure provides a default object model suitable for virtual entity simulation. The object model is defined in a YAML formatted file and uses the MAK Legion Object Model code generator to generate all the defined instance, event, control and view classes. Different object model files can be composed together to add capability to a simulation. Not every application needs to use all of the object model components.

7.1.1. Object Model Files

Object model files define the object model profile (name, attributes, type, and so on). The are located in *./legionCodeGenerator/models* and named with the extension *.yml*. For example, here are some of the simulation concepts the Legion object model defines in *Legion.yml*:

- Bundles. Common attributes for different models: TSPI, EmbeddedSystem, EnvironmentObject, Command, and EntityCommand.
- Controls: Play, Pause, Stop, Forward, Rollback, and so on.
- Events: Collision, Fire, Detonate, RadioSignalEncodedAudio, and so on.
- Interactions: RepairComplete, RepairResponse, and so on.
- Sets: Entity, ArtPart, ArtPartParam, Lifeform, Aircraft, and so on.
- Data type.
- Generators.

7.1.2. Attributes

Attributes are defined from a set of predefined attribute data types. The types are as follows:

- Bool
- UInt8
- Int8
- UInt16
- Int16
- UInt32
- Int32
- UInt64
- Int64
- Float
- Double
- Vector2f (2 dimensional vector with single floating point precision)
- Vector2d (2 dimensional vector with double floating point precision)
- Vector3f
- Vector3d
- Vector4f
- Vector4d
- Object Type (7 Digit enumeration value following SISO Enumerations Document)
- Instance Handle
- GlobalIdentifier.

Attributes also have a cardinality. A cardinality of more than 1 means the attribute is a fixed length, contiguous array of the attribute type. An attribute may also have variable cardinality allowing the attribute to grow in size as needed.

7.2. The MAK Legion Code Generator

Makgen.exe is the tool that is used to parse the YML file and generate an API for the object that is usable with C++, C#, or Java.

The usage of the Makgen.exe application is as follows:

usage: makgen [-h] [-c | -nc] [-l LANGUAGE] [-m MODELSDIR] [-t TEMPLATEDIR] [-o OUTPUTDIR] [-v] [-d] [-e]

where:

Table 7-1:

Argument	Description
(-h | --help)	Display a help message and exit.
(-c | --clean)	Delete old files when re-generating
(-nc | --no-clean)	Do not delete old files when re-generating.
(-l | --language) *language*	Generated language.
(-m | --modelsDir) *modelsdir*	Directory of input model files.
(-t | --templateDir) *templatedir*	Directory of templates.
-o | --outputDir) *outputdir*	Output directory.
(-v | --version)	Show program's version number and exit.
(-d | --dump)	Dump merged model.
(-e | --verbose)	Verbose, elaborated mode.

7.3. Manually Registering Attributes

You can register additional attributes and attribute sets with the simulation database programmatically at runtime. A descriptor is used to describe the data type, cardinality and other properties of an attribute or attribute set. Once a descriptor is created and filled out, it is registered with the database. The database can then be queried to get the Attribute Set ID of a registered set. This Attribute Set ID is consistent throughout an exercise. Attributes also have an ID, which is consistent throughout an exercise. Attributes also have an Index, which is used within an application and is not necessarily consistent from application to application. Users creating Object Model classes or using the MAK Legion Object Model code generator will want to query the Attribute Index after registering the attributes and use the index value when accessing data.

Index

LEG-1.1-1-210505