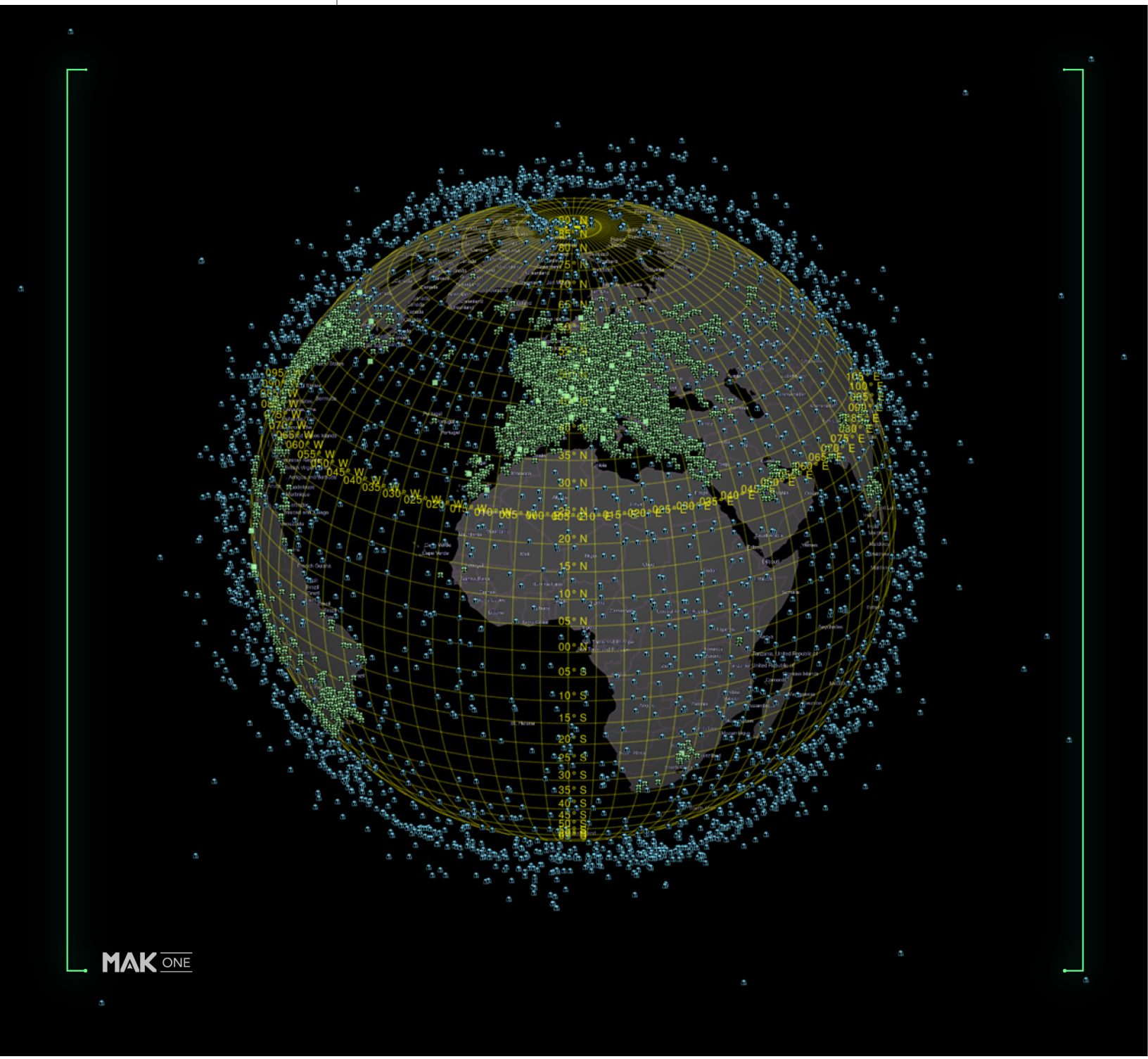
MAK TECHNOLOGIES

VR-Forces Migration Guide



Version 5.0.2

ST Engineering North America

MAK Technologies
10 Fawcett Street, Suite 204
Cambridge, MA 02138 USA

Voice: +1 617.876.8085
Fax: +1 617.876.9208

info@mak.com

www.mak.com

Revision VRF-5.0.2-9-220810

Contents

Preface

1. Migration Overview

Preface

This manual is for developers who must migrate applications from previous versions of VR-Forces to the current version. It is not intended to be fully comprehensive, but it does cover many of the significant items that must be migrated.

VR-Forces Documentation

VR-Forces documentation is provided as manuals in PDF format, online help, and HTML class documentation. The PDF files are in the *./doc* directory. The VR-Forces documentation set is as follows:

- *VR-Forces Users Guide* contains all documentation for running VR-Forces, creating and running scenarios, and configuring VR-Forces.
- *VR-Forces Migration Guide* collates the most substantial migration details for recent releases, targeted at developers.
- *Adding Content to MAK Applications Reference Manual* explains how to add visual models and terrain to VR-Forces.
- Online help. The VR-Forces front-end and the Simulation Object Editor have online help accessible from the Help menu.
- *VR-Forces Developers Guide* and API documentation. Class documentation and developers guides in linked HTML pages.
- VR-Forces Release Notes.
- *VR-Forces First Experience Guide*. A brief introduction to the most basic features of VR-Forces.
- *VR-Forces Entity Model Catalog*. A catalog of all of the simulation objects and tactical graphics configured in the Simulation Object Editor, with basic parameter details and a screen capture of the 3D model or icon.

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 VR-Vantage and the VR-Vantage Toolkit.

  - VR-Vantage 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.
  - 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 Applications 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:	jira.mak.com/servicedesk/customer/portal/1

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 10 Fawcett Street, Suite 204 Cambridge, MA 02138 USA

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:

Sans Serif Bold	Indicates a key on the keyboard or menu options. Also indicates parameters.
`Monospaced`	Indicates values you enter such as commands or arguments.
`Monospaced Italic`	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.
(|)	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.
Italic	Indicates a file name, pathname, or a class name.
➤	Indicates a 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.
	Indicates that a section is valid only for entity-level scenarios.
	Indicates that a section is valid only for aggregate-level scenarios.

Directory names are preceded with dot and slash characters that show their position with respect to the MAK Applications home directory. For example, the directory *.vrforces5.0.2/doc* appears in the text as *./doc*.

For anything in the data directory, the path is shown starting with *.SharedData/16/latest* to be clear that it is not in the MAK Applications home directory. That is the default installation path. If you installed the data in a different directory, substitute your path.

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
- GLM
- HarfBuzz
- LibCURL
- LibSquish
- LibXML
- LUA
- LuaBind
- Proj
- SMHasher
- TCLAP
- Qt Visual Graph Editor

The following libraries are covered by the BSD license:

- FreeType
- GLEW
- JPEG
- LibTiff
- Protobuf
- PCRE
- Ogg Vorbis

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.

SWIG is covered by the GPL license.

Boost License

VR-Link, and all MAK ONE products that use 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 ONE products that use 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, 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.

Lua

Some MAK products use the Lua programming language (www.lua.org). Its license is as follows:

Freefont OpenType Font Set

VR-Vantage applications and VR-Forces use the Freefont OpenType font set from the Free Software Foundation. It is covered by the General Public License (GPL). For details, see: http://www.gnu.org/licenses/gpl.html

Scintilla and SciTE

Autodesk Gameware Navigation

The following are registered trademarks or trademarks of Autodesk, Inc., in the USA and other countries: 3ds Max, AutoCAD, Autodesk, DWF, DWG, DXF, Kynapse, Maya, MotionBuilder, and Powered with Autodesk Technology.

NVIDIA

The HDR implementation uses the HDR NVIDIA gameworks SDK example that has the following copyright notice:

Multiple Files in folder: es3aep-kepler\HDR/

SDK Version: v2.11

Email: gameworks@nvidia.com

Site: https://developer.nvidia.com

Copyright (c) 2014-2015, NVIDIA CORPORATION. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of NVIDIA CORPORATION nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Third-Party Licenses for VR-Vantage Applications

VR-Vantage applications use a variety of third-party libraries. Developers who want to use these libraries may be required to purchase developer's licenses. Please see section 1.2 in *VR-Forces Front-End Developers Guide* for details.

1. Migration Overview

Upgrading a project from one version of VR-Forces to the next may require one or more of the following steps, depending on how you are using VR-Forces.

1.1. Upgrading Scenarios

You can typically load scenario files created in a previous VR-Forces directly into a newer version of VR-Forces, as long as you have migrated the data pieces that the scenario depends on (Simulation Model Set and Terrain).

If your scenarios depend only on a VR-Forces included SMS, such as EntityLevel SMS, and use a terrain database provided by MAK, then you can load your scenarios into the newer VR-Forces version with no additional work. Once saved from the newer VR-Forces version, the scenario is not expected to work in older versions anymore.

The behavior of entities is likely to be somewhat different in the scenario when using it in a new version, because the underlying behavior models will have changed in some way. MAK makes an effort to keep the behavior as consistent as possible.

1.2. Upgrading Simulation Model Sets

If you have a custom Simulation Model Set (SMS), you must upgrade this before using it in a newer version of VR-Forces. The steps and effort required for the upgrade can vary widely based on the nature of your custom SMS.

In general, the guide to different parts of the SMS are:

- Entity files (.entity) can generally be copied directly to a new version of VR-Forces.
- System files (.sysdef) and Platform files (.ope) cannot be used directly in a new version, and must be manually edited. The best practice for these files is to start with a VR-Forces shipped sysdef of platform file from the new version, and make the same edits to it that were made in the old version.
- Some SMSs include visual model definitions. To learn more about migrating these, see "1.3. Upgrading Visual Model Definitions" on the facing page.

1.2.1 Copying versus Including VR-Forces SMS

A custom SMS is typically either a modified copy of a shipped VR-Forces SMS or it includes and extends a VR-Forces SMS. Custom SMSs that include and extend a VR-Forces SMS are much easier to upgrade because these SMSs only contain system files, platform files, and entity files that relate to the customization. In a copied SMS, there is no easy way to differentiate changes made for customization from those originally in the SMS.

The VR-Forces Simulation Object Editor (SOE) includes some functionality for aiding with the upgrade of old SMSs to newer versions of VR-Forces. See the section "Upgrading Old Simulation Model Sets" in the *VR-Forces Users Guide*.

1.3. Upgrading Visual Model Definitions

Visual model definitions are configuration files that map 3D art models, 2D icons, and other visual assets to entities and configure the way they appear in the GUI. These are not the art assets themselves, but rather configuration data that points to these assets.

Visual model definitions can exist in a central location (*./appdata/Configuration/Definitions* in VR-Forces 4.10 and earlier, or *C:/MAK/SharedData/##/latest* for VR-Forces 5.0 and later), or optionally as part of an SMS.

- If you have edited the visual model definitions in the central location, then you must move these configuration files to the new version of VR-Forces manually.
- If your model definitions are part of a custom SMS, you can migrate these to the upgraded SMS using the SOE Upgrade Tool. See the section "Upgrading Old Simulation Model Sets" in the *VR-Forces Users Guide*.

1.4. Upgrading Terrain Configuration Data

Terrain configuration data are files that organize terrain source information into usable terrain data for VR-Forces. These are typically .mtf files and .earth files. If you have custom .mtf and .earth files, you might need to modify them to comply with any new or changed formatting of these files in the newer version of VR-Forces.

Check the *VR-Vantage Migration Guide* for the compatible version of VR-Vantage for any required changes. If you have custom .earth files, they may need modification. The .mtf file format changes very rarely, therefore you can probably use these files without modification.

In addition to the *VR-Vantage Migration Guide*, you can use similar .earth files that ship with the new version of VR-Forces as guides for updating your older .earth file formats.

1.5. Upgrading Plug-ins

You must recompile all plug-in code against the new version of VR-Forces before you can run them successfully. It is typical for some API changes to be made between VR-Forces versions that require some changes to custom plug-in code. The extent and nature of these changes depend on the specifics of that VR-Forces version, and how much of the VR-Forces API your plug-ins use.

For upgrading to VR-Forcesversions 4.10 and earlier, the important API changes are listed in this document. For VR-Forces 5.0 and later, the changes are part of the *VR-Forces Developers Guide*, under the API Migration Guide section.

2. Simulation Model Set Changes in VR-Forces 4.0.4

This chapter summarizes changes to system definitions and OPE files.

2.1. Simulation Model Set Changes

Changes to the SMS are organized by component or system type.

2.1.1 Flight Command Controllers

Air domain Movement Systems. The flight-command-controller was added to Fixed-wing and rotary-wing entities to handle new flight tasks. (For details, please see fixedWingFlightCommandController.h and rotaryWingFlightCommandController.h.) If you have created your own air domain movement systems, you must add this controller to take advantage of these new tasks and behavior.

2.1.2 Weapon Interface

The target-priorities parameters have moved. They are no longer part of the target selection controller. They are now in the weapon controllers inside the weapon systems, as follows:

- **target-selection-controller**:
- Target priorities have been removed from the target selection controller (moved into weapons).
- The **target-select-controller:weapon-system** connections have been removed. The **target-selection-controller** to sensor connections remain.
- The controller uses *DtComponentDescriptor* (component-descriptor) for the descriptor type).
- Weapon systems:
- Target priorities have been added into the weapon controllers. The **target-selection-criteria** parameter has been renamed to **targeting-control**.
- Target priority metadata has been removed.
- For ballistic weapons, the **system:target-list** connection is now **<controllerName>:target-to-acquire**. Please see *125mm-gun.sysdef* for an example.
- Missile Weapons. The **system:target-list** input connection has been removed.

Laser designators have the same changes as weapon systems. They also now have the parameter **integrated-with-launcher**, which determines how target input works (either from the launcher controller, if integrated, or the target selection controller directly if not).

Apache. The laser designator that went along with the laser-guided-hellfire-missile launcher has been moved into the launcher.

The laser-guided-hellfire-missile-launcher system has the following changes:

- Uses the new controller integrated-laser-guided-launcher-controller.
- The laser designator that used to be on the entity is now in this system.
- A new connection (connect missile-launcher:target-to-acquire laser-controller:target-list) was added to represent the missile launcher providing targets to the laser designator.

2.1.3 Radar Modes

Emitter systems now defines a radar-mode-list, each one of which defines a beam-list so these can be switched at runtime. For more information, please see Section 10.5, “Configuring Emitters”, in *VR-Forces Configuration Guide*.

2.1.4 Weapon Enhancements

In weapon systems the munition-wrapper resource “ammo” has been removed. Individual munitions are now regular resources.

Ballistic guns have two changes:

- Burst fire. Some ballistic guns now fire in bursts. The rounds-per-minute and extra-time-between-bursts parameters were added to the ballistic gun controller.
- Magazine-based reloading. The rounds-per-magazine parameter was added for magazine-based reloading.

2.1.5 Range Rings

- Weapon systems. The range-name parameter was added to ballistic weapon actuators and missile launcher controllers. This is the display name for range items associated with this weapon displayed in the GUI.
- Aggregate OPE files. The range-name parameter was added. The combat-range-controller was added to define range rings that can represent disaggregation ranges or subordinate weapon systems.
- Munition OPE files. The range-name parameter was added to the missile/torpedo entity definition to display missile range.

2.1.6 Ammo Select Tables

For weapons that fire a burst, hit probability is for the whole burst.

New weapons have been added, so if you have made your own damage system, you may have to make a new damage table to add new mappings.

3. Migration to VR-Forces 4.1

This chapter lists changes to the APIs for VR-Forces 4.1. For occasional updates to migration information, please check *http://www.mak.com/support/migration-support*.

3.1. API Changes

VR-Forces has the following changes to its APIs:

- Tasks, Sets and Reports now use a string as a type identifier instead of an integer.
- Simulation engine:
- The simulation engine now supports streaming and paging of feature data. The API has been updated to add this support. (For details, please see section 15.6 in VR-Forces Developers Guide.)
- *DtSimComponent* now has a virtual preFirstTickInit() method, called before the first call to tick().
- GUI:
- The *DtNetworkMessageCallbackManager* class was renamed to *DtGuIThreadNetworkCallbackManager*.
- For attributes and capabilities settings, the *attributes.mtl* and *capabilities.mtl* files are no longer used. The values are read from the SISO *.xml* file. All items should now be SISO compliant.
- *dis-entity-types.csv* is no longer used in the Entity Editor or //front-end for selecting entity types. The SISO *.xml* file is now used.
- Many GUI elements are now available to create via the *DtVrfTaskSetVariableWidgetManager*. Any parameter that is added to a scripted tasked is created using this factory. The list of GUI elements that can be created are in *vrfScriptedTasks.h*.
- Task, Set and Create menu configuration is now initially set from configurations found in the *DtVrfSimulationManager*. The menuManipulation example has been modified to show the new usage.

The VR-Vantage Control Toolkit has the following changes:

- View control messages are sent using *DtDataInteractions*. *DtDataInteractions* provide a receiverId field. The recieverId's siteId and applicationId are now used when processing view control messages. If these match the siteId and applicationId of the connection receiving the message, the message is processed. The message is also processed if siteId and applcationId are both -1's (wildcards). For backwards compatibility, siteid/applicationId of 0/0 is also processed.
- Some view control messages control observer-specific settings. These messages now let you specify the name of an observer mode to modify. If no observer mode is specified, then these commands affect all observer modes currently in use. If an observer mode is specified, then only that observer mode is affected.

3.2. Simulation Model Set Changes

Most entity OPE files now have a the new script-controller controller that enables the Lua scripting feature on the entity. See the M1A2_1_1_1_255_1_1_3_-1.ope file in .\data\simulationModelSets\default for an example.

The default-task-rules.tsk file, which controls task concurrency, has changed in format. If you created your own tasks and needed edited this file, you will need to update it.

If you modified any files in the gui folder of the simulation model set, these files have been altered a bit and will need to be updated.

If you created movement system definition (sysdef) files or modified the defaults, note the following changes:

- The path-movement controller has been removed from all systems and OPE files and should not be in any of the files.
- The convoy-task-controller has changed how it finds roads. See ground-disaggregated-movement.ssydef in \data\simulationModelSets\default\systems\movement for an example.
- Some movement sysdef files now have a compatibility-controller, This is because we have removed the plan-and-move-to tasks. Any scenarios that had plans with these tasks will still work if those entities have the compatibility-controller. See ground-wheels-off-road.sysdef in \data\simulationModelSets\default\systems\movement for an example.
- The rail-path-movement controller has been simplifed a bit and some parameters have changed. See ground-wheels-off-road.sysdef in \data\simulationModelSets\default\systems\movement for an example.
- The obstruction-sensor sensor has some different parameters for obstacle avoidance. See air-cushion-default.sysdef in \data\simulationModelSets\default\systems\movement for an example.

4. Migration to VR-Forces 4.2

This chapter lists changes to the APIs for VR-Forces 4.1. For occasional updates to migration information, please check *http://www.mak.com/support/migration-support*.

4.1. Changes to the Object Parameters Database

The object parameters database has been significantly changed. Going forward, these changes should make it much easier to migrate customized simulation model sets to new releases. It will also make it easier to add new entity simulation models.

The "higher level" organization of the OPD now consists of a set of "platforms" and entity files. The platforms are generic OPE files that represent the basic entity and object types, such as ground platforms, surface platforms, line objects, area objects, and so on. These files use the familiar MTL format. It is expected that platforms will rarely change. Any changes to a platform affect all entities of that platform type.

Individual entities no longer have OPE files. Entity specific parameters are saved in XML files with the *.entity* extension. As with the OPE files used in the past, entity files reference the systems that an entity supports. Systems continue to be specified in system definition files in MTL format. Other SMS files such as damage files, detection files, formation files, and so on have not changed from previous releases.

The Entity Editor has been updated to use these new entity files. Although the physical layout has changed, for the most part its usage remains the same as in the previous release. When you edit an entity you now just edit its entity file. When you create a new entity, you create a new entity file. The entity file also contains the object type, which used to be in *vrfSim.opd* and the menu details that used to be in *entity.mst*.

The OPD Editor no longer edits individual entity parameters. It lets you edit platforms and systems. Changes to platforms and systems affect all entities that use them.

The result of these changes is that to add a new entity, you create a new entity file in the Entity Editor. To migrate to a new release, you simply copy your custom entity files (and any files that they reference) into the SMS in the new release. The entity file will pick up any changes made to the platforms for the new release. You no longer have to update OPE files and *vrfSim.opd*. *vrfSim.opd* still exists, but simply contains variable bindings for the platforms.

You will still also have to update entity type mappings and model definitions for your custom entities.

4.1.1 Migrating Legacy Simulation Model Sets to the VR-Forces 4.2 Format

The Entity Editor has a built-in tool that can upgrade simulation model sets from VR-Forces 3.12 through 4.1.1 to the new format.

➤ **To migrate a simulation model set to VR-Forces 4.2:**

1. Open the Entity Editor.
2. Choose **File > Upgrade Old Simulation Model Set**. The Select SMS to Upgrade dialog box opens.
3. Select the SMS that you want to upgrade.

4. In the New SMS Name text box, type a name for the upgraded SMS.
5. Click Finish.

The Entity Editor displays a status window. It copies files from the current default SMS to the new SMS. Then it converts files from the old SMS to the new format.

4.2. Changes to the Lua API

The Lua function vrf::getSimObjectsNear() no longer includes the calling entity in the list. Scripts that used this function might not work correctly any more. For example, if a script checks the size of the returned list and considers a list size of 1 as being empty (because it wants to ignore itself), it will now incorrectly think the list is empty when there is one other entity in the area. If you have a script that uses this function you should examine it to determine whether or not you need to change the code to account for the new behavior.

5. Migration to VR-Forces 4.3

This chapter describes migration issues from VR-Forces 4.2 to 4.3.

5.1. Changes to Simulation Model Sets

Prevous versions of VR-Forces provided one simulation model set (SMS) – *default.sms*. VR-Forces 4.3 introduces an aggregate warfare model. Because entity modeling for aggregate warfare is quite different from that in prevous versions of VR-Forces, a new SMS, *AggregateLevel.sms*, was required. Entity-level modeling, which descrbes the entity modeling in previous versions of VR-Forces, is now provided by *EntityLevel.sms*, which essentially replaces *default.sms*.

Although aggregate-level modeling and entity-level modeling are quite different, they do share some common objects. To minimize duplication of common objects in multiple SMSs and to support greater ease of SMS customization, VR-Forces 4.3 introduces the ability for SMSs to include other SMSs. The objects that are common to *EntityLevel.sms* and *AggregateLevel.sms* are contained in *base.sms*, which is included in both of the other SMSs.

If you load a scenario in VR-Forces 4.3 that references *default.sms*, VR-Forces automatically loads *EntityLevel.sms*. When you save the scenario, it updates it to the new SMS. Therefore, most legacy scenarios should migrate easily to VR-Forces 4.3. If you are using a customized SMS, you will have to evaluate your customizations and decide how to upgrade.

For general details about including SMSs in other SMSs, please see Section 5.3, “Simulation Model Sets”, in *VR-Forces Configuration Guide*. For specific migration options, please see Section 5.3.7, “Migrating a Simulation Model Set to a New Release”, in *VR-Forces Configuration Guide*.

5.2. API Changes

VR-Forces has the following changes to its APIs. For occasional updates to migration information, please check *http://www.mak.com/support/migration-support*.

- We have renamed a lot of header files for clarity, so that they will better match class names and make things easier to find. This release includes tools to help make this part of the upgrade process easy:
- In the compatibility folder, there is a command line tool you can run to upgrade the header file references in your code. Instructions for how to do this are in *./compatibility/readme.txt*.
- Alternatively, header files with the old names have been added to *./include/compatibility*. They point to the new files. You can add this directory to your project's include paths.
- In HLA, VR-Forces can change the transport type Data interactions at run time based on the content type of the message. Before sending a message, VR-Forces checks to see if it should be best-effort or reliable. If the current setting does not match what is required for the content type of the message, it makes an RTI service call to update the transport type.
- *DtVrfMessageInterface* has new methods to allow a plug-in to change transport types as well.

- The DtIfServerStatus message now contains, by back-end, the number of entities (by kind and domain) that are being simulated on that back-end. When a scenario is loaded, the signal_allScenarioObjectsDiscovered signal is sent from the *DtVrfScenarioManager*.
- By default, the following content types are sent reliable. Everything else is best-effort.
- DtRemoveFromOrganizationType
- DtSetSubordinateOrderType
- DtAddToOrganizationType
- DtIfObjectIndexMessageType
- DtScriptedTaskResponseMessageType.

6. Migration to VR-Forces 4.4

This chapter describes migration issues from VR-Forces 4.3 to 4.4.

6.1. UUID Replaces Object Name for Identifying Objects

VR-Forces 4.4 uses a VR-Forces UUID as a unique identifier. Simulation objects and tactical graphics can now have the same names, but can still be identified as unique objects. This makes collaborative creation of scenarios and merging scenarios an easy process. As part of this transition, the Scenario Merge tool has been dropped. You can now merge scenarios simply by importing them into an open scenario in the VR-Forces front-end.

6.1.1 Loading Legacy Scenarios

When you load or import a scenario created in VR-Forces 4.3.x or earlier, all simulation objects are given new unique IDs. These unique IDs replace simulation object names in all locations where they are used, such as plans and the order of battle file.

When a legacy scenario is loaded (or imported) the DtUUID, DtRwUUID, and DtUUIDMarkingTextResolution managers convert object names to new UUIDs. A marking text to UUID mapping is created in the marking text, and all the marking text entries are then remapped to their new UUIDs. This will occur in any place that an object name is used as an identifier (plans, tasks, sets, and so on).

This process is transparent to the end user. Simulation objects are still be referenced by object name and all operations still work on object names from the users perspective.

6.1.2 API Changes

Wherever marking text used to be used to address objects, the object's UUID will be used. If you are sending messages to objects and are using the objectName() in a message, for example:

```
msg.setObjectName(obj->objectName());
```

you will now use the object's UUID:

```
msg.setUUID(obj->uuid());
```

It is still possible to use the marking text of the object, for example:

```
msg.setUUID(obj->markingText());
```

However, using old style code is not recommended. If you do not use the UUID, scenarios cannot use duplicate simulation object names and end users will lose the benefit of this change. For example, importing scenarios will not be guaranteed to work.

If you have written controllers, process state repositories, tasks, sets, and so on, that use DtRwString as an identifier for an object to apply an operation to, all you need to do to migrate your code is change **DtRwString** to **DtRwUUID**.

If you make this change, when you load a scenario, VR-Forces remaps what used to be the marking text to the new UUID representation. In the debugger you will see that the UUID also has a text pointer to the object (via marking text) that this represents.

If you have created methods that take an object name, such as:

```
void MyClass::setObjectName(const DtString&);
```

when you change your member to a DtRwUUID you should also change this prototype:

```
void MyClass::setUUID(const DtUUID&);
```

When looking up *DtVrfObjects* in the object manager you can now use:

```
lookupVrfObjectByUUID
```

If you pass in marking text, this call also tries to look up the object by marking text. However, this is not recommended, because it will not allow the controller to work with similarly named objects.

6.2. Changes to VR-Link's DtVector

In VR-Link 5.2, *DtVector* has been split into *DtVector32* and *DtVector64*. (*DtVector64* is aliased to *DtVector*.) All functions that take velocity, acceleration, and embedded position information have been converted into *DtVector32*. This was done because DIS and the RPR FOM use 32-bit values for those fields and customers using our standard *DtVector* were seeing small rounding issues due to conversion to and from 32-bits when the data was passed over the network. Unfortunately, this also affects a large number of classes in VR-Link.

This change is unlikely to require changes to VR-Forces code. However, it is pointed out here just in case. For more information, please see *VR-Link 5.2 Release Notes* and class documentation.

7. Migration to VR-Forces 4.5

This chapter describes migration issues from VR-Forces 4.4.2 to 4.5.

7.1. API Changes

If your application or plug-in adds visualizer definitions to entity element definitions in the initDeModule, they now need to be added after the signal_postLoadVisual boost signal is sent from the DtVrfScenarioManager.

Scenario rewind now uses the scenario rollback functionality. Previously you would add callbacks for scenario rewound when a scenario was rewound from the GUI. You must now add callbacks for addPreLoadScenarioCallback() and addPostLoadScenarioCallback() in the *DtCgf* class.

8. Migration to VR-Forces 4.6

This chapter describes migration issues from VR-Forces 4.5 to 4.6.

8.1. API Changes

There are no significant changes to the VR-Forces API that affect migrating from VR-Forces 4.5. Please contact support@mak.com with questions.

8.2. Migrating Simulation Model Sets

In VR-Forces 4.6, the Simulation Object Editor has added features to support the recommended method of creating simulation model sets (SMSs) and has a new migration tool for updating SMS.

MAK Technologies recommends that you not edit the SMSs that are provided with VR-Forces. We recommend that you modify SMSs by creating a new SMS that includes one of the standard SMSs and then promote the simulation objects you want to change to the new SMS and edit them there. New simulation objects would also be added to the new SMS.

To support this approach, by default the SMSs provided with VR-Forces are read-only. You can override this setting, but it reminds you that we recommend that you not edit these SMSs.

The Simulation Object Editor also includes a tool that can compare SMSs from VR-Forces 4.4 and later to the SMSs in the latest release and help you make upgrade decisions.

For complete details about editing SMSs and migrating them, please see the chapter *The Simulation Object Editor and SMSs* in *VR-Forces Users Guide*.

9. Migration to VR-Forces 4.7

This chapter describes migration issues from VR-Forces 4.6 to 4.7.

9.1. User-Level Changes

The following changes affect behavior of simulation objects and may therefore affect your existing scenarios.

- Aggregate-level scenarios involving air units may not run as expected in VR-Forces 4.7 and may need to be modified to work. VR-Forces 4.7 includes a major rework of Aggregate Level Air unit behaviors, which are not backwards compatible.
- Rotary wing movement components have been overhauled to more accurately arrive at specified destinations. The flight behavior of these entities has changed, and may result in existing scenarios carrying out differently. In most cases, rotary wing entities will complete tasks more quickly than they previously would, using more aggressive maneuvers. Adjustments to old scenarios may be needed.
- IVE terrain format remains not fully supported. There may be additional issues after upgrading to VR-Forces 4.7.

9.2. Object and System Configuration Changes

The OPD Editor has been removed. The Simulation Object Editor has been enhanced to provide additional system editing functionality to address the most common use cases for the OPD Editor with a more user-friendly approach. Some configuration changes previously available on the OPD Editor now must be done using a text editor.

9.2.1 Changes to Weapon and Damage Systems

In VR-Forces 4.7, configuration of weapon and damage systems has been significantly changed. These changes were made to support management of these systems in the Simulation Object Editor. You can now create and edit weapon systems in the Simulation Object Editor. The OPD Editor is no longer used for this purpose. Overall, these changes should make management of weapon systems much easier than in previous releases.

In VR-Forces 4.6.x and earlier, a complete configuration of a weapon system required:

- The weapon system (*.sysdef)*. Defined the system, the simulation object types that it can target, and the simulation object types that can use it.
- The ammo select file (*.asl*). Specified the type of munition to use for each type of simulation object that a weapon can fire at.
- The damage system (*.sysdef*). Specified the types of munitions that a simulation object is vulnerable to and the damage file to use to determine the probability of damage.
- The damage file (*.dmg*). Specified the probability that a simulation object will sustain damage and the type of damage it will sustain when hit by a particular munition in a particular location.

- The hit file (*.hit*). Specified the probability that a ballistic gun will hit a target at different distances. The hit file is not used for projectile weapons and indirect fire.

For release 4.7, the hit, damage, and ammo select files are no longer used. Common components of similar weapon systems have been abstracted into template files. For example, indirect fire weapon systems have a common template, as do cruise missile launchers. (This is similar to how simulation objects reference generic OPE files.) The individual weapon system files now contain variable bindings for the parameters that are common to that weapon type, targeting priority data, hit probability data, and ammunition selection data.

Similarly, components for similar types of damage systems have been abstracted into template files. Instead of having damage files for each type of power, individual damage systems contain the damage probabilities for all power types.

Power types are still configured in *detonationParams.mtl* in *base.sms*. However if you change the detonation configuration for a particular entity, the changes are now stored in a *detonationParams.mtl* file that is specific to the entity's SMS.

As an example of these changes, previously a complete configuration for the M240 7.62mm gun required editing the following files:

- *M240-7_62mm-mach-gun.sysdef*
- *M240.asl*
- *lifeform-default.sysdef*
- *M2.hit*
- *lifeform-kinetic.dmg*
- *detonationParams.mtl*.

You could have edited these files in the OPD Editor or a text editor, if you knew that they all needed to be coordinated. If you missed some aspect of the configuration, your system would probably not work or not work the way you expected.

In VR-Forces 4.7, a complete configuration for the M240 7.62mm uses the following files:

- *M240-7_62mm-mach-gun.sysdef*, which includes *mm-gun.template_sysdef*.
- *lifeform-default.sysdef*, which includes *lifeform-damage-model.template_sysdef*.
- *detonationParams.mtl*.

To edit this configuration, you would edit the M240 Machine Gun system, the Lifeform Default Armor damage system, and possibly the NATO-7.62x51mm munition in the Simulation Object Editor. You would not need to touch any of the actual files unless you needed to edit a parameter that is not accessible from the Simulation Object Editor.

If you want to create a new system, you can copy from an existing one or create a new one from a template.

There is no automated migration path for weapon systems and damage systems that VR-Forces customers have created or edited. VR-Forces will continue to support systems based on the setup in VR-Forces 4.6.1 and earlier. To the extent that customer systems have been derived from systems supplied with VR-Forces, it should be relatively easy to create and configure comparable systems using the Simulation Object Editor. For systems implemented by customers, they will have to determine if they can adapt them to existing templates, create new templates and system definition files, or continue using their systems in the old format. Systems using the 4.6.1 and earlier configuration files can display some generic information in the Simulation Object Editor, but you cannot edit target probabilities and damage tables in the SOE.

For complete details about the new system definition files, please see relevant chapters in *VR-Forces Users Guide*.

9.3. API Changes

- The VRF 4.2.x compatibility header files have been removed. It is now necessary to include the correct VRF 4.7 header file names.
- The following examples have been renamed to better reflect what they show:
  - addActuator is now modifySimComponent
  - addPsr is now addSimComponent
- A number of “reader-writer” classes have been moved out of the vrfutil library and into a new library called readerWriter. This includes the base classes for the reader-writer system, and the basic types. Any code including these header files will need to have the include path changed to the new directory name.
- Simulation commands that execute methods now take a context pointer which can be used to delay the execution of simulation commands. This was necessary to allow for the create object command to wait until the terrain was paged in before creating the entity. The simulation context can be filled in with data that can indicate the simulation command has not yet been completed. The isSimCommandComplete() method is then called with that context to see if the command has been completed. If it has then that method can return true. So, for example, if you have a simulation command and the execute methods looked like:

  ```
  bool MySimCommandExecutor::execute(const DtSimCommand* command,
  const DtPlan*)
  bool MySimCommandExecutor::execute(const DtSimCommand* command,
  const DtUUID& objectName, const DtPlan*)
  ```

  They must now look like:

  ```
  bool MySimCommandExecutor::execute(const DtSimCommand* command,
  DtSimCommandContextPtr &, const DtPlan*)
  bool MySimCommandExecutor::execute(const DtSimCommand* command,
  DtSimCommandContextPtr &, const DtUUID& objectName, const DtPlan*)
  ```

  If you do not make those changes your simulation command will not work.

- The prototype for the remapping function has changed from **(DtBackendMap* map, void* usr)** to **(DtBackendMap* map, const std::set<DtSimulationAddress>& expectredBackends, void* usr);** to allow the remapping function to also know what back-ends the scenario is expecting to be there.
- The source code for *DtVrfApp* (*vrfApp.cxx*) has been reorganized to break into more discrete functions, making it easier to customize. Users who have made a custom app based on this source should adapt their customizations to the new structure to ease future upgrades.
- It is no longer necessary to create RW variables for scripted task message parameters. For example, a selection parameter used to be defined like this:

  ```
  DtRwInt* p1RwVar = new DtRwInt("p1_selection");
  *p1RwVar = 2;
  task.variables().addVariable(p1RwVar);
  task.variableDataTypes().addVariable(new DtRwString("p1_
  selection", DtScriptedTaskIntegerVariable));
  ```

  It is now defined simply as:

  ```
  task.setValue("p1_selection", 2);
  ```

  You can find a complete list of the parameter types that are possible for the overloaded setValue function in vrfTasks/scriptedTaskTask.h.

10. Migration to VR-Forces 4.8

This chapter describes migration issues from VR-Forces 4.7 to 4.8.

10.1. Articulated Parts

VR-Forces 4.8 allows you to control the appearance of some articulated parts using the Set menu. This applies to articulated parts that are defined for the model, such as propellers, rotors, flaps, wheels, landing gear, tracks, radar, doors, hatches, and so on. It does not include articulated parts that are managed by a controller.

To take advantage of articulated parts, you must first upgrade your SMS to 4.8. Once you have gone through the process, you will have an SMS that works with VR-Forces 4.8 but you need to complete additional steps to use the new articulated part functionality.

As of VR-Forces 4.8, the default configuration for articulated parts comes from simulation components and entity visual models. All parts from these sources are combined into a single list. These default settings can be overridden or have other settings added at the entity level using the Simulation Object Editor. The overrides to part settings are stored in the *.entity* file for the entity.

➤ **To convert your SMS:**

1. Update your *.entity* files.
2. Update your *.sysdef* and *.ope* files.

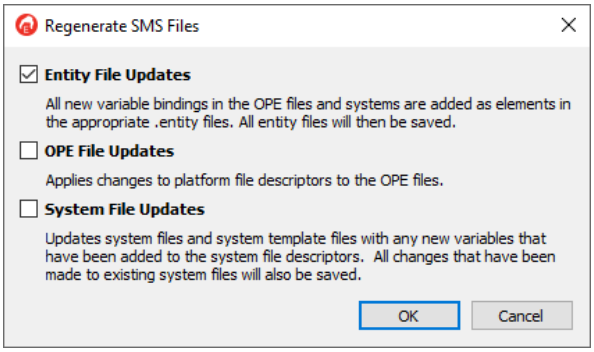
10.1.1 Updating *entity* Files

There are a few simple steps that you need to perform to update your *.entity* files.

➤ **To update the *entity* files in the SMS:**

1. In the Simulation Object Editor, open your SMS.
2. Choose **File > Regenerate SMS Files**.
3. Clear the OPE File Updates and System File Updates check boxes, and leave the Entity File Updates check box selected.

Figure 1. Regenerate SMS Files check box



4. Click OK. All entities are updated with a new articulated part section in their *.entity* files.

10.1.2 Updating *sysdef* and *ope* Files

The steps to update your *.sysdef* and *.ope* files require you to manually edit the files in a text editor.

➤ **To update the *sysdef* and *ope* files in the SMS:**

1. Using a text editor, open a *.sysdef* file and *.ope* file that include articulated part configuration. Articulated parts are configured on actuator components using the art-part-list parameter.
2. Rename art-part-list to default-art-part-list. (The old name will still work, but it is deprecated.)
3. Change actuator parameters to articulated part parameters.

Many configuration parameters are now configured on the part instead of the actuator. The actuator settings are still used to default the part settings if they are specified, but specifying them on the part creates a more unified approach to part configuration. The following example shows the turret actuator configuration from the mm-gun.template_sysdef in 4.7 and 4.8. Table 1 lists the actuator parameters you should convert and the corresponding articulated part parameters.

Example of Turret Actuator Configuration

This is an example of the original configuration:

```
(turret
        (component-descriptor-type "turret-component-descriptor")
        (component-type "turret-actuator")
        (min-tick-period -1.000000)
        (min-tick-period-variance -1.000000)
        (tick-period-uses-real-time False)
        (process-state-repository-name $process-state-name)
        (process-state-repository-type "")
        (is-enabled True)
        (debug-detail False)
        (create-component True)
        (create-on-remote-object False)
        (art-part-list
            (primary-turret
                (art-part-type $turret-art-part-type)
                (parent-part-type -1)
                (attach-point $attach-point)
                (art-part-param-list
                    (part-type 11)
                    (part-type 12)
                )
                (azimuth-neutral-offset 0.000000)
            )
        )
        (max-slew-rate $slew-rate (default $default-slew-rate))
        (fixed-az $fixed-az (default False))
```

```
        (left-slew-limit $left-slew-limit)
        (right-slew-limit $right-slew-limit)
)
This is an example of the configuration after the changes:
(turret
        (component-descriptor-type "turret-component-descriptor")
        (component-type "turret-actuator")
        (min-tick-period -1.000000)
        (min-tick-period-variance -1.000000)
        (tick-period-uses-real-time False)
        (process-state-repository-name $process-state-name)
        (process-state-repository-type "")
        (is-enabled True)
        (debug-detail False)
        (create-component True)
        (create-on-remote-object False)
        (default-art-part-list
            (primary-turret
                (art-part-type $turret-art-part-type)
                (parent-part-type -1)
                (attach-point $attach-point)
                (art-part-param-list
                    (part-type 11)
                    (part-type 12)
                )
                (azimuth-neutral-offset 0.000000)
                (max-azimuth-rate $default-slew-rate)
                (min-azimuth $left-slew-limit (default -3.140000))
                (max-azimuth $right-slew-limit (default 3.140000))
            )
        )
        (fixed-az $fixed-az (default False))
)
```

Table 1. Actuator Parameters for Conversion

Actuator Parameter	Articulated Part Parameter	Actuator Types
muzzle-offset	end-point	indirect-fire-actuator, human-gun-actuator, missile-launcher-actuator, ballistic-gun-joystick-actuator
max-elevation-rate	max-elevation-rate	human-gun-actuator, ballistic-gun-joystick-actuator, sensor-gimbal-actuator
max-elevation-range	max-elevation	human-gun-actuator, ballistic-gun-joystick-actuator, sensor-gimbal-actuator
min-elevation-range	min-elevation	human-gun-actuator, ballistic-gun-joystick-actuator, sensor-gimbal-actuator
neutral-elevation	elevation	human-gun-actuator, ballistic-gun-joystick-actuator, sensor-gimbal-actuator
max-azimuth-rate	max-azimuth-rate	human-gun-actuator, sensor-gimbal-actuator
max-azimuth-range	max-azimuth	human-gun-actuator, sensor-gimbal-actuator
min-azimuth-range	min-azimuth	human-gun-actuator, sensor-gimbal-actuator
max-slew-rate	max-azimuth-rate	azimuth-mount-controller, turret-actuator

Table 1. Actuator Parameters for Conversion (continued)

Actuator Parameter	Articulated Part Parameter	Actuator Types
nominal-azimuth	azimuth-neutral-offset, azimuth	azimuth-mount-controller, turret-actuator
azimuth-range	max-azimuth, min-azimuth (calculated using azimuth-range and nominal-azimuth)	azimuth-mount-controller
max-elevate-rate	max-elevation-rate	elevation-mount-controller
nominal-elevation	elevation	elevation-mount-controller
elevation-range	max-elevation, min-elevation (calculated using elevation-range and nominal-elevation)	elevation-mount-controller
left-slew-limit	min-azimuth	turret-actuator
right-slew-limit	max-azimuth	turret-actuator
x-axis-info	max-rotation-rate, min-rotation, max-rotation	swinging-part-actuator
y-axis-info	max-elevation-rate, min-elevation, max-elevation	swinging-part-actuator
z-axis-info	max-azimuth-rate, min-azimuth, max-azimuth	swinging-part-actuator
translation-direction	Used to determine which translation coordinates should be set.	translatable-part-actuator
max-range	max-translation (x, y, or z value, depending on translation-direction)	translatable-part-actuator
min-range	min-translation (x, y, or z value, depending on translation-direction)	translatable-part-actuator

Table 1. Actuator Parameters for Conversion (continued)

Actuator Parameter	Articulated Part Parameter	Actuator Types
range-change-rate	max-translation-rate (x, y, or z value, depending on translation-direction)	translatable-part-actuator
start-at-lowest-position	translation (default translation position)	translatable-part-actuator

10.1.3 Accessing Articulated Parts in Simulation Engine Plug-ins

Before VR-Forces 4.8, articulated parts used full kinematic states that were accessed through the attachments manager for the object. Now attached objects and articulated parts are separated and articulated parts can be accessed through the articulated part manager (*DtArtPartManager*). The articulated part manager maintains a list of articulated part state repositories (*DtArticulatedPartStateRepository*) for the entity. From a simulation component, you can look up the part of a specific articulated part type with the following code:

```
DtArticulatedPartStateRepository* part =
localStateRepository()->
articulatedParts().findPartByType(artPartType);
```

To find the articulated part types that should be controlled by a given actuator, look at the specified articulated part type list in the actuator's description.

```
DtActuatorComponentDescriptor::ArtPartTypeSet partTypes =
myActuatorDescriptor->artPartTypes();
```

If your actuator only ever controls a single part, you can also quickly look up the first part in the list.

```
DtArtPartType partType = myActuatorDescriptor->firstArtPartType();
```

10.2. Command-line Arguments in Back-end Plug-ins

You can now add command-line arguments in a back-end plug-in without needing to recompile the *vrfSimUser* application; see *./example/addBackendCommandLine*. However, if you implement:

```
DT_VRF_DLL_PLUGIN void DtInitializeVrfSettings(DtApplicationSettings&
settings)
```

in your plug-in, it is called before the plug-in is initialized. You can add new command-line arguments to the settings that are passed in.

10.3. API Changes

10.3.1 Event Manager

The event manager, *DtEventManager*, separates the exercise connection from events and interactions. Controllers, the fire/detonate manager, and the VR-Forces message interface now use the event manager to add and receive events. This gives you more control over how these events are sent over the network.

With the addition of the *DtEventManager* class (see *./include /vrfMsgTransport/eventManager.h*), direct reference to the exercise connection in your code is discouraged. While you still have access to this member through the *DtSimManager*, the *DtEventManager* (also accessed from the *DtSimManager*) is now the preferred method for getting messages onto the network.

The *DtEventManager* class was introduced as an interface to sending events in the system. There are a number of predefined events that are used (see *./include/vrfMsgTransport/event.h*) and, in issuing an event to the system, you would create an event and then use the event manager to add the event to the system. For example, if you were previously sending a *DtDetonationInteraction* on the network:

```
DtDetonationInteraction detonation;
mySimManager.exerciseConn()->sendStamped(detonation);
```

you would now send it as:

```
DtDetonationInteraction detonation;
mySimManager.eventManager()->addEvent(DtDetonationEvent(detonation));
```

The same applies to receiving a callback for a detonation interaction. If you previously had something similar to:

```
DtDetonationInteraction::addCallback(mySimManager.exerciseConn(),
detonationCallback, this);
void DtDetonationManager::detonationCallback(DtDetonationInteraction*
detInter, void* usr)
{
    DtDetonationManager* act = static_cast<DtDetonationManager*>(usr);
    act->processDetonation(*detInter);
}
```

you would want to change it to:

```
mySimManager.eventManager()-> signalForDetonationEvent().connect
(boost::bind(&DtDetonationManager::
detonationEvent, this, _1));
void DtDetonationManager::detonationEvent(const DtEvent& e)
{
    processDetonation(*static_cast<DtDetonationInteraction*>
(e.contents()));
}
```

Each event signal (see *eventManager.h*) i passed a *DtEvent*. This *DtEvent* has an event type and a contents. The contents is the memory representation of that event.

A specialized *DtNetworkEventManager* is created to send the event contents onto the network (that is, the VR-Link exercise connection) and, if the connection is not self-reflecting, post that as an event in the event manager to be processed in the next back-end tick (or in the case of the front-end, since the front-end will not tick the event manager, the event is processed immediately).

This system is put into place to make the sending and receiving of events more flexible. It allows a central point at which events can be intercepted and sent over other transport mechanisms than HLA or DIS.

See *./examples/eventManagerTCP* for an example of how to replace and use the event manager.

The *DtVrfMessageInterface* and *DtVrfTDLMessageInterface* no longer take an exercise connection in their constructor. Also, in the VR-Forces GUI, the creators now take the *DtVrfDriver* in the factory (to reference the driver's event manager), not the *DtVrlinkConnection*.

10.3.2 DtVrfObject Access Limited to Const

All objects looked up in the object manager via the various lookup calls are now returned only as const *DtVrfObject**. If your code looks like:

```
DtVrfObject* object = simManager()->
vrfObjectManager()->lookupVrfObjectByUUID(uuid);
```

it must now change to:

```
const DtVrfObject* object = simManager()->
vrfObjectManager()->lookupVrfObjectByUUID(uuid);
```

This supports thread protection in controllers that should now only affect data for those objects they control. As a side effect of this, many methods you have written (or use in the API) that used to take non-const pointers to *DtVrfObject* now will only take const pointers to the *DtVrfObject*. In addition, a number of methods have been changed to only return const pointer access or be const themselves. Therefore, if you have subclassed an existing class and part of that subclass was overriding a class accessor/mutator that returned (or took) a *DtVrfObject* as a non const-pointer, you should check the header file to see if the prototype has changed.

The callbacks from the object manager and the *DtVrfObject* that pass *DtVrfObject*'s now pass const *DtVrfObjects*.

When coding your controller, it is best to use the **entity()** method, not to reference **myEntity** directly. This will allow you to control the access to that entity (const or non-const). If you want to force the usage of a const **entity**, use the new **const_entity()** member reference:

```
processDetonation(static_cast<DtVrfDetonationInteraction&>(detInter));
```

10.3.3 Reorganization of vrfGuiCoreQt

vrfGuiCoreQt has been broken up into the following libraries:

- vrfGuiCommonQt. This library contains classes that any of the subsequent libraries can use.

- vrfGuiSettingsWidgets. Dependent on vrrfGuiCommonQt, this contains all the settings and options pages that are used in the system.
- vrfGuiDataEntryWidgets. Dependent on vrfGuiCommonQt, this contains objects that are used for data entry and can be used in dialogs and on scripted tasks. It also includes the DtVrfFilteredListView class.
- vrfGuiTaskSetQt. This library depends on vrfGuiDataEntryWidgets and vrfGuiCommonQt. It contains the task, set, conditional and command dialogs (except for the issue plan simulation command, which still lives at *vrfGuiCoreQt* level).

Some of the more significant items that were completed as part of this change (aside from the reorganization itself):

- *DtVrfObjectListViewItem* was renamed *DtObjectListViewItem* as it is now a base class in common, and subclassed at *vrfGuiCoreQt* to add the reference it has to *DtVrfGlobalPlansManager*. Any methods that were overridden to reference *DtVrfObjectListViewItem* must be renamed to now reference *DtObjectListViewItem* or they will not be called.
- *DtVrfScenarioManager* has been broken up into a *DtCommonScenarioManager* and a much smaller derived class at *vrfGuiCoreQt* level, which contains references to some dialogs that were a bit of an effort to separate out. As part of this, realizing that most developers only included this class to get at the signals, we moved the signals to a different *DtScenarioSignaler* (*vrfGuiCore*) class. Any reference to

  ```
  DtVrfScenarioManager::instance(myDe).signal_<>
  ```

  must be changed to

  ```
  DtScenarioSignaler::instance(myDe).signal_<>
  ```

  This was done such that compile time can be lessened to not include *DtVrfScenarioManager*.
- The task, set, condition, and command dialog managers have been moved to *vrfGuiTaskSetQt*. The signals for these dialog managers have been moved to a *DtDialogManagerSignaler* class in *vrfGuiCore*.
- If you had created a task or set action from *DtVrfTaskSetMenuItem*, that class is now in <*vrfGuiTaskSetQt/vrfTaskSetAction.h*>.

11. Migration to VR-Forces 4.9

This chapter describes migration issues from VR-Forces 4.8 to 4.9.

11.1. Display Unit Conversion

Display unit conversion changed from a single translation of one unit to another—that is the distance unit native to display or velocity unit native to display—to a method where the conversions are done for a group or class of objects (Fixed Wings, Rotary Wings, Application, Default).

In previous versions, you might have something similar to:

```
DtDisplayUnitsSettingsManager::instance(myDe).converter
().convertDistanceUnitNativeToDisplayString(d);
```

You can now call the convenience method:

```
DtVrfDisplayUnitsSettingsManager::instance
(myDe).convertDistanceUnitNativeToDisplayString(distance)
```

which takes into account a supplied selection to use for the basis of the conversion, based on display class.

In addition, you can use the native API:

```
double value = converter
 .convertPressureUnitDisplayToNative(v, converter.currentPressureUnit
(DtSharedDisplayUnitsSettings::theWeatherDisplayUnitCollectionName,
DtSharedDisplayUnitsSettings::thePressureDisplayUnitName));
```

The signals for units and decimals have also changed to take into account the new setup. What was previously passed as a single change:

```
myConnections.manageConnection(
 DtDisplayUnitsSettingsSignaler::instance(
 mySimulation.settingsManager()).signal_displayUnitChanged.connect(
 boost::bind(&slot_displayUnitChanged,this,_1,_2,_3)));
void slot_displayUnitChanged(DtUnitConverterCollection::DisplayUnitType
type, int units, int origValue)
```

is now passed given the category and type:

```
DtDisplayUnitsSettingsSignaler& signaler =
DtDisplayUnitsSettingsSignaler::instance(myDe.mainEventQueue());
signaler.signal_displayUnitChanged.disconnect(boost::bind(&slot_display
UnitChanged, this, _1, _2, _3, _4));
void slot_displayUnitChanged(const DtUnicode& category, const
DtUnicode& type, int unit, int oldUnit)
```

where the category can be of the types theApplicationDisplayUnitCollectionName, theDefaultDisplayUnitCollectionName, and so on, and the type is the type of item being changed: theCoordinateSystemDisplayUnitName, theDistanceDisplayUnitName, and so on.

11.2. File Locations

With the reorganization of the data package, files are now distributed differently. If you have any custom files, you must move to them to the new locations.

Entity configuration has been broken up into many files (one for each entity). The files are organized into a folder structure that matches the main data folder and, when VR-Forces starts, it reads all configuration files in the sub-folders.

As of VR-Forces 4.9:

- The model definition configuration file location changed from *./appData/definitions/ModelDefinitions* to *./data/Configuration/Definitions/Entity*.
- The element definition configuration file location changed, as well:
  - *./appData/definitions/Entity/ElementDefinitions* to *./data/Configuration/Definitions/Entity*.
  - *./appData/definitions/Unit/ElementDefinitions* to *./data/Configuration/Definitions/Unit*.
  - *./appData/definitions/TacticalGraphics/ElementDefinitions* to *./data/Configuration/Definitions/TacticalGraphics*.

If you are upgrading from a previous version of VR-Forces, you can copy your *.ommx* files into these new locations and they will be picked up automatically.

11.2.1 Custom SMS with Visual Models

If you have a custom SMS with visual models included in the SMS folder, with files in the *./data/simulationModelSets/mySms/gui/visuals* folder, you must make manual changes to the directory structure in that folder.

1. Create five new folders in the *./visuals* directory:
   - Entity
   - SpotReports
   - TacticalGraphics
   - Unit
   - UserData
2. Move the *.metx* and *.leaf* files that were previously in the *../visuals* folder to the appropriate new folder based on file name. For example, you would move *mySmsdefault_EntityTypeMap.metx* to the Entity folder because it has “Entity” in the name.
3. Move all of the *.ommx* files to the *./userData* directory.

11.2.2 Entity Configuration Files

Entity configuration has been broken up into many files (one for each entity). The files are organized into a folder structure that matches the main data folder and, when VR-Forces starts, it reads all configuration files in the sub-folders.

As of VR-Forces 4.9:

- The model definition configuration file location changed from *./appData/definitions/ModelDefinitions* to *./data/Configuration/Definitions/Entity*.
- The element definition configuration file location changed, as well:
- *./appData/definitions/Aggregates/ElementDefinitions* to *./data/Configuration/Definitions/Aggregates*
- *./appData/definitions/Entity/ElementDefinitions* to *./data/Configuration/Definitions/Entity*
- *./appData/definitions/TacticalGraphics/ElementDefinitions* to *./data/Configuration/Definitions/TacticalGraphics*

If you are upgrading from a previous version of VR-Forces, you can copy your model and element definition *.ommx* files into these new locations and they will be picked up automatically.

12. Migration to VR-Forces 4.10

There are no significant changes to the VR-Forces API or source files that affect migrating from VR-Forces 2.7. For occasional updates to migration information, please check *http://www.mak.com/support/migration-support*.

13. Migration to VR-Forces 5.0

This chapter describes migration issues from VR-Forces 4.10 to 5.0. These are user-specific changes. For API changes that impact developers, please see the VR-Forces 5.0 API Migration Guide in the *VR-Forces Developers Guide*.

13.1. API Changes

As of the VR-Forces 5.0 release, API changes are documented in the API Migration Guide in the *VR-Forces Developers Guide*.

13.2. Shared Data

Code that accessed files formerly found in *./data* (or `$(DATA_DIR)`) may need to look for them in the shared data package instead. Generally, you can use the `$(SHARED_DATA)` prefix and the `DtDe`'s `pathConfiguration().resolvePath()` to ensure that your code does not need to know exactly where the shared data is stored.

The data directory has been reorganized, as illustrated in Figure 2.

Figure 2. Shared data directory structure (partial)

```
$(SHARED_DATA) /
    Fonts
    Icons
    ModelData /
        Configuration
        Lifeforms
        Structures
        Vehicles
        etc.
    TerrainData /
        Terrain /
            California
            Massachusetts
            etc.
        TerrainConfiguaration /
            Ala Moana.mtf
            Ground_DB II.mtf
            etc.
    etc.
```

As an example, what you would previously access with:

```
myDe.dePathConfiguration().resolvePath("$(DATA_DIR)/Other/Numbers_
rgb.meif")
```

is now found at:

```
myDe.dePathConfiguration().resolvePath("$(SHARED_DATA_
DIR)/ModelData/Other/Numbers_rgb.meif").
```

Index

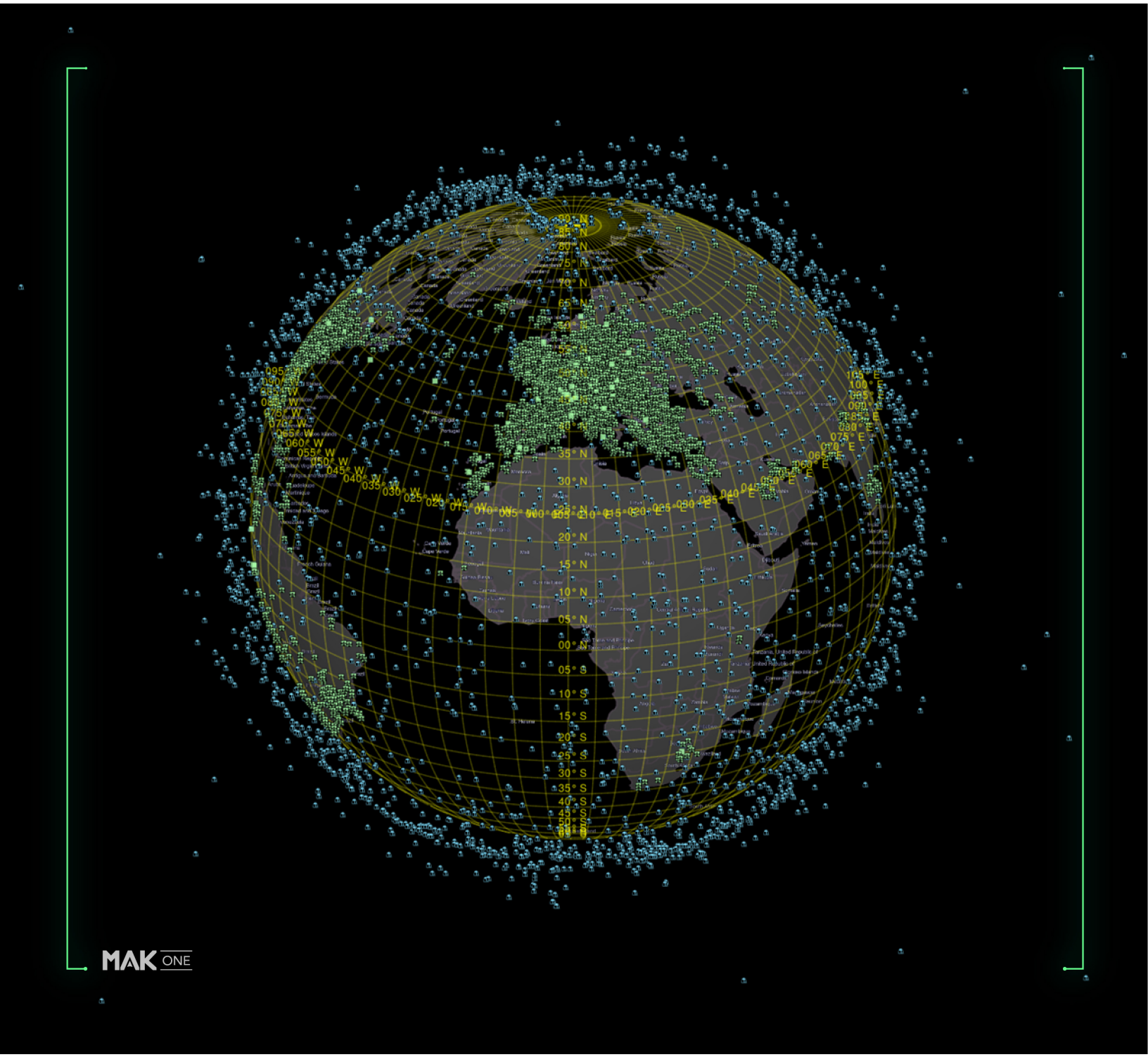
P

S

U

V

W



VRF-5.0.2-9-220810