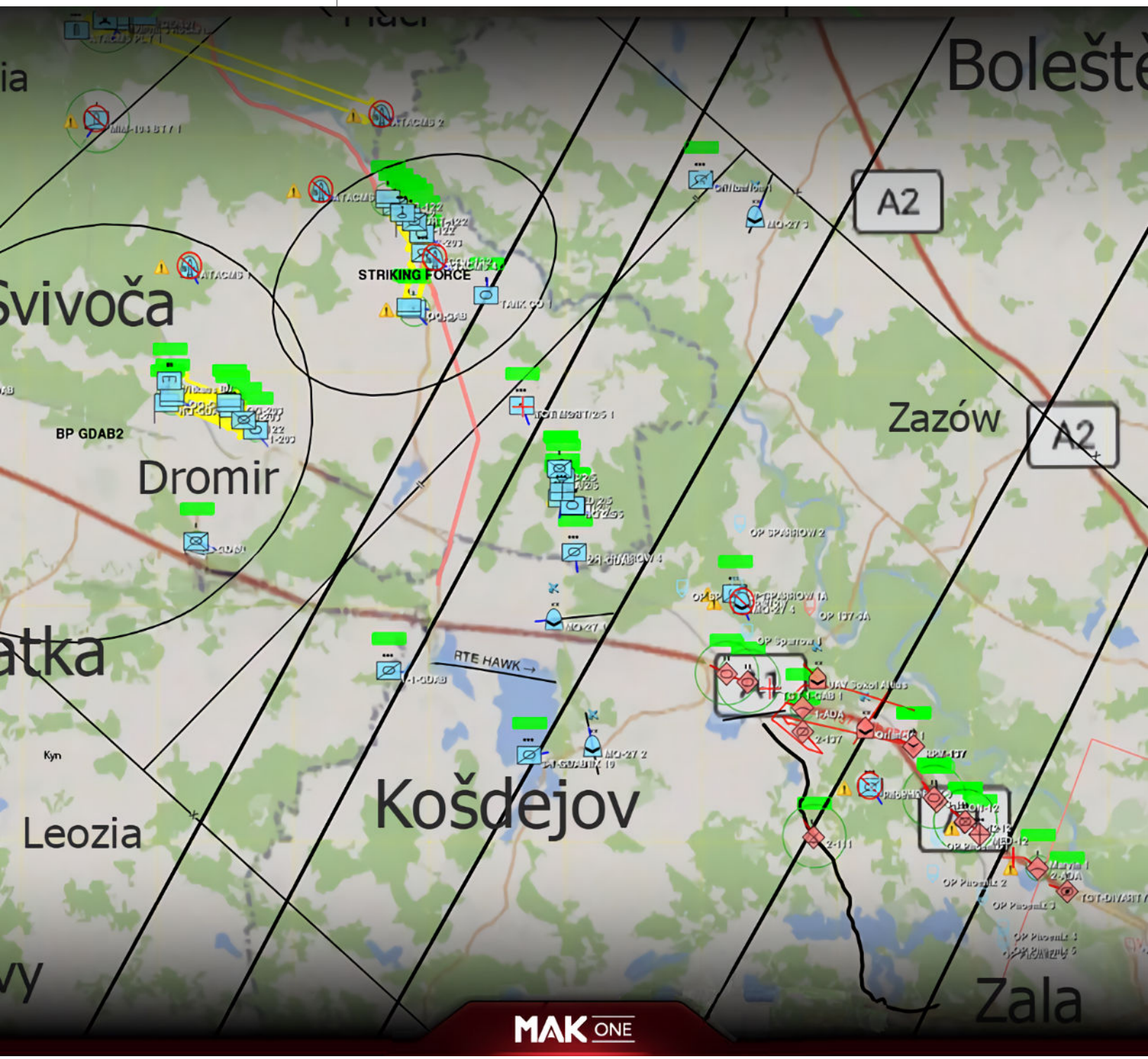
MAK TECHNOLOGIES

VR-Forces User Migration Guide



Version 5.2

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-5.2-09-251017

Contents

Preface

This manual is for developers and users 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.

Documentation Set

VR-Forces documentation is provided as manuals in PDF format, help, and HTML class documentation. Electronic versions of the documentation are in vrforcesx.x/doc. On Windows, you can access the documentation from the VR-Forces 5.2 Documentation folder on the Start menu. You can also find the most current documentation for each product's release at https://www.mak.com/mak-one/support/help/specs. The VR-Forces documentation set includes:

- *VR-Forces User 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.
- *MAK ONE Model Catalog*. A catalog of the simulation objects and tactical graphics configured in the Simulation Object Editor, with the name, object type, and a description and image 3D models where applicable.
- *Adding Content to MAK ONE Applications Reference Manual* explains how to add and edit terrain and visual models for VR-Forces.
- *MAK ONE Products Interoperability Guide* provides an overview of using MAK ONE products in DIS and HLA exercises. It explains how to configure MAK ONE products to interoperate with each other as well as other applications that adhere to the DIS or HLA standards.
- *VR-Forces First Experience Guide*. A brief introduction to the most basic features of VR-Forces.
- Help. The help, accessible from the Help menu, replicates the information in the *VR-Forces User Guide* and *Adding Content to MAK ONE Applications Reference Manual*.
- *VR-Forces Developers Guide* and API documentation. Class documentation and developers guides in linked HTML pages.
- *VR-Forces Release Notes* lists system requirements, release-specific requirements, new features and updates, bug fixes, and known problems.

Try the Help

MAK's HTML5 help is the best way to use our documentation. It's easy to navigate, with improved natural language search tools and filters to focus your search. You can click the illustrations to see high-resolution enlargements that show more detail. It's built into the product and installed locally — simply press **F1** or choose Help from the Help menu to launch the help in your browser. You do not need to be connected to the internet to use it.

If you attempt to view the help in a browser that does not fully support HTML5, you may experience issues with some features.

Document Conventions

This manual uses these typographic conventions:

Sans Serif Bold	Indicates a key on the keyboard or menu options. Mentions of letter keys do not require you to press the **Shift** key. If Shift is necessary, that will be included, for example, **Shift**+**Q**. 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, `(--help | -h)`.
/	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**.
	Click the icon to run a tutorial video in your default browser.
	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 VR-Forces home directory. For example, the directory *./vrforces5.2/doc* appears in the text as *./doc*.

For anything in the data directory, the path is shown starting with *./SharedData/##/latest* to be clear that it is not in the VR-Forces 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.

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.

Refer to https://www.mak.com/mak-one/support/help/migration-support to check for updates and other information.

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. Simulation Model Sets

Each release, there are often configuration changes to existing entities in the simulation model sets (SMSs) included with VR-Forces. As a result, entity behaviors can change when running old scenarios in a newer version. Particularly significant changes are covered in the chapter for that release.

1.2.1 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.

1.2.2 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 User 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 User 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-Forces versions 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. Migration to VR-Forces 5.2

This chapter describes migration issues from VR-Forces 5.1 to 5.2. These are user-specific changes. For API changes that impact developers, please see the VR-Forces 5.2 API Migration Guide in the *VR-Forces Developers Guide*.

2.1. VR-Forces Launcher

The VR-Forces Launcher was refactored. It now works differently than it did and, as a result of these changes, some of its command-line options have changed. If you previously started VR-Forces using the Launcher command-line options, you may need to make some updates. For details, refer to chapter on Running VR-Forces in the help or *VR-Forces Users Guide*.

2.2. Changes to Simulation Model Sets and Entities

2.2.1 Aggregate-Level Simulation Model Sets

Prior to VR-Forces 5.1, there was a single aggregate-level SMS, simply called AggregateLevel. The 5.1 release introduced AggregateLevelBase, which has much of the functionality of the old AggregateLevel SMS. The AggregateLevel SMS is now an extension of that AggregateLevelBase SMS.

The 5.2 release has moved more of the basic functionality to AggregateLevelBase. There is also a new aggregate-level SMS named AggregateTacticalLevel, which is also an extension of AggregateLevelBase. This SMS was designed specifically for tactical-level aggregate simulations.

Any existing scenarios and SMSs based on the old AggregateLevel SMS should largely continue to work. There may be cases where you need to understand the new SMS structure to correct any issues you encounter.

2.2.2 Multiple-Hit Damage Model

Most VR-Forces entity-level models use a probability-based damage model, where each hit has a probability of damaging or destroying the entity. For those entities, damage does not accumulate over multiple hits. There was a multiple-hit model, but it was not widely used. That model has been reworked. Most surface watercraft use this new model. Therefore, any old scenarios that include those entities or any entities based on those models may not take damage in the same way that they used to.

2.2.3 Resources (Entity Level Only)

The resource model in VR-Forces has changed for entity-level SMSs in two significant ways:

- Resources are now represented by their SISO enumeration rather than a string.
- Fuel resource quantities are now represented by real world units (liters or kilograms).

You must update any custom entity-level SMSs to the new resource format for scenarios using those SMS to work correctly. Use the Simulation Object Editor (SOE) to perform this upgrade. If your custom SMS has resource strings in addition to those included in MAK's standard entity-level SMS, see "Mapping Resource Strings to Resource Enumerations" on the facing page.

Fuel resources are a special case where things differ between VR-Forces 5.2 and VR-Forces 5.1.1 or earlier. As of VR-Forces 5.2, the fuel quantities are measured in real-world units (liters, pounds, or other as appropriate). This means that fuel values from scenarios created in previous releases would be incorrect. To address this issue, when you load scenarios in VR-Forces 5.2 that were created in previous releases and include entities with fuel resources, the application sets the fuel amounts to full.

Mapping Resource Strings to Resource Enumerations

To configure the SOE resource upgrade mapping, edit the *./appData/settings/resourcemap.csv* file. There are two sections in the file, which are indicated by the value in the first column, Mapping Type.

- Section 1: The first section translates strings used in set data requests, such as Set Resources, or the Resource conditional test. It uses three columns:
  - Key. This column specifies the text used to identify the resource in previous releases in set data requests or conditional tests. It is case sensitive.
  - Lookup Value. This column specifies the entity matching type to use when trying to match the string to a resource that the entity has.
  - Use Value. This column specifies what entity type should be written out when converting a resource string in a sysdef file.
- Section 2: The second section determines what data type of a resource (that is, integer or double). It matches a resource type, which is usually specified with wildcards (-1) to match a large set of types to a data type such as a double (for example, fuel) or an integer (for example, munitions).

You can upgrade an SMS by using the SOE upgrade process. For details on that process, see "Upgrading Old Simulation Model Sets", found in the *Simulation Object Editor and SMSs* chapter of the help or *VR-Forces Users Guide*. You can also create a backup of your SMS and then use the command-line option `--upgrade` *`file-path-to-SMS`*. This option upgrades your SMS in place, therefore creating the backup is strongly recommended.

When you upgrade *.entity* files, the tool writes out the resources only when it determines that the number of resources that the entity requires is different than the number of resources a system provides. If resources are not written into an *.entity* file, it is because the resources do not vary.

2.3. Consolidation of Tasks That Specify a Destination Using Either a Location or Waypoint

VR-Forces now supports task parameters that can represent either a location or a simulation object, such as a waypoint: Location Reference, with or without altitude. Many existing VR-Forces tasks have been consolidated into single tasks by using this new parameter. For example, VR-Forces previously had the separate Move to Location and Move to Waypoint tasks that have been combined into a single Move To task.

When you load scenarios created in previous versions using VR-Forces 5.2, the application automatically changes the tasks to the new ones. Simply save the scenario to preserve that conversion.

2.3.1 SMS Changes Related to Task Consolidation

Some controller components that were no longer needed have been removed from the *.sysdef* and *OPE* files that referenced them. If you have a custom SMS with your own *.sysdef* or *OPE* files referencing those old controllers, you must update the files.

For a list of the tasks that were consolidated and the controllers that were removed, please see "Merging of Location and Point Tasks" in the VR-Forces 5.2 API Migration Guide page of the *VR-Forces Developers Guide*.

The new Move To task does not support the old functionality of creating a new task for each location clicked when tasking entities to move to a location (Each Click Generates A Task). This was previously available only in plans.

2.4. Ground Vehicle Movement

There are extensive changes to the ground vehicle movement system. This led to changes to the movement tasks, and in some cases may produce different results than seen in previous releases.

Scenarios that used the old movement tasks should still function because the old tasks are mapped to the new ones, however, the exact behavior of the entities may be different. Ground vehicles might take different paths or perform different collision avoidance. In particular, for ground vehicles:

- Movement to a point is now performed by a Move To task, implemented as Lua script. The old tasks move-to-waypoint, move-to-location, move-to-waypoint (plan along roads), and move-to-location (plan along roads) are no longer available on the Task menu or in the Command Panel. For more information on changes to these and other tasks, see "2.3. Consolidation of Tasks That Specify a Destination Using Either a Location or Waypoint" on the previous page.
- The new Move To task automatically uses road movement (respecting lanes and traffic) where appropriate. You can use the Navigation Preferences set data request to specify whether an entity will prefer to use or ignore roads. By default, vehicles that use the tracked or wheels-off-road movement systems ignore roads, while vehicles that use the wheels-road movement system prefer roads.
- If there is no navigation mesh available from the start of the entity's movement path to the end, the Move To script accounts for feature obstacles and plans the entity's path to avoid them. The script also has some ability to dynamically replan paths around blockages if the vehicle encounters a situation where it cannot move.
- Entities in scenarios created in VR-Forces 5.1.1 and earlier performing the move-to-waypoint or move-to-location tasks use this road planning and obstacle feature planning. Those performing the move-to-waypoint (plan along roads) or move-to-location (plan along roads) still plan paths on road networks (which are features) and follow them.
- During off-road movement, vehicles use a new dynamic obstacle avoidance mechanism as they move along their paths.

- The move-along (route) task has been improved to better control the entity's speed and also uses dynamic obstacle avoidance.

Due to the new ground movement behavior, which now includes a planning phase, the Continue On option is no longer available to ground vehicles. It is an option for most other entities when you assign the Move To task in a plan.

Ground vehicles no longer use Gameware for steering or obstacle avoidance. For more information, refer to the Migration Guide section of the *VR-Forces Developers Guide*.

2.4.1 Task Updates

While old scenarios will update to use the new scripts, you should update existing Lua scripts and plans to use the new tasks.

Table 1. Old and new task IDs and names

5.1.1 Tasks		5.2 Tasks		
Task ID	**GUI Name**	**Task ID**	**GUI Name**	**Notes**
move-to-location	Move to Location	ground-vehicle-move-to	Move To	
move-to-location-path-plan	Move to Location (Plan Along Roads)	ground-vehicle-move-to	Move To	Set Navigation Preferences to "Prefer Roads".
move-to	Move to Waypoint	ground-vehicle-move-to	Move To	
move-to-waypoint-path-plan	Move to Waypoint (Plan Along Roads)	ground-vehicle-move-to	Move To	Set Navigation Preferences to "Prefer Roads".
move-along	Move Along Route	move-along	Move Along Route	
move-to-location (units)	Move to Location	move-to or maneuver-to	Move To or Maneuver To	Use Move To when entities should move without coordination. Use Maneuver To when entities should stay closer to one another.

Table 1. Old and new task IDs and names (continued)

5.1.1 Tasks		5.2 Tasks		
Task ID	**GUI Name**	**Task ID**	**GUI Name**	**Notes**
move-to (units)	Move to Waypoint	move-to or maneuver-to	Move To or Maneuver To	Use Move To when entities should move without coordination. Use Maneuver To when entities should stay closer to one another.
move-along (units)	Move Along Route	move-along	Move Along Route	Works for both lower-level and higher-level units, though the lower-level units will simply call maneuver-along.
convoy-to-task (units)	Convoy To	convoy-to-task	Convoy To	
convoy-along-task (units)	Convoy Along	convoy-along-task	Convoy Along	

2.4.2 SMS Changes Related to Ground Vehicle Movement

The ground movement systems have changed significantly. The collision avoidance and obstruction sensors are no longer used. If you modified any of the ground movement systems (tracked, off-road, amphibious), you must inspect the new ground movement systems to see what is configured differently in your systems, if anything. If there are differences, you must copy the system definition from the new system into your existing system and then make the necessary modifications.

2.5. Air Defense Simulation Object Groups

The air defense Simulation Object Groups, specifically the Patriot Battery and SA-21 Battalion, have been updated to use a circular launch basket. Check any of your existing scenarios that use either of these object groups to determine whether you need to make updates.

2.6. AI Enabled Renamed to Autonomous Actions Enabled

The AI Enabled set data request has been renamed to Autonomous Actions Enabled to more accurately reflect how it affects entity behavior. It performs the same functions, as described in the help and *VR-Forces User Guide*.

The AIEnabled state property has been renamed AutonomousActionsEnabled. If you have any Lua scripts that reference this property, update them to reference the new property name.

3. Migration to VR-Forces 5.1

This chapter describes migration issues from VR-Forces 5.0 to 5.1. These are user-specific changes. For API changes that impact developers, please see the VR-Forces 5.1 API Migration Guide in the *VR-Forces Developers Guide*.

3.1. FOM Support

Previous versions of VR-Forces supported the RPR FOM with custom MAK extensions. VR-Forces now uses the NATO Education and Training Network (NETN) FOM with custom MAK extensions. This FOM is based on RPR FOM 2.0, which means that support continues for all of the RPR classes that were previously supported. However, the NETN FOM is now required by VR-Forces for some of the additional attributes it adds on top of RPR.

For more information on the MAK Object Model, please refer to the MAK One Interoperability Guide.

3.2. SMS Changes

The target-selection-controller was removed and replaced with the weapon-management-controller in the platform (*.ope*) files. The target selection logic has moved to Lua scripts.

The indirect-fire-controller was also removed and should no longer be referenced in platform files. The behavior of this controller has been moved to Lua scripts.

The tick-period-uses-real-time was replaced with periodic-tick-while-paused in all simulation components. It is no longer possible to tick components based on real time. There is a backward compatibility alias that allows periodic-tick-while-paused to be set from the old name.

There are many new state-data items in the platform (*.ope*) files that are required for some features of VR-Forces 5.1 to operate.

Many files in the AggregateLevel SMS have been renamed to distinguish them from identically named files in the EntityLevel SMS. This includes *.ope*, *sysdef*, and *ui* files. Files in any custom SMS that includes the AggregateLevel SMS may need to be updated to reference the new files.

3.3. Civilian Air Traffic

In previous releases, users would create background air traffic using a Air Traffic Area tactical graphic or a Global Plan. Those features have been deprecated in favor of the Aircraft Clutter Area, which easily simulates aircraft for a specified area. If you have scenarios that used those older methods, you must edit them to use Aircraft Clutter Areas instead.

For more information, see the chapter *Generating Traffic (Pattern of Life)* in the help or *VR-Forces User Guide*.

4. 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*.

4.1. 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*.

4.2. 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 1.

Figure 1. 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").
```

4.3. Scenarios with Artillery Tasks

If you have any scenarios from a release prior to VR-Forces 5.0 that include entities that are using the M107-155mm resource as a munition for Fire for Effect tasks, you must modify the scenario to use newly assigned resources, such as M107-155mm-HE. The scenarios will still continue to work overall, but the Fire for Effect tasks assigned to those entities will not until you update them.

➤ **To update the scenario:**

1. Load the scenario in VR-Forces.
2. Reinitialize the scenario by choosing **File > Reinitialize Scenario**.
3. For each entity that has a Fire for Effect task (either as an immediate task or part of a plan) that originally used the M107-155mm resource, edit the task to use one of the new resources.

5. 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 4.9. For occasional updates to migration information, please check https://www.mak.com/mak-one/support/help/migration-support.

6. Migration to VR-Forces 4.9

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

6.1. 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.

6.2. 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.

6.2.1 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.

6.2.2 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.

7. Migration to VR-Forces 4.8

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

7.1. 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.

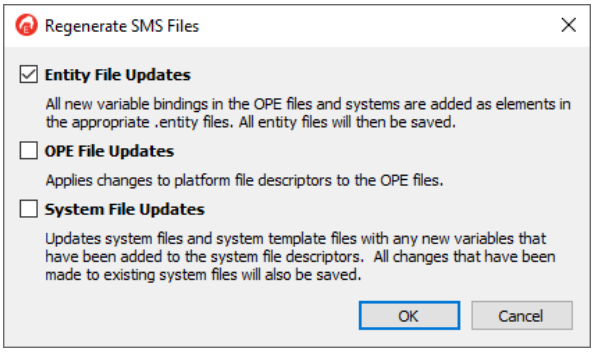
7.1.1 Updating *entity* Files

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 2. Regenerate SMS Files check box



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

7.1.2 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 2 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 2. 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

Table 2. Actuator Parameters for Conversion (continued)

Actuator Parameter	Articulated Part Parameter	Actuator Types
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
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

Table 2. Actuator Parameters for Conversion (continued)

Actuator Parameter	Articulated Part Parameter	Actuator Types
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
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

7.1.3 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();
```

7.2. Command-line Arguments in Simulation Engine Plug-ins

You can now add command-line arguments in a sim engine 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.

7.3. API Changes

7.3.1 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 sim engine tick (or, in the case of the GUI, since the GUI 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*.

7.3.2 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));
```

7.3.3 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>*.

8. Migration to VR-Forces 4.7

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

8.1. 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.

8.2. 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.

8.2.1 Changes to Weapon and Damage Systems

In VR-Forces 4.7, configuration of weapon and damage systems changed significantly. 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*.

8.3. 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 simulation engines 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.

9. Migration to VR-Forces 4.6

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

9.1. 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.

9.2. 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*.

10. Migration to VR-Forces 4.5

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

10.1. 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.

11. Migration to VR-Forces 4.4

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

11.1. 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 GUI.

11.1.1 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.

11.1.2 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.

11.2. 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.

12. Migration to VR-Forces 4.3

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

12.1. API Changes

We renamed many header files for clarity. They now 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 simulation engine, the number of entities (by kind and domain) that are being simulated on that sim engine. 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.

12.2. 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*.

13. 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*.

13.1. Changes to the VR-Forces API

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.

By default, the following content types are sent reliable. Everything else is best-effort.

- DtRemoveFromOrganizationType
- DtSetSubordinateOrderType
- DtAddToOrganizationType
- DtIfObjectIndexMessageType
- DtScriptedTaskResponseMessageType

13.2. 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.

13.3. Changes to the Object Parameters Database

The object parameters database changed significantly. 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.

13.3.1 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.

14. 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*.

14.1. 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.

14.2. 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.

15. Simulation Model Set Changes in VR-Forces 4.0.4

This chapter summarizes changes to system definitions and OPE files.

15.1. 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.

15.2. 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.

15.3. 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*.

15.4. 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.

15.5. 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.

15.6. 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.

16. Migration from VR-Forces 3.12 to 4.0.3

Before migrating from VR-Forces 3.12 to 4.x, you should understand the major areas that have changed:

- The front-end was completely redesigned. In VR-Forces 3.12, the 2D and 3D interfaces were separate executables. VR-Forces 4.0 and later have a completely integrated 2D/3D graphical user interface (GUI) based on the VR-Vantage toolkit. As a consequence, the GUI API changed dramatically.
- The Simulation API and Remote Control API have incremental improvements to support new features, but are largely similar to previous releases.
- The Terrain API has been updated to support the terrain agility features used in VR-Vantage and to support synchronization of the front-end and back-end views of the terrain.

This chapter describes the major differences between VR-Forces 3.12 and VR-Forces 4.0 through 4.0.3.

16.1. Getting Information About Objects

In VR-Forces 3.12, DtModelKey was used to find object information in various dictionaries, most notable the DtModelDataDictionary. In VR-Forces 4.x, this has been replaced with DtElementId. The DtElementId is the main key used to look up data in the system. For details, see "16.2. Storing Data" below.

The API extends the element ID with the scene object ID. The scene object ID (DtUniqueId) represents an individual screen representation of an object, given a particular model set. For example, the 2D representation of an object has a different scene object ID than the 3D representation. If you have a scene object ID, you can use the DtElementData object to retrieve the element ID based on scene object ID. If you need a scene object ID from an element ID, the DtElementData can also do that mapping.

Selection sets (discussed in "16.3. Managing Object Selection" on the next page) contain the DtElementId of the objects selected. You can use these IDs to retrieve information about selected objects.

16.2. Storing Data

In VR-Forces 3.12, DtModelData was used as the foundation for data storage, symbol creation, and symbol updating. The DtModelData class does not exist in VR-Forces 4.x. It has been replaced with the DtStateView. A DtStateView is a template class that takes a DtSimState object as a template. This object is implemented as a subclass, and those subclasses contains the data that represents the information about the object. In VR-Forces 4.x, the main DtSimState objects are:

- DtVrlinkSimulatedBaseState - Information about any VR-Link-based object, such as marking text, type, force
- DtVrlinkSimulatedEntityState - Information about VR-Link entities
- DtVrlinkSimulatedAggregateState - Information about VR-Link aggregates
- DtVrlinkSimulatedEnvironmentProcess - Information about VR-Link environmentals
- DtVrfObjectDataState - Information about an object that is VR-Forces-specific.

When an object is discovered on the network, information about that object from its VR-Link reflected object is placed into the appropriate object state. This state is added to a queue for updating.

In VR-Forces 3.12, as soon as an object update was received over the network, it was immediately updated. In VR-Forces 4.x this is not the case. Each object is updated at a certain frame rate. This allows for better performance and control over when object information is updated. For example, if an object is selected, its object update rate is increased, assuming the user might want to know more information about this object. This means all other objects continue to update at a default rate, or not at all. This does not affect the position, speed, or orientation of the object, simply the additional data that is transmitted to the front-end for that object.

This information is stored in DtStateViewCollection classes. Each DtStateViewCollection keeps a list of all particular objects of a state view type and can be used to retrieve information about that object type. So, for example, there are collections of entities, aggregates, and environmental processes.

In VR-Forces the DtVrfStateViewCollectionManager manages all these collections. It is your way of retrieving information about a particular object and its DtStateView information. The DtVrfStateViewCollectionManager is analogous to the VR-Forces 3.12 DtModelDataDictionary class.

In the DtVrfStateViewCollectionManager, all objects that are in the system are stored in the mySimulatedBaseStateViewCollection member. This member holds all structures that are of a DtVrlinkSimulatedBaseState (which all state view objects subclass from). Using the findStateView() methods of the DtVrfStateViewCollectionManager, you can find the basic state view information about any object. This information can be retrieved by marking text name or by DtElementId of the object. If you know that an item is specifically an entity, aggregate, or environmental, you can use the state view collection that is appropriate to find the state data for that object.

16.3. Managing Object Selection

In VR-Forces 3.12 you would use the map area of the currently active window to get a list of objects that were currently selected. In VR-Forces 4.x you use the DtSelectionManager class to get the list of object selected:

```
DtSelectionManager::IdSet currentSelections =
    DtSelectionManager::instance(myDe).currentSelection();
```

You can then iterate over the list to get information about these objects from the DtVrfStateViewCollectionManager:

```
DtSelectionManager::IdSet::const_iterator iter =
currentSelections.begin();
  while (iter != currentSelections.end())
  {
    DtStateView<DtVrlinkSimulatedEntityState>* stateView =
      const_cast<DtStateView<DtVrlinkSimulatedEntityState>*>(
      DtVrfStateViewCollectionManager::instance(myDe).entityStateView
()->findStateView( *iter ) );
    ++iter;
  }
```

VR-Forces 4.x also supplies a convenience class called the DtVrfSelectionHandler. This class has methods for retrieving information about current selections. For example, if you want to get data for all the currently selected objects, you can use the code in the previous example or you could do the following:

```
std::vector<makVrv::DtSimEntry*> entries =
    DtVrfSelectionHandler::instance(myDe).getSelectedEntries();
```

Remember that each DtSimEntry contains the list of all available data for that object. Most objects in VR-Forces have two DtSimEntry views of data: the VR-Link State Repository data (DtVrlinkSimulatedEntityState) and the VR-Forces-specific object data

(DtVrfObjectDataState). You can iterate over the simStates() of the state entry, dynamically casting to the object in question that you want. Once the dynamic cast succeeds, you can use that data.

16.4. Creating Symbols

In VR-Forces 3.12, if you wanted to affect the way that symbols were created, you would override the DtMtlSymbolMapper, and install your own version.

In VR-Forces 4.x, symbols are created from DtVisualDefinitionSets. These visualizer sets are organized in DtObjectDictionary classes. Each class of object (entity, aggregate, environmental, interaction) has its own object dictionary. To affect the look of a symbol, modify the object dictionary by adding your own DtVisualDefinition or DtVisualDefinitionSet. The exampleObjectDictionary, exampleRangeLine, and exampleVisualDefinitionSet examples show how this is done.

Mapping entity type enumerations to models is now done in the Entity Type Mappings dialog box rather than in symbol map files. Entity types are mapped to entity definitions that contain the correct visualizer information for that entity type.

In VR-Forces 4.x, the actual visual definitions are not created in the main GUI thread. A DtElement object (or a subclass) is created. It contains all the visualizers needed to visualize an object. DtElement objects are created in the network thread and are protected from the main thread. If you want to—in the code—change how these items look, you must do that before creation by changing the visual definition set for that particular object. Individual visualizers, however, can be created to self-configure, given certain conditions. There has been an attempt to move away from direct manipulation of the symbols and towards a more object-oriented approach for symbol manipulation.

If you require access to an individual symbol, you can do so through the DtAgentManager. Since each object is scene dependent, you must know the model set you are referencing to get the scene object ID of the object you care about:

```
  DtElementData::SceneObjectIdList ids;
  myDe.dataBank().elementData().getSceneObjectIds(elementId, ids,
myDe.driverManager().
    inputDriver().currentChannel()->currentModelSet());
  DtUniqueID idToUse = elementId;
  if (ids.size())
  {
    idToUse = ids[0];
  }
  DtAgentUpdateResolverInterface* updater = myDe.agentManager
().findUpdater( idToUse );
  if(updater)
  {
    DtSceneObject* sceneObject = updater-
>castObjectTypeFromUpdater<DtSceneObject>( "DtSceneObject");
    if ( sceneObject )
    {
      sceneObject->getPosition(0, dbLocation);
    }
  }
```

Please refer to the DtSceneObject class for the information you can retrieve. Note that the scene objects are screen representations only and do not contain simulation data.

16.5. Sending Messages to the Sim Engine

In VR-Forces 3.12, to communicate with the sim engine (sending sets, tasks or interface content messages), the remote controller was used: `DtVrfGuiAppEventController::remoteController()`.

In VR-Forces 4.x there is no direct access to the remote controller from the GUI thread. Since the GUI is not network dependent, the DtVrfSimMessager class was created to interface the GUI to the back-end. This class should be used to send all messages. The interface content example shows how to send interface content messages using DtVrfSimMessager.

16.6. Adding Callbacks to Receive Messages

In VR-Forces 3.12 the DtVrfMessageInterface class was used to register callback messages for message types. In VR-Forces 4.x, the DtVrfSimMessageHandler is used. The DtVrfSimMessager also interfaces with this class, and that can be used as well. The interface content example shows how to register for a callback message using DtVrfSimMessageHandler.

16.7. Keyboard and Mouse Handling

In VR-Forces 3.12, you would subclass and install an event handler to handle mouse events and keyboard events. You might also have subclassed and installed your own version of the DtPvdMapArea or connected to signals to handle events.

In VR-Forces 4.x you subclass and install a DtEventProcessor. The DtEventProcessor subclasses have methods for mouse and keyboard messages. If you want to handle a mouse or keyboard message, override the appropriate method (`processKeyboardEvent()` or `processMouseEvent()`). If you handle the message you can return true or false. You can also be notified of events that were handled by other event processors before your event processor got a chance, by overriding the above methods and handling the case of a MouseLost event.

Add your event processor subclass into the DtInputDriver class. The DtInputDriver object is referenced through the driver manager:

```
DtInputDriver& inputDriver = myDe.driverManager().inputDriver();
```

You can add the event processor to the front or the back of the list (`addEventProcessorToFront()` or `addEventProcessorToBack()`). You can also get the list of event processors and insert it directly (`eventProcessorList()`). The exampleEventProcessor example has an example of how to use an event processor.

16.8. Changes to Signal Usage

VR-Forces 3.12 used mostly Qt signals. VR-Forces 4.x mostly uses Boost signals. While some Qt signals are used in the Qt layer, they are usually translated back into Boost signals.

To use a Boost signal you must connect to a signal and give a method to call when the signal is invoked.

The following examples show how to connect to signals without parameters (preTick), and with parameters (postTick). The binding is done to an object class and an instance of that object.

```
myDe.signal_preTick.connect(boost::bind(&MyObject::method, this));
myDe.signal_postTick.connect(boost::bind(&MyObject::method,
    this, _1));
```

Note: It is very important to disconnect signals when they are no longer needed or when the object is freed. Failing to disconnect signals can lead to application instability, as, unlike Qt signals, when the object is destroyed, the signal is not disconnected.

16.9. Application Tick Notification

In VR-Forces 3.12, there was no way to be notified when the application was about to tick or had completed a tick. In VR-Forces 4.x, you can get this information. The main class of a VR-Vantage application is DtDe. The DtDe class is typically passed to any object that might need to use it, and is available to any plug-in on startup.

To be notified when the application is about to tick, connect to the signal_preTick signal. On a post tick, connect to the signal_postTick signal.

16.10. Selection Management

In VR-Forces 3.12, to find out when selections had changed, you would connect to the Qt signal sent by the map area on selection changed (selectionUpdated, selectionChanged, and so on). In VR-Forces 4.x, you use the Selection Manager and its signals to be notified when selections have changed. You can get a reference to the Selection Manager as follows:

```
DtSelectionManager& selManager = DtSelectionManager::instance(myDe);
```

The signal_currentSelectionChanged signal is sent any time the current selection is changed. It supplies the following parameters:

- The current selection
- What was unselected
- The new selection type
- The old selection type

The IDs supplied are element IDs, which can be used to retrieve data about the selected objects. Managing Object Selection explains how to use this information.

The signal_vertexSelected and signal_vertexDeselected signals are sent when vertices on a line are selected or unselected.

16.11. Tracking New Objects

In VR-Forces 3.12, to find out when an object was added, you would have used the modelDataAdded signal. In VR-Forces 4.x, you use the DtElementData object. You can get to the DtElementData as follows:

```
DtElementData& elementData = myDe.dataBank().elementData();
```

The signal signal_elementsAdded is sent for any elements added during that tick.

You can then use the State View Collection Manager to find information about the elements added. Due to the nature of the multithreading of the application, it is possible that the state view data does not yet exist for those elements. In that case, for the state view collection you want to retrieve information for, connect to the signal_ stateViewAdded signal. For example:

```
DtVrfStateViewCollectionManager::instance(myDe).baseStateView()
    ->signal_stateViewAdded
```

When the state view is added you can then get information about that object.

16.12. Tracking Object Removals

In VR-Forces 3.12, you would use the modelDataAboutToBeRemoved signal. In VR-Forces 4.x, you use the element data's signal_elementsToBeRemoved signal. This provides a list of elements that are being removed in this tick. See "16.11. Tracking New Objects" above to learn how to retrieve a reference to the DtElementData.

16.13. Simulation Model Set Changes

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

16.13.1 All Entities

- General:
  - All OPE and sysdef files have every instance of "tick-period-uses-real-time" set to False by default now instead of True.
  - A new general parameter has been added to entities to determine whether they can use the new ground rail movement system (see ground entities movement models below). This parameter looks like the following: (using-rail-movement False)
- Weapon Systems. "target-selection" controllers have an additional parameter in the "target-selection-criteria" as follows: (fire-with-spot-report-only False). The parameter determines whether the target selection controller will fire at an entity automatically or not based on the spot report.
- Movement Models:
  - All entity movement systems have an added "preload-terrain-controller" to allow for entities moving in terrains that page data in (such as VR-TheWorld terrains) to page that data in before it gets to the foreseeable un-paged terrain area. Without the controller, entities would page the terrain data in at the time it enters an area where the data has not yet been paged in.
  - Systems including the "path-movement" controller have an added parameter that looks like the following: (register-vrf-specific-tasks True)
- Other Systems:
  - Systems with "spot-report-generator" now have a "radio-name" field as follows: (radio-name "default")
  - Systems with "laser-controller" can be enabled or disabled with the following parameter: (is-enabled True)

16.13.2 Ground Entities

- Damage Models. Ground entities have changed their damage tables against "Guided anti-ship missile damage" tables to use the cruise missile damage instead of the standard missile damage. For instance "heavy-armor-vs-missile.dmg" is now "heavy-armor-vs-cruise-missile.dmg"
- Movement Models:
  - Ground movement systems have been split into various sysdef files to support multiple movement systems, as follows:
    - Vehicles can drive on roads and now use a "combined-movement-system" to switch between them. See "ground-wheels-road.sysdef" and "ground-wheels-road-default.sysdef" and "Technical-Truck_1_1_1_222_27_1_-1_-1.ope".

  - Several systems now include the "rail-path-movement" controller if they are to use the road following options as well as a "system-switch-controller" to switch between regular and rail movement systems.
- Disaggregated movement systems have a "convoy-task-controller" necessary for allowing for convoy movement tasks on disaggregated aggregates.

16.13.3 Air Entities

- Counter Measures. Several air entities have counter measures defined in the systems section of their OPE files. It is usually defined in an "other" field and more commonly "other-4". See "US-fighter-air-defense-fixed-wing-platform_1_1_2_225_1_-1_-1_-1.ope".
- Movement. "flight-kinematics" actuator now has an added parameter for a drag coefficient defined as follows: (drag-coefficient 0.100000)

16.13.4 Lifeform Entities

- General:
  - Several entities now have a "contact-fusion" controller to tie in several sensor systems into a single place. These controllers have associated connections to connect the sensor ports to the contact-fusion controller for entity detection. New lifeforms should use the "contact-fusion" controller.
  - Several entities now have a "target-selection" controller much like other entities. These controllers have associated connections to connect the target selection controller. New lifeform entities should use the "target-selection" controller.
- Movement. All movement systems for lifeforms have removed the "human-patrol" controller.

Index

L

M

N

O

P

R

S

T

U

V

W

MAK TECHNOLOGIES 10 FAWCETT STREET, SUITE 204, CAMBRIDGE, MA 02138 USA | +1 617.876.8085 | www.mak.com