



XII. Creating and Editing Simulation Models

This section explains how to use the Simulation Object Editor (SOE) to create and edit simulation model sets and the simulation models used by simulation objects. It also covers configuration files that you must edit by hand.



2. The Simulation Object Editor and SMSs

The Simulation Object Editor is a standalone tool that you can use to configure simulation objects and tactical graphics, and also create new simulation model sets.

2.1. Introduction to the Simulation Object Editor	5
2.2. Starting the Simulation Object Editor	5
2.3. Simulation Model Sets	9
2.3.1 Including Simulation Model Sets in Other Simulation Model Sets	9
2.3.2 Enabling and Disabling Read Only Status for SMSs	11
2.3.3 SMS Priority for Included and Multiple SMSs	11
2.3.4 Lua Scripts in Included SMSs	12
2.3.5 Application-Level Files in Included SMSs	13
2.3.6 Search Paths for Referenced Files in an SMS	13
2.3.7 Hiding Included Objects	13
2.3.8 Promoting a Model to the Derived SMS	14
2.3.9 Promoting a System to the Derived SMS	15
2.4. Visual Model Definitions	15
2.4.1 User-Created Visual Definition Files	16
2.5. Upgrading Old Simulation Model Sets	17
2.5.1 The Old SMS Includes a Standard SMS	17
2.5.2 The Old SMS Was Copied from a Standard SMS	17
2.5.3 The Old SMS Was Completely New or Copied from base.sms	19
2.5.4 Copying Visual Data	19
2.5.5 Running the SMS Upgrade Tool	19
2.5.6 Migrating an SMS to a New Release by Hand	20
2.5.7 Migrating Pre-4.7 Weapon and Damage Systems	21
2.5.8 Converting Pre-4.8 SMSs To Use Articulated Parts	21
2.6. Opening a Simulation Model Set	21
2.7. Creating New Simulation Model Sets	22
2.7.1 Creating a New SMS that Includes an Existing SMS	22
2.7.2 Creating a New SMS with Minimal Objects	23

2.7.3 Creating a New SMS by Copying an SMS	24
2.8. Importing .entity Files	24
2.9. Applying New and Updated Variables	24
2.10. Importing and Exporting Object Information	25
2.10.1 Exporting Object Information	25
2.10.2 Importing Object Information	25
2.11. Editing the Tags List	26
2.11.1 Adding a New Tag	26
2.11.2 Removing a Tag	27
2.11.3 Renaming a Tag	27
2.11.4 Undoing Tag Changes	28
2.11.5 Using Auto Tags	28
2.11.6 Assigning Tags Based on Visual Model Characteristics	28
2.11.7 Exporting Tags	29
2.11.8 Importing Tags	29
2.11.9 Managing Tags Using the Object Information File	29
2.12. Managing the Forces List	30
2.12.1 Adding a Force	30
2.12.2 Editing a Force	31
2.12.3 Removing a Force	31
2.13. Copying the Object List to the Clipboard	32

2.1. Introduction to the Simulation Object Editor

Using the Simulation Object Editor (SOE), you can create and edit simulation model sets and the objects in them.

Many VR-Forces users want to edit some or all of these simulation object characteristics:

- Weapon systems
- Basic parameters
- Sensors
- 2D icons and 3D models
- Formations

Users also want to add new simulation objects or tactical graphics types. The Simulation Object Editor allows you to edit most of the aspects of a simulation object or tactical graphic that a typical VR-Forces user will want to change. You can easily add and remove sensors, weapon systems, and other major component systems, change the 2D and 3D models used to represent them, and edit their properties.

In addition to editing simulation objects, you can use the Simulation Object Editor to view information about and configure some types of systems.

The Simulation Object Editor does not allow you to add or remove individual components or edit some of the more detailed entity parameters. For these advanced editing tasks, you must edit specific configuration files, which are described throughout this manual.

The Simulation Object Editor primarily allows you to edit properties that affect simulation. Simulation objects also have visualization models, which are configured in the GUI in the Visual Model Editors and Simulation Object Type Mappings dialog boxes. However, you can also use the Simulation Object Editor to configure the 3D models and 2D icons used to display simulation objects.

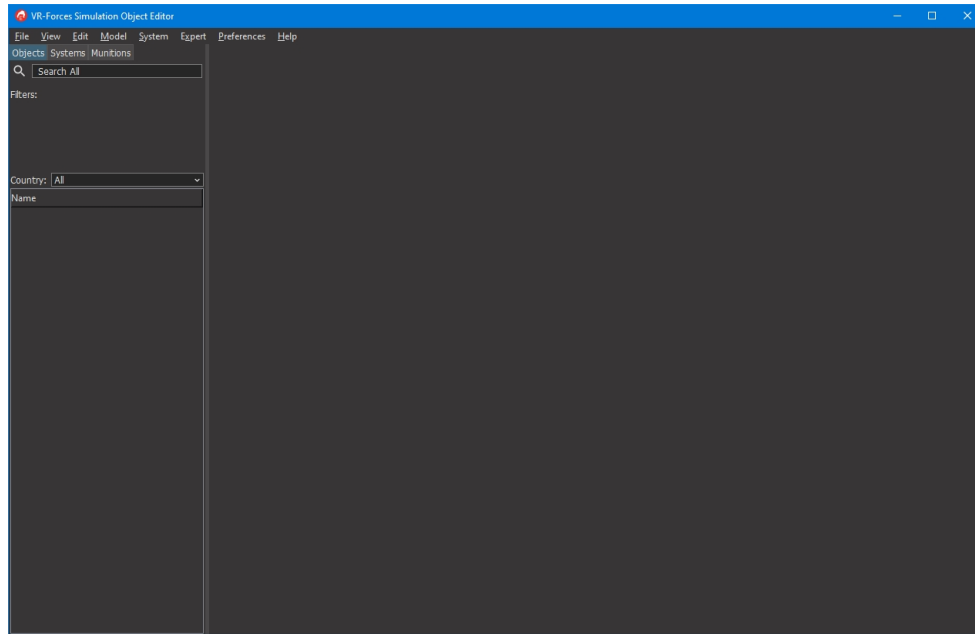


- To edit particle systems, such as smoke and explosions, you use the Particle System Editor. See the [Special Effects \(Particle Systems\)](#) chapter in the *Adding Content to MAK ONE Applications Reference Manual*.
- The Expert menu contains items that are beyond the scope of this documentation and are not fully supported. Those features are intended for use by highly knowledgeable end users, including employees of MAK Technologies.

2.2. Starting the Simulation Object Editor

- To start the Simulation Object Editor in Windows, choose **MAK Technologies**
> **VR-Forces #.# Simulation Object Editor**. The Simulation Object Editor opens (Figure 1).

Figure 1. Simulation Object Editor



► To start the Simulation Object Editor from the command-line in Windows or Linux:

1. Change directory to `./bin64`.
2. Run: `simulationObjectEditor options`

Table 1 describes the command-line arguments for the Simulation Object Editor.

Table 1. Simulation Object Editor command-line arguments

Argument	Description
<code>(-- --ignore_rest)</code>	Ignore all of the command-line options that are listed after this argument. This option is useful for disabling part of a command line in a script for test purposes so that you do not have to retype all of the command options at a later point.
<code>--appDataDir directory</code>	Specifies the location of application data. If not specified, VR-Forces uses the default: <code>./appData</code> .
<code>--dataDir directory</code>	Specifies the location of the application's data directory. If not specified, VR-Forces uses the default: <code>./data</code> .
<code>--doNotCheckPluginVersions</code>	Do not check versions in the plug-ins to make sure they can load against this version of the application.

Table 1. Simulation Object Editor command-line arguments (continued)

Argument	Description
<code>--doNotLoadVrfPlugins</code>	Prevents loading of plug-ins in the <code>./plugins</code> directory.
<code>(-G --locale) language</code>	The name of the language to load into the GUI.
<code>--gui filename</code>	Specify a GUI configuration file. This file specifies the layout of menus, toolbars, and dialog box pages. A simple filename assumes <code>./appdata/settings/vrfGui</code> . A relative filename is from the <code>./bin64</code> directory. A full path is used as is.
<code>(-h --help)</code>	Display command-line usage information in a console window.
<code>--launcherLocation location</code>	Specifies the location of the VR-Forces launcher.
<code>--layoutSettingsFile filename</code>	Specifies the GUI layout settings file to use. <i>filename</i> specifies the base filename. It is prefixed with <i>default_</i> and the extension <i>.uisx</i> is added. Default: <code>SOEUILayoutSettings</code> .
<code>--lightMode</code>	The SOE loads in light mode rather than the default of dark mode.
<code>--limitEntityTextures "width height"</code>	Uses only mipmap levels at or below the resolution for entities. Use two integers for the maximum width and height of the texture mipmap to use, for example, "1024 1024" or "2k 2k". To turn off the texture limit function, use <code>--limitEntityTextures "-1 -1"</code> .
<code>--limitEntityTexturesExclusions "texture"</code>	Excludes textures from texture reduction limits for indirect. Specify the texture(s) that you want to exclude, for example, <code>--limitEntityTexturesExclusions "SurfaceCVN-75HarrySTruman, SurfaceCruiseShip, FixedWingF-16, FixedWingRQ-1Predator"</code> .



In most situations, using `--limitEntityTextures` and `--limitTerrainTextures` prevents crashes with high VRAM. If you continue to experience crashes, use `--limitGpuMemoryPercentage` in addition to these command-line options.

Table 1. Simulation Object Editor command-line arguments (continued)

Argument	Description
<code>--limitGpuMemoryPercentage</code> <i>integer</i>	Set the cutoff percentage as a value between 0 and 100. (Typical values are between 85 and 100.) Beyond the specified percentage, VR-Forces does not make new textures or buffers as a percentage of the available video RAM.
<code>--loadPlugin</code> <i>filename</i>	Load the specified Simulation Object Editor plugin. Can be a specific plug-in or plug-in XML file.
<code>(-n --notifyLevel)</code> <i>notification_level</i>	Specifies the notification level for application messages, where <i>notification_level</i> is a number from 0 through 4. The least verbose message level is 0, the highest is 4. The default is 2. For details, see "Setting the Notification Level".
<code>--settingsDirectory</code> <i>string</i>	Sets the directory to load all UI and settings files from.
<code>--sharedDataDir</code> <i>directory path</i>	Use this to specify the location of the shared data directory if you installed it somewhere other than the default, which is <i>./SharedData</i> .
<code>--sharedDataRoot</code> <i>directory root</i>	Specifies the location of the shared data root. This combined with the stem and data version creates the shared data path.
<code>--sharedDataStem</code> <i>directorystem</i>	Specifies the location of the shared data stem. This combined with the data root and data version creates the shared data path.
<code>--sharedSettingsDirectory</code> <i>directory</i>	Specifies the directory where shared settings are stored. The default directory is <i>\$(APP_DIR)/settings</i> . If the <code>--useUserSettingsDirectory</code> flag is set, then the user settings directory is used to store the settings instead.
<code>(-c --showConsole)</code>	Show the console.
<code>--smsFile</code> <i>string</i>	Loads the specified SMS file.
<code>--useUserSettingsDirectory</code>	Indicates that the system should store settings in the user login settings directory, rather than the application settings directory. Use in conjunction with <code>--sharedSettingsDirectory</code> .
<code>(-v --version)</code>	Displays version information and exits.
<code>--visualTerrain</code> <i>string</i>	Terrain used when displaying simulation objects.

2.3. Simulation Model Sets

A simulation model set (SMS) defines all of the simulation objects, tactical graphics, and other object models that you can create in a VR-Forces simulation. The simulation model set file tells VR-Forces where to find all of the configuration files it needs to create, simulate, and display simulation objects and other objects. Every scenario must reference one or more simulation model sets.

If you use the Simulation Object Editor to edit simulation object models, the only configuration file that you need to be aware of is the simulation model set file that you are using. All of the other files are handled by the Simulation Object Editor.

VR-Forces includes these SMSs:

- *base.sms*. Defines platforms and graphics used by other SMSs.
- *EntityLevel.sms*. Defines entities, units, and other objects used for entity-level scenarios. Includes *base.sms*.
- *AggregateLevelBase.sms*. Defines general units and other objects used by other aggregate-level SMSs. Includes *base.sms*.
- *AggregateLevel.sms*. Legacy SMS preserved for backward compatibility. Includes *base.sms* and *AggregateLevelBase.sms*.
- *MAKTest.sms*. Defines some fictional entities for testing entity-specific features such as lighting. Includes *base.sms* and *EntityLevel.sms*.

For details about entity- and aggregate-level modeling, see "Entity-Level Modeling and Aggregate-Level Modeling".



When you use the Simulation Object Editor, you are editing a simulation model set. We strongly recommend that you keep the default SMSs as reference model sets and not edit them. If you want to edit entity models, create a new SMS that includes one of the provided SMSs. Promote the objects you want to edit to the new SMS and use it as the default SMS for your scenarios. For details about creating a new SMS and changing the default SMS, see "2.7. Creating New Simulation Model Sets" on page 22.

2.3.1 Including Simulation Model Sets in Other Simulation Model Sets

VR-Forces simulation model sets can include other SMSs. This allows you to create core SMSs that can be included in multiple other SMSs that may have incompatible characteristics or which override some characteristics of the core SMS, but which still need core objects. VR-Forces provides several SMSs, described in "2.3. Simulation Model Sets" above.

It is important to understand the difference between using an SMS that includes other SMSs and creating a scenario that uses multiple SMSs. Since VR-Forces users usually create new SMSs to configure new types of simulation objects, or add new systems to existing objects, we will consider the differences in that context.

If you want to create a custom standalone SMS for a new or enhanced object, that SMS must include all of the files required to support that object, such as an OPE file, entity file, system files, damage files, and so on. When trying to build such an SMS, it can be easy to overlook required files. This process also requires you to duplicate files that are contained

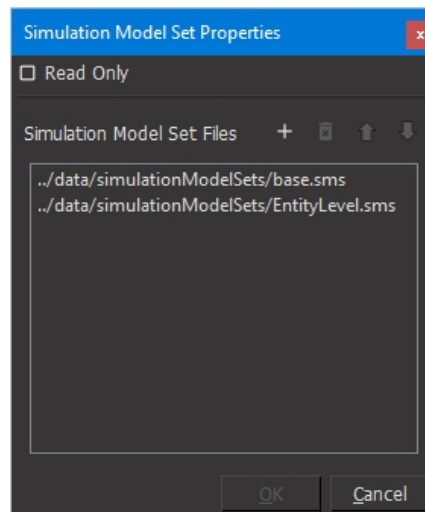
in the supplied SMSs, even if you do not need to change them, because the SMS must contain all the files it needs. Then, if you do not load the SMSs in the correct order, your customization can be overridden by one of the other SMSs you are using in the scenario. In summary, it is a cumbersome process that can create needless duplication of files and be difficult to debug if problems arise.

Using included SMSs minimizes the problems inherent in using multiple standalone SMSs. When an SMS includes another SMS, it has a higher priority than the included SMS. When an SMS is loaded, included SMSs are loaded first, and each subsequent SMS has a higher priority than the previously loaded SMS. Any parameters or files in the higher priority SMSs that are also in the lower priority SMSs override those in the lower priority SMSs. The higher priority SMSs do not need to contain any files that they do not change from the included SMSs. Therefore the total footprint of the SMS is smaller. For more information about SMS priority, see "2.3.3 SMS Priority for Included and Multiple SMSs" on the facing page.

► **To include an SMS in another SMS:**

1. Choose **MAK Technologies > VR-Forces 5.2 Simulation Object Editor**. The Simulation Object Editor opens (Figure 1).
2. Choose **File > Open Simulation Model Set**. The Open Model Set dialog box opens.
3. Select the SMS where you want to include another SMS.
4. Click Open. The SMS loads.
5. Choose **File > Properties**. The SMS Properties dialog box opens.

Figure 2. SMS Properties dialog box



6. Click the Add button (+). A line is added to the window with the text <empty file>.
7. Click the Browse button at the end of the line. The Choose File dialog box opens.
8. Select the SMS that you want to include.
9. Click Open. The SMS is added to the list in the window.
10. Click OK. The loaded (derived) SMS is saved and reloaded.

2.3.2 Enabling and Disabling Read Only Status for SMSs

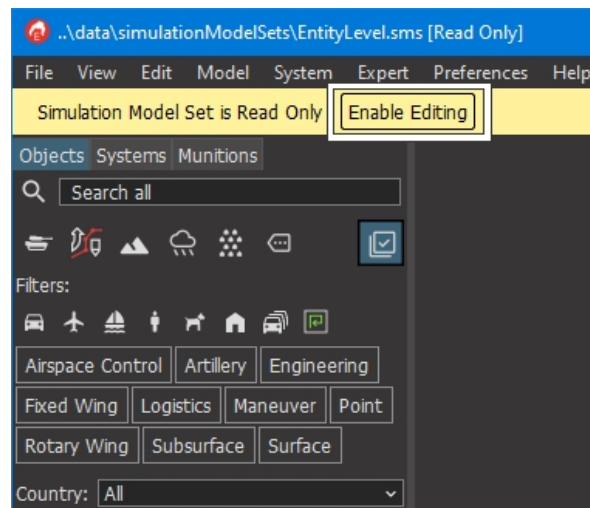
By default, the SMSs provided with VR-Forces are read-only. You can enable and disable editing.

i Read-only status is to encourage you to create new SMSs for your customizations to the SMSs provided by VR-Forces. Leaving the default SMSs unedited simplifies upgrades to future versions of VR-Forces.

► To enable editing of the objects in an SMS, do one of the following:

- Click the Enable Editing button (Figure 3). You are prompted to create a new SMS that includes the current one, to enable editing for this and future sessions, or to enable editing for this session only. If the current SMS is one of the three basic SMSs provided by VR-Forces, we recommend that you create a new SMS.

Figure 3. Enable Editing



- Choose **File > Properties** and clear the Read Only check box in the SMS Properties dialog box.

► To enable read-only status for an SMS:

1. Choose **File > Properties**. The SMS Properties dialog box opens.
2. Select the Read Only check box.

2.3.3 SMS Priority for Included and Multiple SMSs

When you load multiple SMSs, either through inclusion or by specifying multiple SMSs for a scenario, conflicts are resolved through the priorities of the SMSs.

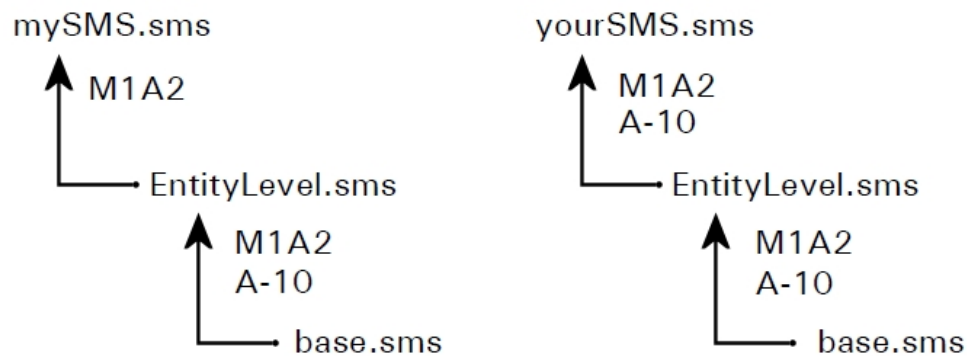
For included SMSs, priority is based on the inclusion relationships. The highest level SMS is loaded last and has the highest priority. Simulation objects in higher priority SMSs override those in lower priority SMSs. For example, suppose you create a new SMS called *mySMS.sms* that includes *EntityLevel.sms*. *EntityLevel.sms* already includes *base.sms*. When

VR-Forces loads *mySMS*, it loads *base.sms* first, then *EntityLevel.sms*, and finally *mySMS.sms*. Objects in *mySMS.sms* override objects in *EntityLevel.sms* that have the same object type, which in turn overrides *base.sms*.

When you load multiple SMSs, priority is based on the order in which they are loaded. Assuming a simple case of two SMSs that do not include other SMSs, if the second SMS loaded specifies objects that are also in the first SMS loaded, the specification in the second SMS takes precedence.

If you load multiple SMSs that include other SMSs, priority is based on the order loaded, but duplicate SMSs are not loaded twice. Assume the SMS organization shown in Figure 4.

Figure 4. Multiple SMSs



If *mySMS.sms* is the first SMS loaded, VR-Forces loads *base.sms*, then *EntityLevel.sms*, then *mySMS.sms*. Priority is determined as described previously for an SMS with included SMSs. The M1A2 entity configured in *mySMS.sms* overrides the one in *EntityLevel.sms*.

Then VR-Forces loads *yourSMS.sms*. It starts with *base.sms*. Since *base.sms* is already loaded, VR-Forces ignores it. Then it moves to *EntityLevel.sms*, which is also already loaded and therefore VR-Forces ignores it, as well. Then it loads *yourSMS.sms*. The M1A2 in *yourSMS.sms* overrides the configuration in *mySMS.sms*. The A-10 in *yourSMS.sms* overrides the A-10 in *EntityLevel.sms*.

The overall priority for the SMSs in this example, from lowest to highest, is *base.sms*, *EntityLevel.sms*, *mySMS.sms*, *yourSMS.sms*.

The priority of multiple SMSs is determined by the order in which they are listed in the **Simulation-Model-Set-Files** parameter in the scenario file. In the following example, *EntityLevel.sms* has the highest priority. When you create a scenario and list multiple SMSs, the priority in the SMS list is top to bottom.

```
(Simulation-Model-Set-Files
  "..\data\simulationModelSets\EntityLevel.sms"
  "..\data\simulationModelSets\developer_toolkit_
examples\addTask.sms")
```

2.3.4 Lua Scripts in Included SMSs

System scripts are stored in simulation model sets. When you load an SMS that includes another SMS, the scripts in the highest priority SMS supersede those in the lower priority SMSs.

All simulation objects use the highest priority scripts, regardless of which SMS they are in. For example, suppose *EntityLevel.sms* has a script called Patrol Ground Area and *mySMS.sms* (the higher priority SMS) also has a script called Patrol Ground Area. If the entity M1A2, which is defined in *EntityLevel.sms* executes the Patrol Ground Area task, it uses the version in *mySMS.sms*.

Lua scripts in included SMSs have additional relationships:

- Any scenario scripted task promoted to a system scripted task is promoted into the highest-priority SMS.
- If you edit a scripted task that is in an included SMS, it is saved into the highest-priority SMS. The version of the script in the included SMS does not change. Therefore if you want a change to a script to affect all uses of that script, edit it in its own SMS.

2.3.5 Application-Level Files in Included SMSs

Application-level SMS files, such as *commModelParams.mtl* or *gui/initialCreateMenuConfiguration.xml*, are included the same way entity-specific files are and you do not need to have copies of them in an derived SMS as long as one of the included SMSs has them. However, every SMS must have *vrSim.opd*.

2.3.6 Search Paths for Referenced Files in an SMS

The simulation objects in a scenario reference many files to determine their behavior, such as system definition files, scripts, platform files, and so on. If all the simulation objects in a scenario are configured in one SMS, then all of the referenced files are within the file hierarchy of that SMS. However, if you use an SMS that includes other SMSs, then simulation objects may have to search within multiple SMSs to reference the correct files.

An SMS can load files from any SMS it includes, and any currently loaded SMS it is included in.

File referencing, as with other aspects of included SMSs, is based on the priority of the SMS in which an object is configured. Given the SMSs in Figure 4 on the previous page, the search paths for each SMS from lowest to highest priority would be:

- *base.sms*. Can search in *base.sms*, *EntityLevel.sms*, *mySMS.sms*, and *yourSMS.sms*.
- *EntityLevel.sms*. Can search in *base.sms*, *EntityLevel.sms*, *mySMS.sms*, and *yourSMS.sms*.
- *mySMS.sms*. Can search in *base.sms*, *EntityLevel.sms*, and *mySMS.sms*.
- *yourSMS.sms*. Can search in *base.sms*, *EntityLevel.sms*, and *yourSMS.sms*.

2.3.7 Hiding Included Objects

By default, the Simulation Object Editor lists the simulation objects that are part of included SMSs. However, you can hide included objects if you just want to see the objects in the top-level SMS.

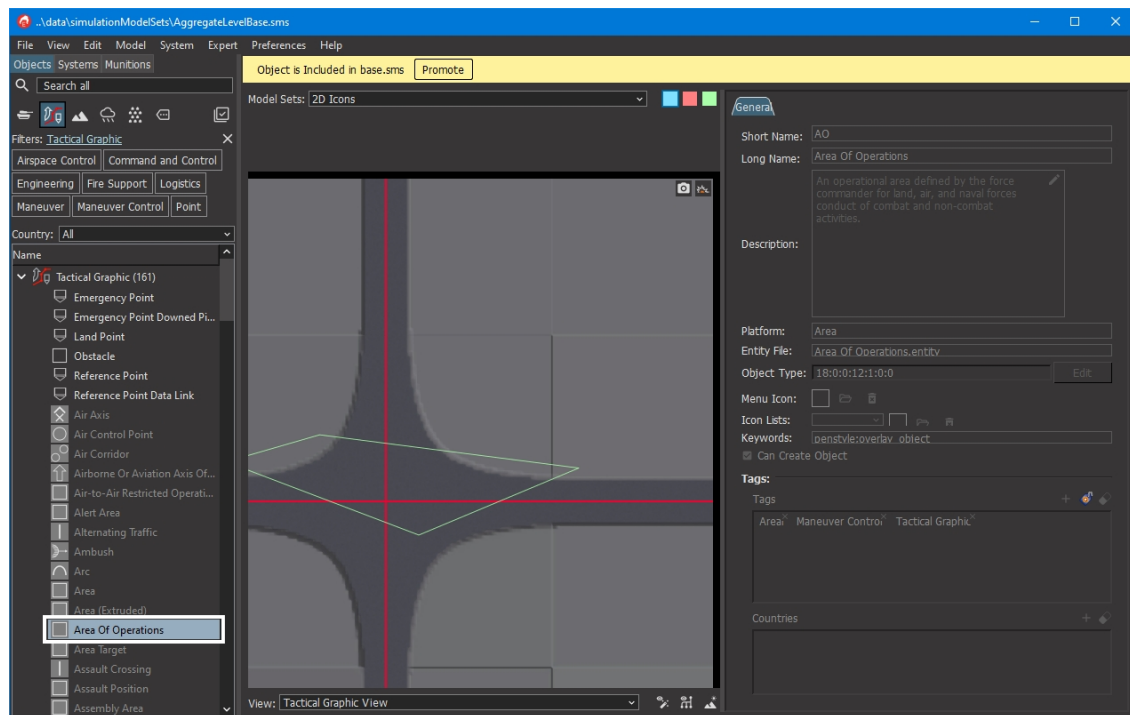
- To hide or display included objects in the objects list, choose **Preferences > Show Included Simulation Objects**.

2.3.8 Promoting a Model to the Derived SMS

The preferred approach when you need to customize something from an included SMS is to promote it for edits in the derived SMS.

When you load an SMS that has included other SMSs, the objects from the included SMSs are listed but you cannot edit them. Without promoting them, you can only edit them by editing the SMS to which they belong. Figure 5 shows *AggregateLevelBase.sms* filtered for tactical graphics. The objects at the end of the list are not editable because they are included from *base.sms*. Note in the figure that most of the parameters for the selected object are read-only.

Figure 5. Included simulation object



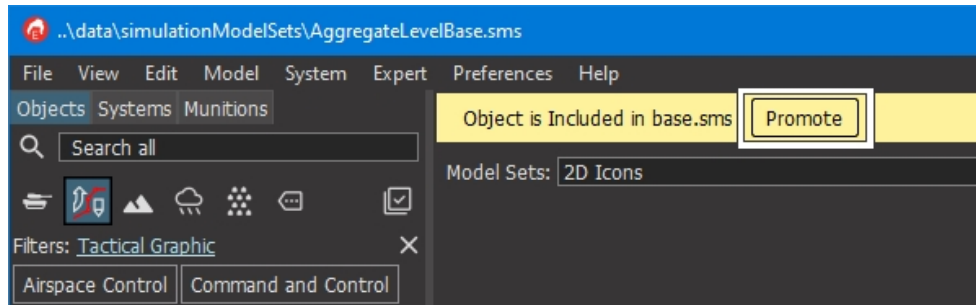
Since one of the purposes of being able to include SMSs in the loaded (derived) SMS is to customize simulation objects in the included SMSs, you can promote objects from the included SMS to the loaded SMS. The object then becomes editable and is saved in the loaded SMS. When the derived SMS is loaded, the object in the loaded SMS overrides the object in the included SMS.

► **To promote an object to the derived SMS:**

1. Load the derived SMS.
2. Choose **Preferences > Show Included Simulation Objects** if it is not already enabled.
3. On the Objects tab or Munitions tab, select the object you want to promote. (Although munitions are on a separate tab, they are also simulation objects.)
4. Choose **Model > Promote to Main Simulation Model Set** or click the Promote button (Figure 6) (above the parameter configuration tabs). The object is added to the

derived SMS and is now fully editable. When you save the SMS, a new *.entity* file is added to it.

Figure 6. Promote button



2.3.9 Promoting a System to the Derived SMS

Just as with simulation objects ("2.3.8 Promoting a Model to the Derived SMS" on the previous page), if you want to edit a system in an included SMS, you must promote it to the loaded (derived) SMS.

► **To promote an object to the derived SMS:**

1. Load the derived SMS.
2. Choose **Preferences > Show Included Simulation Objects** if it is not already enabled.
3. Select the system you want to promote.
4. Choose **System > Promote to Main Simulation Model Set** or click the Promote button (Figure 6 above). The system is added to the derived SMS and is now fully editable.

2.4. Visual Model Definitions

VR-Forces stores visual model definitions in different places depending on which SMS is loaded:

- When the Simulation Object Editor loads one of the SMSs that ship with VR-Forces (for example, *base.sms*, *EntityLevel.sms*, *AggregateLevel.sms*, or *AggregateLevelBase.sms*), it reads the appropriate definition files for the SMS from the directories under *./SharedData/##/latest/ModelData/Configuration/Definitions*. When you save changes to one of these SMSs, the changes are written back to the original definition files.
- If you create a new SMS, model and element definition files are saved with the SMS in the *./data/simulationModelSets/<sms>/gui/visuals* directory for the new SMS. When you load the SMS, it reads model and element definitions from the SMS-specific files.

For information about model and element definition files, see the [Model Definitions](#) and [Element Definitions](#) chapters in the *Adding Content to MAK ONE Applications Reference Manual*.

2.4.1 User-Created Visual Definition Files

Although we recommend that you use the Simulation Object Editor for all changes to visual models, it is possible to add new model definitions and element definitions using the Visual Model Editors dialog box instead of the Simulation Object Editor. Additionally, some attributes of element definitions cannot be set in the Simulation Object Editor, so if you want to change them, you must use the Visual Model Editors.

You can export the new and changed definitions to files other than the default set of definition files. In fact, in many cases, we recommend that you export them to their own files. However, creating new model or element definitions in the Visual Model Editors and saving them to new files has implications for how they are loaded and saved by the Simulation Object Editor.

When the Simulation Object Editor loads one of the five default SMSs, it loads all files in the appropriate directories, including any additional files that you have created. However, when VR-Forces saves changes to the SMS, it writes the changes only to the default definition files for the SMS. It does not write back to the user-created definition files. Since the user-created files are still in the directory, the next time you load the SMS, there may be conflicts between the files and the results may be unexpected.

Given this behavior, the recommended workflow for reconciling new definitions created in the Visual Model Editors dialog box is:

1. In the Visual Model Editors dialog box, save new model or element definitions to new files in the default definition directories.
2. Open the affected SMS in the Simulation Object Editor.
3. Confirm that your new model or element definitions have been loaded by the SMS.
4. Save the SMS. This saves the new definitions into the default set of SMS definition files.
5. Delete the individual definition files that you created in step 1 so that there are no conflicts with the SMS.

If you create a new SMS, when the Simulation Object Editor loads the SMS, it only loads the definition files that it previously created. If you create additional leaf or model definition files with the Visual Model Editors dialog box, they are not loaded. You must import them into your SMS. Therefore, the recommended workflow is:

1. Open a scenario that uses your SMS.
2. In the Visual Model Editors dialog box, load the *.hier* file for your SMS. It is either *./SharedData/##/latest/ModelData/Configuration/Definitions/Aggregate/ElementDefinitions/coreAggregateElementDefinitions.hier* or *./SharedData/##/latest/ModelData/Configuration/Definitions/Entity/coreEntityElementDefinitions.hier*.
3. Import the model definition or element definition file to which you want to add new definitions. Be sure to import it from the files in the */data/simulationModelSets/<sms>/gui/visuals* directory for your SMS. Be sure that you only edit one element definition file at a time.
4. Add new definitions as desired.

5. Export the new definitions to the definition file that you imported.
6. Open the SMS in the Simulation Object Editor and confirm that the new definitions are available.

2.5. Upgrading Old Simulation Model Sets

The Simulation Object Editor has an upgrade tool that can upgrade SMSs created in VR-Forces 4.4 or later to the current release. The upgrade process varies depending on how you created the SMS that you are upgrading. The possibilities it takes into account are:

- The old SMS was created by including a standard SMS (*EntityLevel.sms*, *AggregateLevelBase.sms*, or *AggregateLevel.sms*).
- The old SMS was created by copying one of the standard SMSs.
- The old SMS was a new SMS that did not copy or include a standard SMS. It could have included or be a copy of *base.sms*.

The following sections describe the upgrade process for each of these possibilities. If you need to upgrade an SMS from VR-Forces 4.3.x or earlier, or want to upgrade your SMS manually, see "2.5.6 Migrating an SMS to a New Release by Hand" on page 20.

2.5.1 The Old SMS Includes a Standard SMS

If you created the old SMS by including a standard SMS, the assumption is that you made no changes to the included SMS. You edited your SMS by promoting simulation objects from the standard SMS and editing them, and by adding new simulation objects in your SMS. You may also have made other changes to the SMS, such as adding scripts and adding or editing system definition files.

Since you did not touch the standard SMS, the upgrade tool assumes that you will simply include the standard SMS that is provided in the new version of VR-Forces and that you want to keep all the changes you made when creating the old SMS. The upgrade wizard does the following:

- Copies all files from the old SMS (except visual data as described in the next paragraph).
- If the old SMS is from VR-Forces 4.5 or later and you specify in the upgrade wizard that you want to copy visual data, it is copied to the new SMS. If you specify that you do not want to copy visual data, no visual data files are copied. (For more information about copying visual data, see "2.5.4 Copying Visual Data" on page 19.)
- Compares the files that were promoted into the old SMS to files of the same name in the new included SMS. If the files are different, these files are listed as needing investigation.

2.5.2 The Old SMS Was Copied from a Standard SMS

Upgrading an SMS that was created by copying a standard SMS (*EntityLevel.sms*, *AggregateLevelBase.sms*, or *AggregateLevel.sms*) is more complex than upgrading an SMS that was created by including an SMS. The upgrade tool tries to preserve all changes made to the files in the old SMS while incorporating the improvements made to the new version of the standard SMS. However, it errs on the side of keeping changes made to files in the

old SMS. This means that if you want your SMS to take advantage of the improvements to the new standard SMS, you may have to compare files by hand. If changes have been made to platform files, you can quickly capture those changes by selecting Apply Platform Changes in the Simulation Object Editor.

The upgrade tool does the following:

- It generates checksums for the files in the old SMS. It compares these checksums to checksums for the files in the standard SMS in the old version of VR-Forces and to the files in the standard SMS in the version of VR-Forces you are upgrading to. It moves files to the new SMS based on these comparisons. Table 2 describes the possible results of these comparisons.

Table 2. Decision table for comparing SMS files

	Yes	Notes
Is the file the same in old standard SMS, old SMS, and new standard SMS?	Copy file to new SMS.	
Is the file the same in old standard SMS and old SMS, but changed in new standard SMS?	Copy file from new standard SMS to new SMS.	Optionally, run Apply Platform Updates to capture changes in OPE files.
Is the file only in old SMS?	Copy file from old SMS to new SMS.	Optionally, run Apply Platform Updates to capture changes in OPE files.
Is the file in old standard SMS and old SMS, but removed from new standard SMS?	If the file was changed in old SMS, copy it to new SMS. Otherwise, remove it.	Optionally, run Apply Platform Updates to capture changes in OPE files.
Is the file only in new standard SMS?	Add this file to the new SMS.	
Did the file change from the old standard SMS to the old SMS and from the old standard SMS to the new standard SMS?	Copy file from old SMS to new SMS.	Consider merging in changes from the file in the new standard SMS.
Is the file the same in old standard SMS and new standard SMS, but different in old SMS?	Copy file from old SMS to new SMS.	Optionally, run Apply Platform Updates to capture changes in OPE files.

- If the old SMS is from VR-Forces 4.5 or later and you specify in the upgrade wizard that you want to copy visual data, it is copied to the new SMS. If you specify that you do not want to copy visual data, no visual data files are copied. (For more information about copying visual data, see "2.5.4 Copying Visual Data" on the facing page.)

- The upgrade tool provides a report of files that have conflicts, files that were added, and files that were removed. You can use this report to determine what files you want to examine for manual upgrades.

2.5.3 The Old SMS Was Completely New or Copied from *base.sms*

In rare cases, VR-Forces users may create an SMS by copying or including only *base.sms*, by selectively copying files from one of the standard SMSs, or by creating all the files themselves. In this case, there is no reference for the upgrade process. The upgrade tool does the following:

- Copies all files from the old SMS (except visual data as described in the next paragraph).
- If the old SMS is from VR-Forces 4.5 or later and you specify in the upgrade wizard that you want to copy visual data, it is copied to the new SMS. If you specify that you do not want to copy visual data, no visual data files are copied. (For more information about copying visual data, see "2.5.4 Copying Visual Data" below.)
- Reports files that might need to be reviewed, such as system definitions and OPE files.

2.5.4 Copying Visual Data

In VR-Forces 4.4.x, all visual model data was stored in files in *./appData/definitions* and in *./appData/settings/vrfGui*. Visual data was not specific to an SMS. In VR-Forces 4.5 and later, visual data files were restructured and an SMS can have visual data that is specific to the SMS.

For SMSs created in VR-Forces 4.5 or later, the upgrade tool gives you the option of copying visual data from the old version of VR-Forces. If you choose to copy visual data and the old SMS has SMS-specific visual data, the files from the SMS are copied to the new SMS. If the old SMS did not have SMS-specific data, the files from *./SharedData/###/latest/ModelData/Configuration/Definitions* are copied to the new SMS.

If you choose not to copy visual data, the new SMS uses the visual data provided in the new version of VR-Forces.

If you copy visual data to the new SMS, it will take longer to load the SMS when you open a scenario.

2.5.5 Running the SMS Upgrade Tool

► To upgrade an SMS to a new version of VR-Forces:

1. Open the Simulation Object Editor.
2. Choose **File > Upgrade Old Simulation Model Set**. The Upgrade Simulation Model Set dialog box opens. (The upgrade process varies, depending on the type of SMS you are upgrading.)
3. Select the SMS that you are upgrading (the old SMS).
4. Specify a name for the upgraded SMS. The default is the name of the old SMS.
5. Click Next.

- If the old SMS was created by copying a standard VR-Forces SMS or was a completely original SMS, the Original Simulation Model Set page displays.
 - If the old SMS was created by including a standard VR-Forces SMS, the Simulation Model Set Version page displays. Skip to step 8.
 - If the old SMS was created by copying a standard SMS, select the standard SMS it was copied from.
 - If the source is completely original or was copied from *base.sms*, select Neither. (SMSs created by including standard VR-Forces SMSs do not see this page.)
6. Click Next.
 7. The Simulation Model Set Version page displays.
 8. Select the version of VR-Forces used to create the old SMS.
 9. Click Next.
 - If the old SMS was created with VR-Forces 4.4 or 4.4.2, the upgrade process runs and a report is generated.
 - If the old SMS was created with VR-Forces 4.5 or later, the Visual Data page displays.
 10. Specify whether to copy visual data.
 11. Click Next. The upgrade process runs and generates a report. You can save the report ()
 12. When you are done, click Finish.

2.5.6 Migrating an SMS to a New Release by Hand

If you need to migrate an SMS that was created in VR-Forces 4.3.x or earlier or do not want to use the upgrade tool, you can migrate your old SMS by hand. You can do this by copying your customized files to a new SMS or by recreating your changes in a new SMS.

When you edit an SMS in the Simulation Object Editor, it:

- Creates or edits *.entity* files for the simulation objects that you edit.
- Modifies visual definition files and entity type mappings.

The Simulation Object Editor does not create or change any other files. However, there are many other files in an SMS that could be customized, such as system definitions, the various damage and ammo select files, and system scripts.

If you upgrade to a new version of VR-Forces, you can quickly add any of the new simulation objects you created by copying their *.entity* files into an SMS in the new release. This will make the simulation objects available to the simulation engine, but will not include their visual definitions. For each entity that you add this way, you will have to edit the 2D and 3D models in the Simulation Object Editor.

If you made any changes to other files, you may have to copy those changed files to the new SMS.

If you have written scripts, use the script export process to export them from the previous release of VR-Forces and then import them into the new release.

Since SMSs often have significant changes from one release to the next, the older your legacy SMS is, the harder it will be to upgrade it simply by copying changed files. We recommend that you review the release notes for interim releases to understand the changes that you might need to account for when migrating old files. You can view the release notes for each release on the Product Specifications & Dependencies page for VR-Forces at <https://www.mak.com/mak-one-specs/vr-forces>.

There are several major SMS divergence points to be aware of:

- In VR-Forces 4.2, *default.sms* was reorganized as *base.sms* and *EntityLevel.sms*.
- In VR-Forces 4.5, visual model files were restructured.
- In VR-Forces 4.7, weapon system and damage system files were changed significantly so that they could be edited in the Simulation Object Editor.

2.5.7 Migrating Pre-4.7 Weapon and Damage Systems

Due to changes to the system definition files to support editing weapon and damage systems in the SOE, the SMS upgrade tool does not migrate weapon and damage systems from the VR-Forces 4.6.1 and earlier format to VR-Forces 4.7 or later. If you have modified a system using the legacy file formats and want to replicate it in the current release, you have two options:

- Copy the old system files to the new version of VR-Forces. VR-Forces continues to support the VR-Forces 4.6.1 and earlier format. If you copy the old files to the current SMS, you will not be able to edit them in the SOE.
- Use the SOE to edit an existing system or create a new system to match the system that you customized in the older version of VR-Forces.

 There are more details about the changes to weapon and damage systems in the *VR-Forces Migration Guide*.

2.5.8 Converting Pre-4.8 SMSs To Use Articulated Parts

In VR-Forces 4.8, entities' articulated parts were enhanced. You can add and adjust them in the Simulation Object Editor, and, using the Set menu, you can set the appearance of articulated parts that are not managed by another controller. To take advantage of this feature, you must update the *.entity*, *.sysdef*, and *.ope* files as described in the *VR-Forces Migration Guide*.

To learn more about editing articulated parts in the Simulation Object Editor, see "4.3. Editing a Model's Articulated Parts" on page 85.

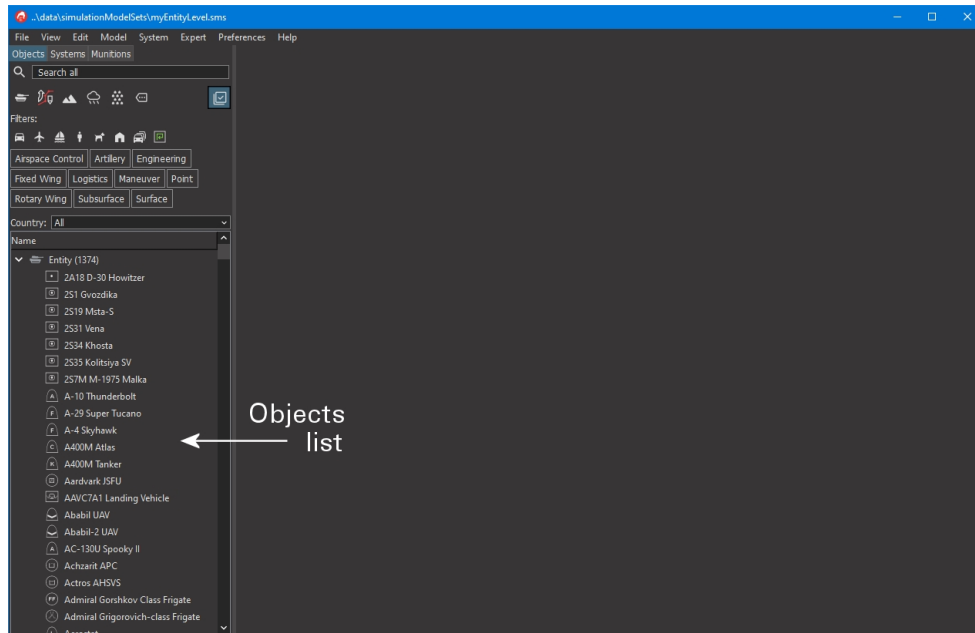
2.6. Opening a Simulation Model Set

To edit simulation object models in the Simulation Object Editor, you must open a simulation model set.

► To open a simulation model set:

1. Choose **File > Open Simulation Model Set**. The Open Model Set dialog box opens.
2. Select the SMS you want to open.
3. Click Open. A progress bar indicates what files are loading. Once the SMS loads completely, a list of simulation objects displays in the Simulation Object Editor.

Figure 7. Simulation Object Editor with simulation model set loaded



2.7. Creating New Simulation Model Sets

There are a couple ways to create a new SMS. You can create an SMS that contains no simulation objects but includes an existing SMS, or you can create a new SMS that contains only the most basic objects that VR-Forces requires. You can also create an SMS by copying an existing SMS. In the first two cases, you would add new objects or promote objects from the included SMSs into the derived (loaded) SMS. In the third case, you would edit the simulation objects already in the SMS and add new ones as desired.

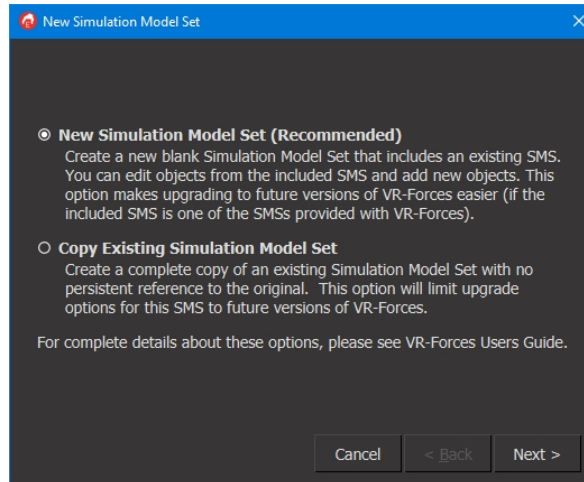
To simplify upgrading to future versions of VR-Forces, the recommended choice is to create a new simulation model set by including an existing SMS.

2.7.1 Creating a New SMS that Includes an Existing SMS

The recommended method is to use an existing set that includes models that work in VR-Forces, and then customize it by adjusting the existing models or adding new ones.

► To create a new simulation model set based on an existing one:

1. Choose **File > New Simulation Model Set**. The New Simulation Model Set dialog box opens (Figure 8).

Figure 8. New Simulation Model Set dialog box

2. Select New Simulation Model Set.
3. Click Next. The Simulation Model Set Type dialog box opens.
4. Select Extend Existing Simulation Model Set.
5. Click Next. The Extend Existing Simulation Model Set dialog box opens.
6. Select the SMS to use as the basis for your new SMS.
7. Click Next. The New Model Set Name dialog box opens.
8. In the File name box, enter a name for the new model set.
9. Click Finish. The Simulation Object Editor loads systems information from the selected model set.
10. Edit the SMS as desired.
11. Choose **File > Save Simulation Model Set**.

2.7.2 Creating a New SMS with Minimal Objects

Every SMS must include a basic set of files, such as platform files, GUI configuration files, and system definitions. You ensure that your new SMS has those files by including an existing SMS in it. The minimum set of files is provided by *base.sms*.

► **To create a new simulation model set with minimal objects:**

1. Choose **File > New Simulation Model Set**. The New Simulation Model Set dialog box opens (Figure 8 above).
2. Select New Simulation Model Set.
3. Click Next. The Simulation Model Set Type dialog box opens.
4. Select Minimal Simulation Model Set.
5. Click Next. The New Model Set Name dialog box opens.
6. In the File name box, enter a name for the new model set.
7. Click Finish. The Simulation Object Editor loads basic object information.

8. Edit the SMS as desired.
9. Choose **File > Save Simulation Model Set**.

2.7.3 Creating a New SMS by Copying an SMS

You can create a new SMS by copying an existing SMS. When you do this the new SMS includes any SMSs that were included by the original SMS. All of the simulation objects in the new SMS are editable.

► **To create a new simulation model set by copying an SMS:**

1. Choose **File > New Simulation Model Set**. The New Simulation Model Set dialog box opens (Figure 8 on the previous page).
2. Select Copy Existing Simulation Model Set.
3. Click Next. The Select Model Set to Copy dialog box opens.
4. Select a model set and click Next.
5. Click Next. The New Model Set Name dialog box opens.
6. In the File name box, enter a name for the new model set.
7. Click Finish. The Simulation Object Editor loads information from the source model set.
8. Make any changes you want to the model set. (For details about how to add and remove included SMSs, see the procedure in "2.3.1 Including Simulation Model Sets in Other Simulation Model Sets" on page 9, starting with step 6 (page 9).)
9. Choose **File > Save Simulation Model Set**.

2.8. Importing .entity Files

If you have *.entity* files that you created with a previous version of VR-Forces or by multiple persons on different computers, you may want to import them into an SMS. The easiest way to do this is to copy the files into the *vrSim* directory in the SMS (*./data/simulationModelSet/<model_set>/vrSim*). However, you can also import the files from the Simulation Object Editor.

► **To import .entity files into an SMS:**

1. Choose **File > Add Entity File**. The Add Entity File dialog box opens.
2. Select the file that you want to import.
3. Click Open. The file is copied to the SMS.

2.9. Applying New and Updated Variables

All simulation object models reference one of the basic platforms, such as ground, fixed-wing, and point object. Most simulation objects reference one or more system definition files. If you update the variable bindings in a platform file or system definition file, you must apply these changes to the simulation objects of that platform type. Although you could do this by hand, as described in "9.5. Adding Variable Bindings to Platform and System Files" on page 183, that would be a tedious and error-prone activity. The Simulation Object Editor can apply all updates to all files automatically.

Similarly, if new variables are added to the API in descriptor files, these variables must be added to the system definition files and OPE files.

► **To apply new and updated variable bindings:**

1. Choose **File > Regenerate SMS Files**. The Regenerate SMS Files dialog box opens.
2. Select the check box for each update you want:
 - Entity File Updates. All new variable bindings in the OPE files and systems are added as elements in the appropriate *.entity* files.
 - OPE File Updates. Applies changes to platform file descriptors to the OPE files.
 - System File Updates. Updates system files and system template files with any new variables that have been added to the system file descriptors.
3. Click OK.

2.10. Importing and Exporting Object Information

You can export a list of all simulation objects in an SMS to a CSV file. You can open the CSV file in a spreadsheet application. You can edit some of the object information and then import the updated object information back into the SMS. This format lets you quickly edit information without selecting objects one-by-one in the Simulation Object Editor.

For each object in the SMS, the object information file lists the object type enumeration, name, short name and filtering tags. You can edit the name, short name and tags.



Although you can also edit the object type, doing so will break the relationship between the limited information in the spreadsheet and the complete configuration stored in the SMS. It is strongly recommended that you not edit the object type.

2.10.1 Exporting Object Information

► **To export object information:**

1. Choose **Model > Export Object Information**. The Export Object Information dialog box opens.
2. Enter a name for the file.
3. Click Save.

2.10.2 Importing Object Information

► **To import object information:**

1. Choose **Model Import Object Information**. The Import Object Information dialog box opens.
2. Select the file you want to import.
3. Click Open.

2.11. Editing the Tags List

Tags allow you to organize and filter simulation objects in VR-Forces. You can also search by tag. If you want to organize simulation objects into tags that are not included in the default tags list, you can add new tags. You can also remove and rename tags. Tags for simulation objects are stored in `./data/simulationModelSets/<sms>/gui/tags.csv`. Tags for systems are stored in `./data/simulationModelSets/<sms>/gui/system-tags.csv`.

i The procedures in this section also apply to editing the Country list. Select Country on the Edit menu instead of Tags.

2.11.1 Adding a New Tag

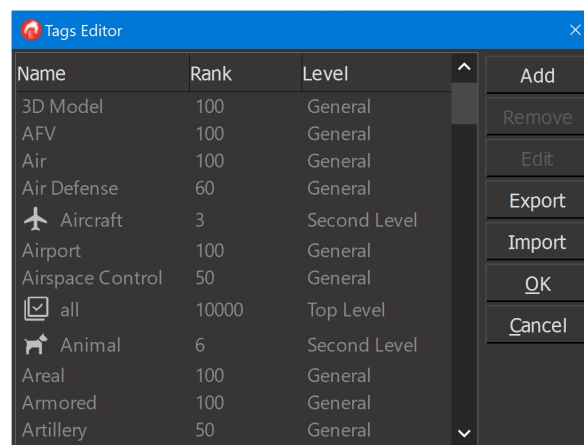
When you add a new tag, you can give it a rank. Ranks determine how tags display in the tag window:

- If a tag is a top-level tag, the rank determines the order in which it is listed relative to other top-level tags. The default top-level tags included with VR-Forces are Clutter (🗶), Entity (🚗), Simulation Effect Object (💣), Tactical Graphic (📊), Terrain (🏔️), Weather (☁️), Uncategorized, and All.
- If a tag is not a top-level tag, the rank determines the preference for including the tag in the tag list. The width of the Create Object Palette determines the number of tags that display, and then there are further rules determining which tags display. If a top level tag has more possible sub-tags than will fit in the available space, VR-Forces sorts the tags by rank order starting with 1. Within each group of tags with the same rank, it sorts them based on the number of simulation objects using that tag. Once it determines the tags to display, it lists them in alphabetical order. If you adjust the width of the Create Object Palette, the list updates dynamically.

► To add a new tag:

1. Choose **Model > Tags**. The Tags Editor opens (Figure 9).

Figure 9. Tags Editor



2. Click Add. The Add Tag Information dialog box opens.

3. Enter a name for the new tag.
4. Optionally, specify a rank.
5. From the Top Level Tag list, select the level where you want the tag to display:
 - Top Level. Displays with other top-level tags such as Entity, Tactical Graphic, and so on, and typically uses an icon.
 - Second Level. Displays with other second-level tags for the appropriate top-level tag and as sorted. Second-level tags are those found immediately below the top-level tags and typically use icons, for example, the Ground Vehicles, Aircraft, and other tags for entities.
 - General. Displays only as text below the second-level tags.
6. To make the tag available for people to select in the GUI, select the Visible check box. If this check box is not selected, the tag is not shown in the GUI but you can still use the tag for organization and also look for it using the Search box.
7. Optionally, select the Organize On List check box to indicate that the tag should be used as a group in the VR-Forces Create Object Palette and the SOE Object list.
8. To use an icon instead of the tag name, click the Add Icon button (+). The Select Tag Icon dialog box opens.
9. Select an icon file.
10. Click Open.
11. Click OK.



- If you add a tag and do not assign a simulation object to it, it does not display in the Create Object Palette even if there is space for it.
- In the VR-Forces Create Object Palette or the SOE Object list, if two top-level tags have the same rank and you select a tag, one of the top-level tags with the duplicated rank is hidden.

2.11.2 Removing a Tag

If you remove a tag, it is removed from any entity that has it assigned. You must save the SMS to make the change permanent.

► **To remove a tag:**

1. Choose **Model > Tags**. The Tags Editor opens.
2. Select the tag that you want to remove.
3. Click Remove.
4. Click OK.

2.11.3 Renaming a Tag

When you rename a tag, all simulation objects that are assigned to the old name are reassigned to the new name. The old name remains in the Tags list until you save the

simulation model set, but it is not listed in the Tags Editor. If you assign a simulation object to the old tag name before saving the SMS, the old name is retained and added to the list in the Tags Editor.

► **To rename a tag:**

1. Choose **Model > Tags**. The Tags Editor opens.
2. Select the tag that you want to rename.
3. Click Edit. The Edit Tag Information dialog box opens.
4. Enter the new name.
5. Click OK. The tags list updates.
6. Click OK.

2.11.4 Undoing Tag Changes

You can undo changes made to the tag list and entity tag assignments.

 When you undo tag changes, all changes to the tags list and all changes to entity tag assignments made since the last save of the simulation model set are reversed. Be sure that this is what you want before completing this procedure.

► **To undo tag changes:**

1. Choose **Model > Revert Tags**. A confirmation prompt is displayed.
2. Click Yes.

2.11.5 Using Auto Tags

You can automatically assign a set of tags to new models, to specific models, and to all objects. The tags are assigned by object type based on the information in *./appData/settings/simulationObjectEditor/autoTagInformation.csv*. (The Simulation Object Editor reads the file when it starts up, so if you edit the file while the SOE is open, you must restart the SOE to pick up the changes.)

When you create a completely new model (not a new model from an existing model), if the object type has an entry in *simulationObjectEditor.csv*, the associated tags are automatically added to the model. (When you create a new model from an existing model, the new model has the same tag list as the model it was created from.)

► To assign auto tags to the entire object list in an SMS, choose **Model > Apply Auto Tags**.

► **To apply auto tags to a specific model:**

1. Select the model whose tags you want to update.
2. Click the Apply Auto Tags button () above the Tags window.

2.11.6 Assigning Tags Based on Visual Model Characteristics

You can automatically assign tags to simulation objects that have 3D models based on their visual characteristics, such as articulated parts and display states. This process examines every visual model and can take a long time. You can use the filter and check

boxes to limit the list to only the models that you want to process.

► **To assign tags based on visual model characteristics:**

1. Choose **Model > Assign Tags Based on Visuals**. A message informs you of what the process entails.
2. Click Yes. The Select Visual Models dialog box opens. It lists all of the simulation objects in the SMS. The list does not include models that are inherited from an included SMS.
3. Optionally, use the Filter to include only the objects you want to process.
4. Optionally, use the check boxes, Select All, or Clear All to narrow down the list.
5. When the desired objects are selected in the list, click OK.

The SOE displays a message as it processes the simulation objects and assigns tags. The message closes when the process completes.

2.11.7 Exporting Tags

You can export the tags in an SMS. They are saved to a comma-separated values (CSV) file. You can then import the tag list into another SMS or another VR-Forces installation.

The tag list lists of tags, priorities, and icons associated with the tags, if any. It does not include any information about simulation objects.

► **To export the tag list:**

1. Choose **Model > Tags**. The Tags Editor opens.
2. Click Export. The Export Tag Information dialog box opens.
3. Specify a name for the tag file.
4. Click Save.

2.11.8 Importing Tags

► **To import a tag list:**

1. Choose **Model > Tags**. The Tags Editor opens.
2. Click Import. The Import Tag Information dialog box opens.
3. Select the tag file you want to import.
4. Click Open.

2.11.9 Managing Tags Using the Object Information File

If you want to edit the tags for many simulation objects, including adding new tags, editing individual simulation objects one-by-one can be tedious. You can speed up the process by editing them in a spreadsheet. To do this:

1. Export the simulation object list to a CSV file, as described in "2.10.1 Exporting Object Information" on page 25.
2. Make the changes you want to tag assignments.
3. Import the edited object information file into the SMS.

2.12. Managing the Forces List

When you create an object in the GUI, you must choose a force. You manage the list of forces available and their coloring in the Simulation Object Editor.

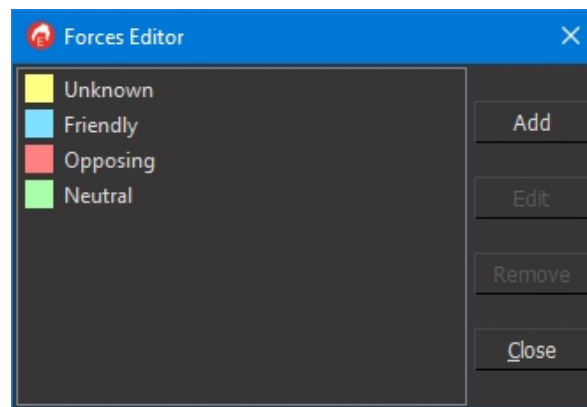
2.12.1 Adding a Force

Adding new forces may be useful if you are simulating participants from multiple countries or if you want to easily model and change force relationships for non-governmental forces such as militias or insurgent groups.

► **To add a force:**

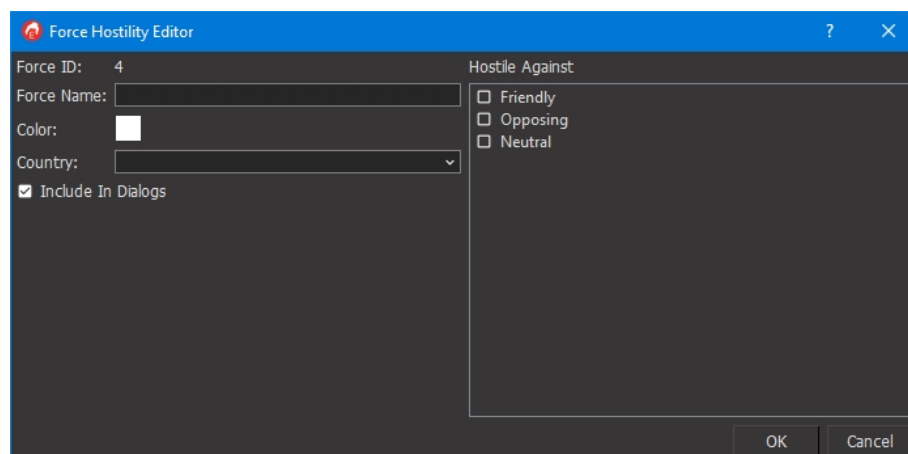
1. Choose **Edit > Forces**. The Forces Editor opens (Figure 10).

Figure 10. Forces Editor



2. Click Add. The Force Hostility Editor opens (Figure 11). The new force is assigned a force ID. You cannot edit the force ID.

Figure 11. Force Hostility Editor



3. In the Force Name box, type a name for the force.
4. Click the Color button. A color picker opens.
5. Select a color for the force.

6. Optionally, in the Country list, specify a country to associate with the force.
7. Optionally, in the Hostile Against pane, select the forces that the new force will be hostile towards. (You can change hostility at run time in the Hostility Matrix. For details, see "Managing Force Hostility Relationships".)
8. If you want the force to be listed in dialog boxes that list forces, select the Include in Dialogs check box.
9. Click OK.
10. Click Close.

2.12.2 Editing a Force

You can edit a force's name, color, hostility, and use. Each force has a force number. When you edit a force, you are changing the attributes of that numbered force.

► **To edit a force:**

1. Choose **Edit > Forces**. The Forces Editor opens (Figure 10 on the previous page).
2. Select the force that you want to edit.
3. Click Edit. The Force Hostility Editor opens (Figure 11 on the previous page).
4. To edit the name, change the text in the Force Name box.
5. To edit the color associated with the force, click the Color button and choose a color from the color picker.
6. To edit hostility relationships, in the Hostile Against window, select the forces that the force should be hostile towards.
7. To edit whether the force is listed in dialog boxes that list forces, select or clear the Include in Dialogs check box.
8. Click OK.
9. Click Close.

2.12.3 Removing a Force

If you create a scenario that has simulation objects with a particular force and you then remove the force, the next time you open the scenario, VR-Forces still maintains that force assignment for the simulation objects. The force will be listed in the Hostility Matrix. However, entity icons will not be colored and you will not be able to create any simulation objects with the removed force.

► **To remove a force:**

1. Choose **Edit > Forces**. The Forces Editor opens (Figure 10 on the previous page).
2. Select the force that you want to remove.
3. Click Remove. The force is removed from the list.
4. Click Close.

2.13. Copying the Object List to the Clipboard

You can copy the object list to the clipboard and paste it as text into word processing applications.

- To copy the object list to the clipboard, choose **Edit > Copy Names in List to Clipboard**.



3. Editing Common Parameters

This chapter explains how to use the Simulation Object Editor to edit simulation object parameters that are common to most simulation object types for both entity-level scenarios and aggregate-level scenarios. For details about editing parameters that are specific to individual entities, see "Chapter 4. Editing Entities for Entity-Level Scenarios" on page 73. For details about editing units, see "Chapter 5. Editing Units for Entity-Level Scenarios" on page 103 and "Chapter 6. Editing Aggregate-Level Objects" on page 115.

3.1. Editing Generic Object Parameters	35
3.1.1 Selecting an Object to Edit	35
3.1.2 Editing an Object's Tags	35
3.1.3 Changing an Object's Object Type Enumeration	36
3.1.4 Changing an Object's Name	38
3.1.5 Editing an Object's Description	39
3.1.6 Specifying Whether an Object Can Be Created	39
3.1.7 Specifying a Default User-Created Overlay	39
3.1.8 Specifying the Platform	40
3.1.9 Editing a Simulation Object's Country List	40
3.1.10 Creating Icon Lists	40
3.1.11 Undoing (Reverting) Changes to an Object	43
3.1.12 Reverting an Inherited Object's Parameters	43
3.2. Editing a Simulation Object's Visual Model	44
3.2.1 Changing the 3D Model or Colorized Model	45
3.2.2 Changing the 2D Icon Using a Font	46
3.2.3 Changing the 2D Icon Using an Image	47
3.2.4 Editing a Spot Report Model	48
3.2.5 Adding a New Visual Model or Image	48
3.2.6 Displaying Details about the Visual Model	50
3.2.7 Using Render Settings	52
3.2.8 Changing a Visual Model's Type	54
3.2.9 Specifying Visual Markings	54

3.2.10 Importing Visual Models	56
3.2.11 Pruning Visual Models	57
3.2.12 Exporting Visual Models	57
3.2.13 Reloading Visual Models in the VR-Forces GUI	57
3.3. Creating a New Simulation Object	57
3.3.1 Creating a New Model by Copying an Existing Model	58
3.3.2 Inheriting a New Model from an Existing Model	59
3.4. Deleting an Object Model	60
3.5. Creating a Screen Capture for the Create Object Palette	60
3.6. Creating a Simulation Object Group	60
3.6.1 Setting the Simulation Connection when Creating an SOG	61
3.7. Editing Tactical Graphics	61
3.7.1 Editing a Tactical Graphic's Menu Icon	61
3.7.2 Setting the Default Values for Tactical Graphics Properties	62
3.7.3 Specifying Tactical Graphic Properties that Can Be Edited in the GUI	62
3.7.4 Editing the Visual Model for a Tactical Graphic	63
3.8. Creating a Template	66
3.9. Editing Systems in the Simulation Object Editor	66
3.9.1 Creating a New System	67
3.9.2 Editing Probability Tables	68
3.9.3 Deleting a System	71
3.9.4 Reverting a System	71

3.1. Editing Generic Object Parameters

This section explains how to edit the parameters that are common to most objects. The procedures in this section apply to simulation objects, tactical graphics, cultural features, and other objects such as weather and contamination areas.



- By default, the SMSs provided with VR-Forces are read-only. It is strongly recommended that you not edit these SMSs. You should create copies or new SMSs that include them. For details, see "2.7. Creating New Simulation Model Sets" on page 22.
- If one of the object parameters described in this chapter does not apply to a particular object type, it is either blank or uneditable in the Simulation Object Editor.

3.1.1 Selecting an Object to Edit

Most of the Simulation Object Editor procedures assume that you have selected an object in the object list to edit.

► **To select an object to edit:**

1. Optionally, in the object list, select the tags for the type of object you want to edit.
2. Select the object in the list.

3.1.2 Editing an Object's Tags

In VR-Forces, objects are organized by tags on the Create menu and Create Object Palette. You can change the tags (or countries) assigned to an object. For details about adding new tags to the list, see "2.11.1 Adding a New Tag" on page 26.

► **To assign a tag to an object:**

1. Select the object whose tags you want to edit.
2. Select the General tab (Figure 12).

Figure 12. General tab (entity and aggregate)

3. Click the Add/Remove button (+) above the Tags window. The Select Tags dialog box opens.
4. To select one additional tag and close the dialog box, click the tag.

To add multiple tags, **Ctrl**+click the names of ones you want to add. You can finish and close the dialog box by either clicking the last tag or the Close button.

3.1.3 Changing an Object's Object Type Enumeration

The Simulation Object Editor lets you change an object's object type (DIS/RPR enumeration) and matching type. (For details about published and matching object types, see "Published Object Types and Matching Object Types" on page 168.) You might want to or have to do this for any of these reasons:

- If you create a new model, you must specify an entity enumeration (Object Type). (The Simulation Object Editor automatically constructs a Matching Type whose Domain and Kind field match the Object Type and whose other fields are wild cards.)
- If you create a new model from an existing model, the Simulation Object Editor automatically increments the existing model's enumeration for use with the new model. However, you might want to use a different enumeration than the one the Simulation Object Editor generates. (The Simulation Object Editor automatically creates a Matching Type that is the same as the Object Type.)

- You might want to change an existing object's enumeration.
- You might want to change an object's matching object type.

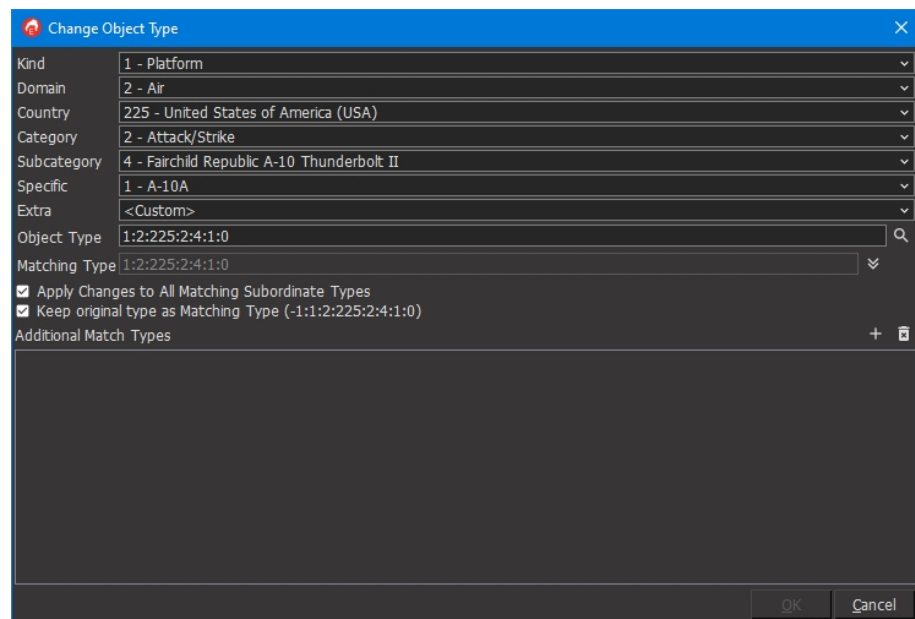


- The Entity Enumeration value displayed in the Simulation Object Editor is the 7-digit DIS or RPR FOM enumeration that VR-Forces uses to build the published object type. The matching object type is not displayed in the Simulation Object Editor, but you can specify it in the Change Enumeration dialog box as described in this procedure.
- You cannot use wildcards in an object type. They are permitted in the matching object type.

► **To change an object's enumeration:**

1. Select the object in the object list.
2. Select the General tab (Figure 12 on the previous page).
3. Click Edit next to the Object Type text box. The Change Object Type dialog box opens (Figure 13).

Figure 13. Change Object Type dialog box



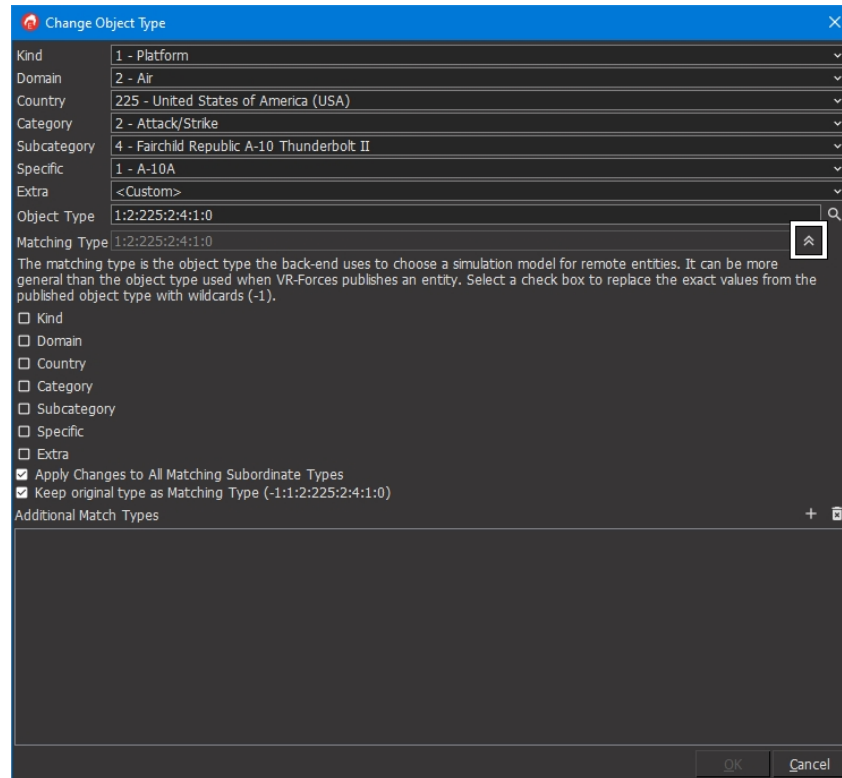
4. Edit the enumeration by selecting options from the lists for enumeration elements or enter a new enumeration in the Object Type text box.

You can click the Search button (🔍) to open a dialog and search for the entity type by description.

5. Optionally, specify the Matching Type:

- a. Click the double down arrow next to the Matching Type box. The dialog box expands to show the Advanced properties (Figure 14).

Figure 14. Change Object Type dialog box, Advanced



- b. For each field in the matching type enumeration that you want to be a wild card (-1), select the appropriate check box.
6. Optionally, clear the Apply Changes to All Matching Subordinate Types check box. Usually you will want an enumeration change to affect all cases where this simulation object will be created. However, if there is some reason that you want an enumeration change to not affect simulation objects created as part of a unit, you can do that by clearing this check box.

A more straightforward approach would be to create a new simulation object from the existing simulation object, rather than changing the enumeration.

7. Click OK. The enumeration changes.

3.1.4 Changing an Object's Name

The object name is the name listed on the object creation palettes and Create menu. The text placed on the screen next to a 2D icon is the marking text. The marking text is based on the Short Name, which is configured on the General tab.

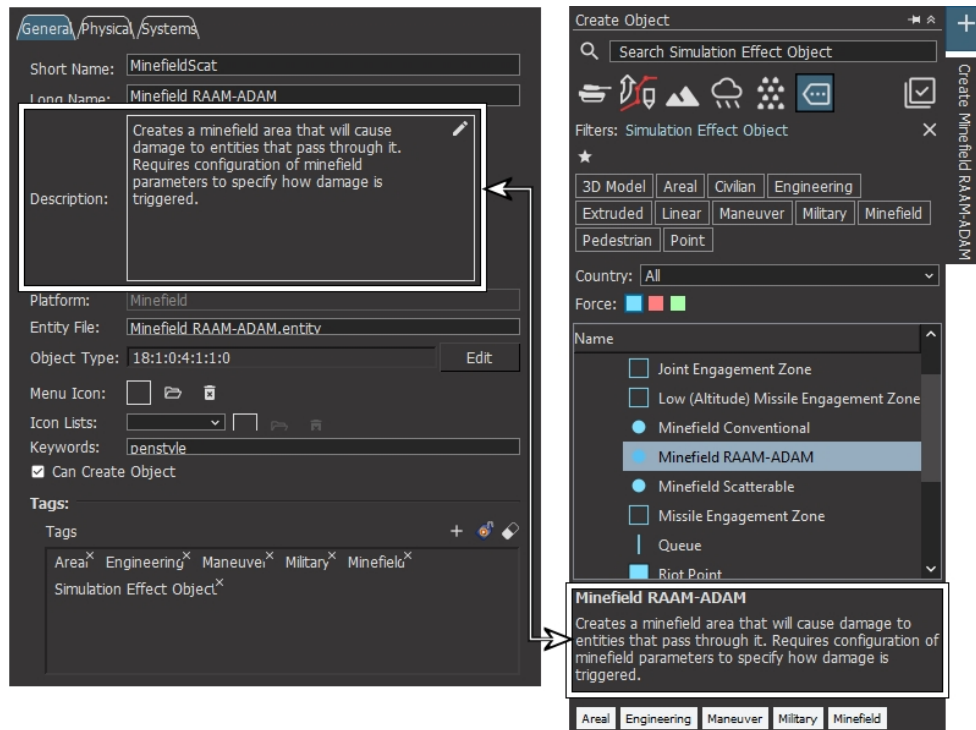
► **To change an object's name:**

1. Select the object in the object list.
2. Select the General tab (Figure 12 on page 36).
3. Enter a new name in the Long Name text box.

3.1.5 Editing an Object's Description

There is an optional field for the object's description. Descriptions display in the Create Object Palette in VR-Forces (Figure 15 below). It is useful for explaining objects whose purpose might not be straightforward before creating them.

Figure 15. Simulation object description



3.1.6 Specifying Whether an Object Can Be Created

You can control whether users can create a simulation object from the Create Object Palette. It may not make sense to be able to create some types of simulation objects, such as missiles, which are normally added to a scenario and removed dynamically.

► **To enable or disable creation of a simulation object from the Create Object Palette:**

1. Select the object in the object list.
2. Select the General tab.
3. Select or clear the Can Create Object check box (Figure 12 on page 36).

3.1.7 Specifying a Default User-Created Overlay

When an object is created, it is placed on an overlay. By default, the object is placed on an overlay according to its type (aircraft, ground vehicle, human, tactical graphics, and so on). You can assign an additional user-created default overlay to objects. You can still assign the object to a different overlay when it is created. For details about overlays and assigning objects to them, see "Creating and Editing Overlays" and "Specifying an Object's Properties Before You Create It (Click to Locate)".

► **To specify a default overlay:**

1. Select the object in the objects list.
2. Select the General tab (Figure 12 on page 36).
3. In the Keywords box, enter **overlay_***name* where *name* is the custom overlay.

3.1.8 Specifying the Platform

You cannot change the platform for existing simulation objects or new objects created from existing models. You can only specify the platform for new models that are not created from existing models. For details, see "3.3. Creating a New Simulation Object" on page 57.

3.1.9 Editing a Simulation Object's Country List

The country code in a simulation object's object enumeration specifies the country in which the object is manufactured. It does not indicate where it is used, and many object types are used in multiple countries. When creating a scenario, it can be convenient if the Create Object Palette only lists the objects used by the country for which you are laying down forces. The Country list in the Simulation Object Editor allows you to specify the countries that use a vehicle. You can then filter the Create Object Palette by country.

You can also apply a country to humans and cultural features if they are specific to a country.

The Country list works exactly like the Tags list. For details about how to configure the list of countries, see "2.11. Editing the Tags List" on page 26.

► **To assign a simulation object to a country of use:**

1. Select the object in the object list.
2. Select the General tab (Figure 12 on page 36).
3. Click the Add button (+) above the Countries pane.
4. The Select Countries dialog box opens.
5. To select one country and close the dialog box, click the country name. To add multiple countries, **Ctrl**+click the names of the countries you want to add.

Clearing the Countries List

If you add a new model by copying another model, you may want to quickly clear the Countries list before selecting the countries for the new model. (You can clear the other lists the same way.)

- To clear the Countries assigned to an object, on the General tab, click the Clear button (🗑) above the Countries window.

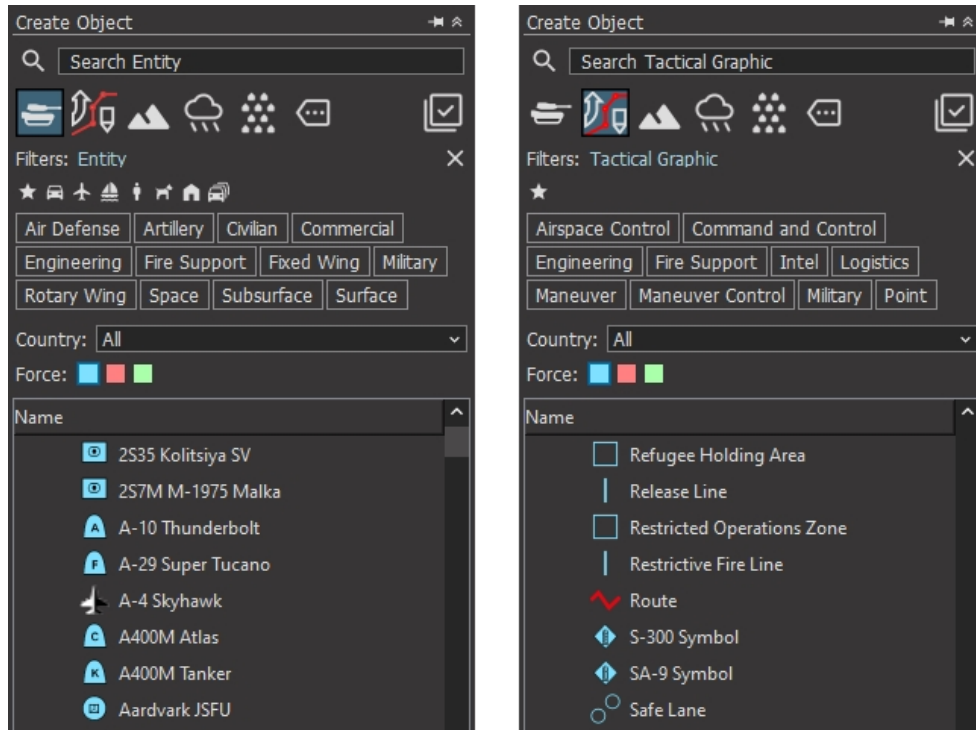
3.1.10 Creating Icon Lists

By default, the object lists on the object creation palettes and the Objects List Panel show the 2D icon for each object. For entities, units, and some tactical graphics, this is the MIL-STD 2525B icon. For other tactical graphics, it is a simple graphic representation of the object, such as a line for linear graphics or a box for areal graphics. However, if you want

to use images that are more specific to simulation object types, you can create custom icon lists that replace the default images. For example, instead of a straight line for a route, you might want to use a segmented line. Or instead of a 2D icon for an entity, you might want an image of the entity or a generic platform image.

Figure 16 illustrates object palettes with custom images (A-4 Skyhawk and Route).

Figure 16. Alternative icons



When you create a custom icon list, you create a list and then specify an image for each object that you want to change. If you do not specify an image for an object, it uses the default icon. You can create as many icon lists as you want. You specify the icon list to use on the Application Settings dialog box, GUI Settings page.

► **To create an icon list:**

1. Choose **Edit > Icon Lists**. The Icon Lists Editor opens.
2. Click Add. The Add List dialog box opens.
3. Enter a name for the new list.
4. Click OK.
5. In the Icon Lists Editor, click OK.

► **To change the name of an icon list:**

1. Choose **Edit > Icon Lists**. The Icon Lists Editor opens.
2. Select the icon list whose name you want to change.
3. Click Edit. The Edit List dialog box opens.
4. Enter a new name for the list.

5. Click OK.
6. In the Icon Lists Editor, click OK.

► **To remove an icon list:**

1. Choose **Edit > Icon Lists**. The Icon Lists Editor opens.
2. Select the icon list that you want to remove.
3. Click Remove.
4. Click OK.

Specifying Icons for an Icon List

After you create an icon list, you specify the images for each simulation object that you want to change the icon for.

► **To specify an image for an icon list:**

1. Select a simulation object in the objects list.
2. Select the General tab.
3. In the Icon Lists drop-down list, select the icon list that you want this object to use.
4. Click the Browse button (). The Select Menu Icon dialog box opens.
5. Select an image. It can be a PNG, SVG, JPG, or BMP file.
6. Click Open. The image displays next to the Browse button.
7. Repeat this procedure for each simulation object for which you want to specify an image.
8. Save the Simulation Model Set.

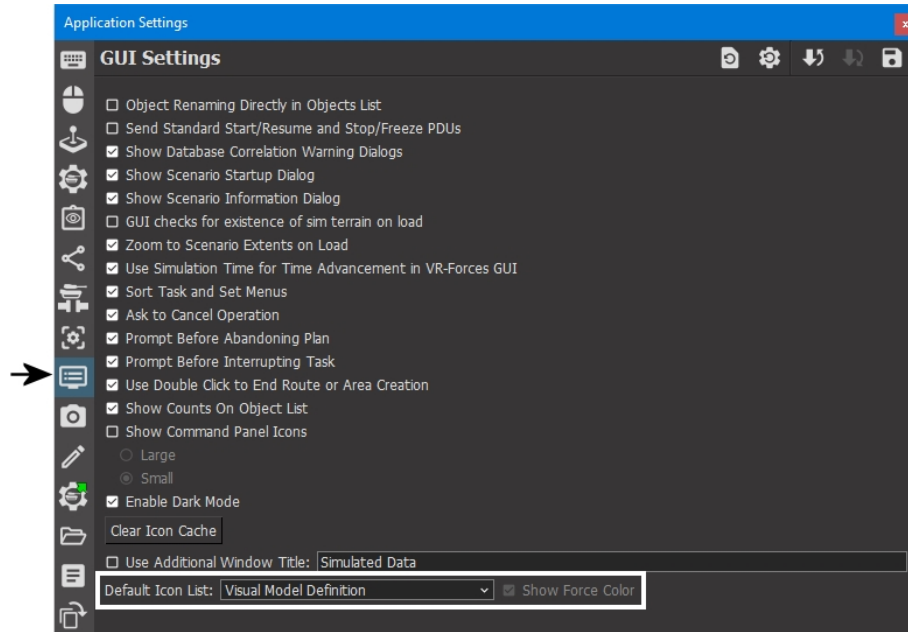
Using an Alternative Icon List

If you create an icon list, you can specify that VR-Forces use it instead of the default icon list.

► **To specify use of an alternative icon list:**

1. In VR-Forces, choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the GUI Settings page (Figure 17).

Figure 17. GUI Settings page



3. In the Default Icon List list, select the icon list that you want to use.
4. Optionally, select Show Force Color. This places a colored square behind the image to show its force.

3.1.11 Undoing (Reverting) Changes to an Object

If you have made changes to an object and have not yet saved the simulation model set, you can revert to the original simulation object configuration. You can revert an object at any time prior to saving the SMS, regardless of how many other objects you may have changed since you edited the simulation object you want to revert.

► **To undo changes to an object:**

1. Select the object in the object list.
2. Choose **Model > Revert Model**.

3.1.12 Reverting an Inherited Object's Parameters

You can create a new simulation object from another by inheritance, where you create a copy of the original (parent). See "3.3.2 Inheriting a New Model from an Existing Model" on page 59. You can then make edits to those parameters. At any time, you can revert the parameters to the parent's parameters.

Please note, reverting a parameter resets it to the values that existed for the parent at the time you created the child. If the parameters changed on the parent after you created the child, reverting does not set the child parameters to the parent's new values.

► To revert an inherited object's parameter to its parent's parameter:

1. Select the object in the model list.
2. Next to the parameter that you want to revert, click the Revert to Parent button (↶).

3.2. Editing a Simulation Object's Visual Model

The Visual Model Editors dialog box allows you to configure all aspects of entity visualization, including the 2D icon and 3D model used. (For details, see the chapters in [Section III, Visual Models](#), in *Adding Content to MAK ONE Applications Reference Manual*.) However, the Simulation Object Editor allows you to quickly and easily change the 3D model or 2D icon used to represent a simulation object. When you change a model in the Simulation Object Editor, the entity's model definition also changes.

You edit visual model parameters on the Visual Models Panel (Figure 18).

Figure 18. Visual Models Panel



The Visual Models Panel is a fully functional window that supports the same mouse-look and keyboard controls as the VR-Forces main window. If you change the observer view of the selected entity's visual model, the window maintains that offset as you select other entities. Press **Spacebar** to reset the view.

The panel has these buttons:

-  Edit Visual Model. Automatically opens the correct editor based on the model set that is displayed.
-  Edit Wake Model. Changes the wake settings for surface entities. (Hidden for other platforms.)
-  Visual Effects. Lets you display how the model looks when trailing effects, shadows, and wakes are enabled.
-  Switch Node States. Lets you display how the model looks when switch node states are set.
-  Environment Settings. Lets you display what the model looks like at different times of day and with different wind and ocean states.

► **To change the visual model for a simulation object:**

1. In the object list, select the object whose model you want to change. The model for the currently selected model set is displayed in the Visual Model Panel.

 The visual model window responds to the same keyboard and mouse controls as the GUI.

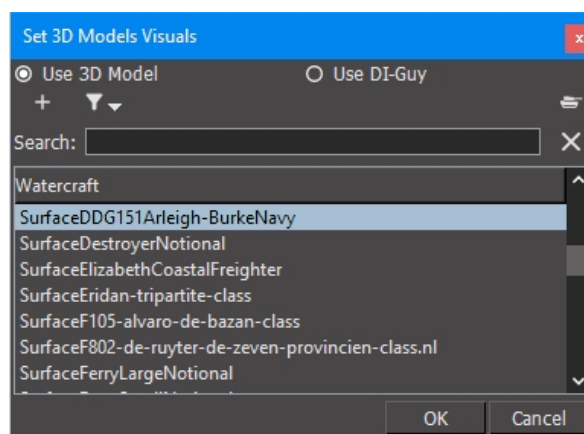
2. In the Model Set list in the Visual Model group box, select the model set that you want to edit for this simulation object.
3. Change the 3D model, 2D icon, or 2D image as described in the following sections.

3.2.1 Changing the 3D Model or Colorized Model

► **To change a 3D model or 3D colorized model:**

1. In the Visual Model panel, click the Edit button (). The Set 3D Models Visuals dialog box opens (Figure 19).
2. If the dialog box has options for 3D Models or DI-Guy, select Use 3D Model. The Modify Visuals dialog box lists available models (Figure 19). (You can add new models. For details, see "3.2.5 Adding a New Visual Model or Image" on page 48.)

Figure 19. Modify Visuals dialog box (3D)



3. Select the model that you want to use. The new model displays in the panel.

i By default, the Modify Visuals dialog box tries to filter the list to show only the type of entity that you are editing. If you do not see the models you expect to see in the list, particularly if you know that the models are available in the `./SharedData/##/latest` directory, try changing the filter. For example, surface entities are of type Watercraft, while subsurface entities are of type Entity.

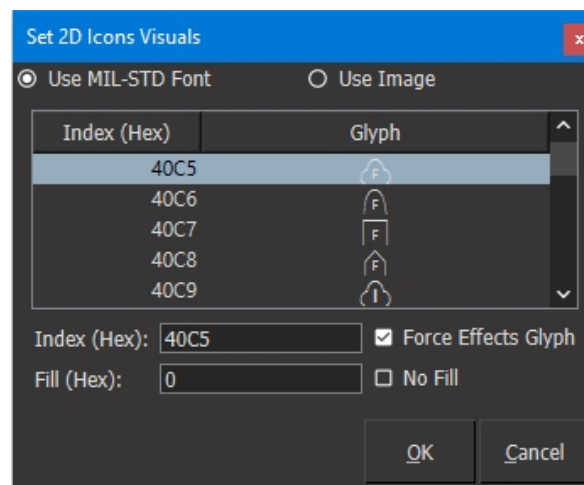
4. Click OK to close the Modify Visuals dialog box.

3.2.2 Changing the 2D Icon Using a Font

► To change a 2D icon using the MIL-STD font:

1. Select the simulation object whose icon you want to edit.
2. In the Visual Models Panel, in the Model Sets list, select 2D Icons.
3. Click the Edit button (✎). The Set 2D Icons Visuals dialog box opens (Figure 20).

Figure 20. Modify Visuals dialog box (2D)



4. Select the Use MIL-STD Font option. The dialog box lists index values and associated glyphs.
5. Select the icon glyph that you want to use. The new icon is displayed in the panel.
6. Optionally, change the parameters.
7. Click OK to close the dialog box.

Regenerating Spot Report Definitions

When you change the 2D icon for a simulation object, the spot report icon for that object does not change. You can change the icon manually, as described in "3.2.4 Editing a Spot Report Model" on page 48, or you can automatically update spot report definitions.

► **To regenerate spot report definitions:**

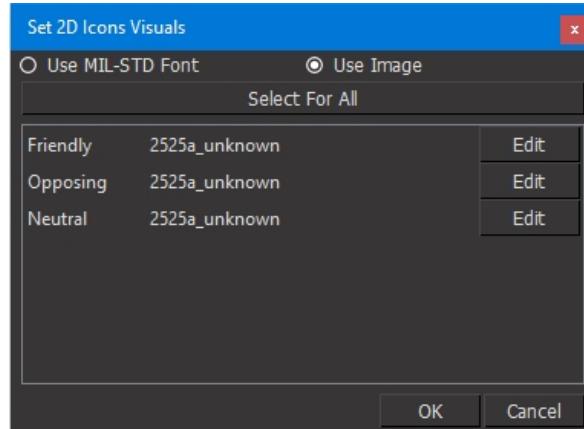
1. Choose **File > Regenerate Spot Report Definitions**. A prompt opens.
2. Click Yes.

3.2.3 Changing the 2D Icon Using an Image

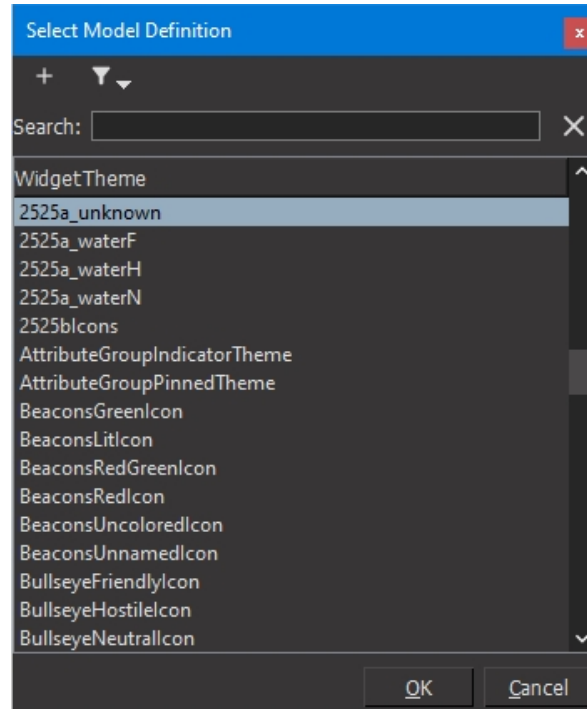
► **To change a 2D icon using an image:**

1. Select the simulation object whose icon you want to edit.
2. In the Model Sets list, select 2D Icons.
3. Click the Edit button (✎). The Set 2D Icons Visuals dialog box opens (Figure 20 on the previous page).
4. Select the Use Image option. The dialog box lists the forces that are configured for this simulation model set (Figure 21).

Figure 21. Modify Visuals dialog box (2D image)



5. Click the Edit button for the force whose icon you want to change or, to choose the same image for all forces, click Select for All. The Select Model Definition dialog box opens (Figure 22). It lists the available images. By default, the list is for MIL-STD 2525a icons. (You can add new images. For details, see "3.2.5 Adding a New Visual Model or Image" on the next page.)

Figure 22. Select Model Definition dialog box

6. Select the image that you want to use. The new image is displayed in the panel.
7. Click OK.
8. Click OK to close the Modify Visuals dialog box.

3.2.4 Editing a Spot Report Model

You can edit the model that is used when a spot report is at the full knowledge level. The default configurations assign 2D icons to spot reports. In theory, you could assign some other type of model to a spot report. (The icons used at the detected, classified, and identified levels are not editable.)

► **To edit a spot report icon:**

1. Select the simulation object whose icon you want to edit.
2. In the Visual Models Panel, in the Model Sets list, select one of the spot report options.
3. Click the Edit button (✎). A dialog box appropriate to the model type opens.
4. Change the model, as described in "3.2.1 Changing the 3D Model or Colorized Model" on page 45, "3.2.2 Changing the 2D Icon Using a Font" on page 46, or "3.2.3 Changing the 2D Icon Using an Image" on the previous page.

3.2.5 Adding a New Visual Model or Image

When you change a model or image in the Simulation Object Editor, you are presented with a list of the currently configured model definitions or images. You can add additional

models to the list. You can add new model definitions in the context of editing a simulation object or independently of editing a simulation object. If you edit the list in the context of editing a simulation object, you can also add images to the list.

Adding A Model Definition while Editing a Simulation Object

► **To add a new model or image to the visual models list while editing a simulation object:**

1. In the object list, select the object whose model you want to change. The model for the currently selected model set is displayed in the Visual Model group box.
2. In the Visual Model group box, in the Model Sets list, select the model set that you want to edit.
3. For 3D models or 3D colorized models:
 - a. Click the Edit button (). The Set 3D Models Visuals dialog box opens (Figure 19 on page 45).
 - b. Click the Add button (). The Model Definition dialog box opens. This is a standard file chooser dialog box.

For 2D icons:

- a. Click the Edit button (). The Set 2D Icons Visuals dialog box opens (Figure 20 on page 46).
- b. Select the Use Image option. The dialog box lists the forces that are configured for this simulation model set.
- c. Click the Edit button for the force to which you want to add an image. The Select Model Definition dialog box opens (Figure 22 on the previous page).
4. Select the file for the model that you want to add to the list. It must be a model or image type that VR-Forces supports. (This file becomes the value for the filename attribute of the model definition.)
5. Click OK. The Model Definition Name dialog box opens.
6. Enter a name for the model definition.
7. Click OK.
8. Click OK to close the Modify Visuals dialog box.

Adding Model Definitions Independently of Editing a Simulation Object

You can add model definitions in the Simulation Object Editor independently of any simulation object just as you can in the Visual Model Editors dialog box. The only attribute that you can set is the filename. However, in many cases that is the only attribute you need. If you need to specify other attributes, you must use the Visual Model Editors dialog box.

► **To add a model definition independently of editing a simulation object:**

1. Choose **Edit > Add Model Definition**. The Add Model Definition dialog box opens.
2. Click the Add button (). The Model Definition dialog box opens. This is a standard file chooser dialog box.

3. Select the file for the model that you want to add to the list. It must be a model or image type that VR-Forces supports. (This file becomes the value for the filename attribute.)
4. Click OK. The Model Definition Name dialog box opens.
5. Enter a name for the model definition.
6. Click OK. The new model definition is added to the list.
7. Click OK to close the Add Model Definition dialog box.

3.2.6 Displaying Details about the Visual Model

The Visual Models Panel displays the model used for the selected simulation object. You can change how the model is displayed and show information about it:

- Bounding volumes. Display the bounding volume.
- Environment settings. Show the model at different times of day, different view magnification, different wind speed, and different water quality.
- Markers and marker labels. Displays labels for the markers for sensor and weapon positions. Also, show markers as 2D only.
- Visual effects. Show effects such as shadows, trailing effects, smoke, and flames.
- Model sets. Specifies which model set to use to display the entity.
- Switch node states. Displays the different switch nodes for the model, if applicable. If a model has articulated parts, you can enable them individually. They move at five times the minimum rotation or translation speed.
- View. Perspective view or view the model from the specified side (left, front, top, and so on.)

Changing the View of the Model

The default view for 3D models is a perspective view. You can also view a model from the front, back, sides, or top.

- To change the view of a model, in the Visual Models Panel, select an option from the View list.

Changing the Model Set

You can select the model set to use to display a visual model. In addition to the three primary model sets, you choose a spot report model set.

- To change the model set used to display a visual model, in the Visual Models Panel, select an option from the Model Sets list.

Displaying a Visual Model's Bounding Volume

You can display the bounding volume of a 3D model. You can edit the dimensions of the bounding volume on the Physical tab.

- To display a model's bounding volume, choose **Preferences > Show Bounding Volumes**.

Displaying Switch Node States

If a visual model has switch node states, you can display them. Switch nodes can represent damage states and articulated parts.

► **To display switch node states:**

1. Choose **View > Switch Nodes States** or click the Switch Nodes States button (). The Switch Nodes States dialog box opens.
2. To see a damage state, select an option from the Damage list.
3. To see articulated parts, select the check box for the state you want to display.
4. Click Close.

Displaying Markers and Marker Labels

You can display markers that show the location of support points for a model. You can display the markers as 2D or 3D and can also display text labels that identify them. You can edit the location of support points on the Physical tab.

- To toggle markers on and off, choose **Preferences > Show Markers**.
- To toggle marker labels on and off, choose **Preferences > Show Marker Labels**.
- To toggle markers between 2D and 3D, choose **Preferences > Show Markers as 2D**.

Displaying the Effects of Environment Settings

You can display how changing the following environment settings affects a visual model:

- Time of day
- View magnification
- Wind speed
- Wind direction
- Choppiness
- Ocean quality

Time of day affects all models. The other settings only affect surface entities.

► **To display the effects of environment settings for a model:**

1. Choose **View > Environment Settings** or click the Environment Settings button (). The Environment Settings dialog box opens.
2. Change the settings as desired.
3. Click Close.

Displaying Visual Effects for a Model

You can display the following visual effects for models:

- Dust clouds
- Contrails
- Tracks

- Flames
- Smoke
- Footprints
- Wakes
- Shadows
- Speed

If an effect is not valid for the selected model, the check box is not available.

► **To display visual effects for a model:**

1. Choose **View > Visual Effects** or click the Visual Effects button (). The Visual Effects dialog box opens.
2. Select the check box for each visual effect that you want to display.
3. To change the apparent speed of a moving model, adjust the Speed slider.
4. Click Close.

3.2.7 Using Render Settings

You can use the render settings while working with a simulation object.

- To access the Render Settings, choose **System > Render Settings**. The Render Settings window opens.

The render settings are:

- Wireframe Rendering Mode. Displays the polygons that make up the simulation object.
- Performance Statistics Overlay. Performance Statistics show information about the simulation object.
- Developer Debug Overlay. The SOE uses the ImGui API to create an overlay that developers can use to test changes to some of its visual features. It includes an overlay for testing ocean settings, shadow settings, and other visual features. It also includes an ImGui test user interface that demonstrates some of its features.

This feature is provided "as is" for the use of interested developers. For complete information about how to use ImGui, please see documentation at <https://github.com/ocornut/imgui> and other internet resources.

- Channel Projection Resize Policy. The Projection Resize Policy determines how a channel behaves when you resize it. The options are:
 - Fixed. Keep the aspect ratio the same regardless of how the window is resized.
 - Horizontal. Change the horizontal field of view (FOV) so that the ratio of the FOV angles matches the aspect ratio of the viewport.
 - Vertical. Change the vertical FOV so that the ratio of the FOV angles matches the aspect ratio of the viewport.

You can change the projection resize policy for individual channels by editing the Projection Resize Policy attribute. You can also change the projection resize policy for all channels.

- Occluded Symbol Opacity.
- Crepuscular Rays. The SOE can simulate the effect of crepuscular rays (sunbeams) beaming through clouds.
- Depth of Field Autofocus.
- Droplets Scacle. You can change the size of water droplets for splash effects and sea spray.
- HBAO. Horizon-based ambient occlusion (HBAO) adds screen space ambient occlusion lighting effects to the scene. This improves the realism by approximating the amount of ambient light that impacts a scene, which provides more realistic lighting of three-dimensional objects. There are parameters that adjust how ambient occlusion appears:
 - HBAO Radius. The distance from occluding objects where the HBAO effect is visible. Higher values cause the effect to extend further from occluding objects.
 - HBAO Bias. The bias for thin items. At higher values, larger objects cause the most noticeable effect on surrounding objects. This can reduce polygon edge artifacts and artifacts caused by thin geometry.
 - HBAO Exponent. The intensity of the HBAO effect. Higher values create a more intense effect.
 - HBAO Scale Small AO. The intensity of the HBAO effect on objects that are close to each other. Higher values have a more intense effect on the small corner areas of objects and also objects that are close together.
 - HBAO Scale Large AO. The intensity of the HBAO effect on objects that are at the edges of the specified HBAO Radius. Higher values make the area for objects that are around the HBAO radius distance from each other show a more intense effect.
 - HBAO Depth Clamp Mode. The depth clamp mode to use. To Border uses a method that reduces the false occlusion of objects along the edge of the screen. To Edge uses a method that correctly applies the occlusion effect to objects along the edge of the screen.
- Occlusion Queries. If you are simulating on a terrain with many buildings, such as in a city, enabling occlusion queries can improve the frame rate. When they are enabled, the SOE draws draw bounding boxes around some objects and checks to see whether they are being rendered or are blocked visually. You can view the bounding boxes by enabling occlusion nodes and OC Draw Debug in the Developer Debug Overlay Terrain Patch Object Settings.
- Models Alpha to Coverage, Terrain Alpha to Coverage, and Alpha to Coverage Dither. Alpha to coverage is a way of handling objects with transparency that improves rendering of objects that have a sharp transition between opaque and

transparent. Grass, trees, and fences are a few examples of when alpha to coverage is a good option. In addition to improving visualization of some objects, there is a slight performance improvement when alpha to coverage is enabled.

If alpha to coverage is enabled, you can enable a dither shader that modifies the alpha with a noise pattern so that there are no hard edges between different alpha levels. The tradeoff for this setting is that it adds a small amount of noise as the object or camera moves.

- **Debug Visuals.** The SOE provides some shader debugging capabilities within its lighting shaders. The shaders are instrumented such that setting different flags (through uniforms) changes the behavior of the shader. Typically it changes fragment colors based on some parameter or parameters.

3.2.8 Changing a Visual Model's Type

The entity element definition hierarchy categorizes element definitions by type. These types associate a variety of parameters with the models in the category. Changing the visual type changes the 3D model for any simulation object that is using a generic model. If a simulation object has a specific model, changing the visual type does not affect it. However, it affects how the model is displayed in the SOE. Surface models are displayed with a background that lets you view wakes and other effects.

► **To change a visual model's type:**

1. In the object list, select the model whose type you want to change.
2. Choose **Model > Advanced Visual Model Editing**. The Advanced Visual Model Editing dialog box opens.
3. Select an option in the Visual Type list.

3.2.9 Specifying Visual Markings

VR-Forces uses a process called texture mapping to apply different surface textures to a simulation object's physical geometry. For example, you can use this technology to change the colors of a car, change the logo and design on aircraft, and change markings on a vehicle. The Simulation Object Editor allows you to see how different markings look on a vehicle, to apply specific markings to a vehicle, and to configure a vehicle to apply randomly assigned markings.

You can specify the visual markings that are applied to a simulation object when it is added to a scenario. Markings can be specific or random. A simulation object may have more than one type of visual marking. You can set each of them individually. The options are:

- Specific. Use one of the specific marking configured for the object.
- Random. Choose randomly from the markings available for the object. You can choose to be simply random or specify how to randomize the marking:
 - Weighted Random (20/80). Given a list of markings, there is an 80% chance that one of the markings in the top 20% of the list will be applied. There is a 20% chance that one of the other markings will be applied.
 - Weighted Random (Exponential). The markings towards the top of the list are more likely to be chosen than those later in the list.
- Variant. This is a possible pattern, such as country, for subvariant markings.
- Subvariant. Choose the manner in which the variant is displayed, for example, insignia.
- Unused. Do not show the option in VR-Forces.

For weighted random options, the order of the list is based on the order that markings are listed in the relevant metadata file. If you want the markings to have a different priority, you must edit the metadata file.

For example, if colors in the metadata file are listed in the order red, blue, green, yellow, brown and you select the weighted random approach, most cars will be red, because there is an 80% chance of selecting the color in the top 20% of the list. (In a list of five items, the first item in the list is the top 20%.) If you want most cars to be brown, you would need to edit the metadata file to put brown at the top. (For details about metadata files, please see "[The Metafile for Markings](#)" in the *Adding Content to MAK ONE Applications Reference Manual*.)

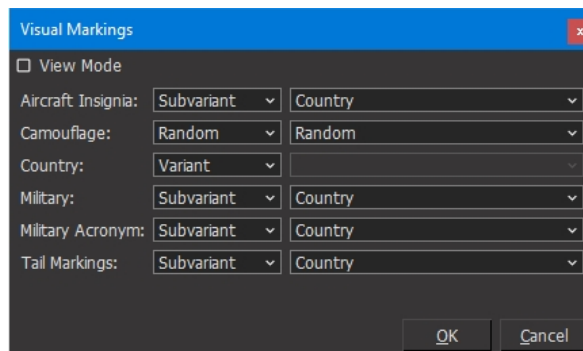
For details about how to change markings at run time, see "Markings". For details about how to configure texture maps and their associated files, please see "[Adding Dynamically Changeable Markings to Models](#)" in the *Adding Content to MAK ONE Applications Reference Manual*.

Viewing a Model's Visual Markings

The Simulation Object Editor lets you see how the available visual markings look on a simulation object.

► To view a model's visual markings options:

1. Select the simulation object.
2. In the Visual Model panel, click the Visual Markings button (). The Visual Markings dialog box opens (Figure 23). It has a list of options for each type of marking that the simulation object supports.

Figure 23. Visual Markings dialog box for the C-130 Hercules aircraft

3. Select specific options in the lists for each marking type that you want to view.
4. Click Cancel. (If you click OK, the options you have chosen are applied to the model.)

Setting Visual Markings

In the Simulation Object Editor, you can specify the visual markings that are applied to a simulation object when it is added to a scenario.

► **To set the visual markings for a simulation object:**

1. Select the simulation object.
2. In the Visual Model panel, click the Visual Markings button (🏷️). The Visual Markings dialog box opens (Figure 23).
3. For each type of marking:
 - a. If you want to use a specific marking, select it from the list.
 - b. If you want markings assigned randomly, select a random option in the list. For weighted random options, the markings at the top of your list are the most likely to be used.
4. Click OK.

3.2.10 Importing Visual Models

You can import visual models (element definitions, model definitions, and object type mappings) into an SMS. You might want to do this if you have edited an SMS and then decide that you need to revert to a previous version that you have saved in `./factory` or elsewhere. When you import visual models, the imported model definitions replace any matching definitions in the SMS. The import operation imports all leaf files in the selected directory.

► **To import visual models:**

1. Choose **File > Import Visual Models**. A confirmation prompt opens.
2. Click Yes. A file chooser dialog box opens.
3. Select the directory from which you want to import visual models.
4. Click OK.

3.2.11 Pruning Visual Models

Pruning visual models (element definitions, model definitions, and object type mappings) removes duplicate visual models from the set of leaf files that are loaded for an SMS. This improves the load time in the SOE and when you load a scenario.

Pruning only affects the SMS-specific definitions files. It does not touch any additional definition files that may have been saved from the Visual Model Editors dialog box.

► **To prune visual models:**

1. Choose **File > Prune Visual Models**. A confirmation prompt opens.
2. Click Yes. A file chooser dialog box opens.
3. Select the directory from which you want to prune visual models.
4. Click OK.

3.2.12 Exporting Visual Models

When you update an SMS, you can export the visual models (element definitions, model definitions, and object type mappings) to a directory other than the default directory in the SMS. This would typically be done for archival purposes, such as updating the files in *./factory*.

The SOE exports only the SMS-specific definitions files. If user-created leaf files were previously loaded, the definitions are not exported back to those leaf files. They are included in the SMS-specific files.

► **To export visual models:**

1. Choose **File > Export Visual Models**. A confirmation prompt opens.
2. Click Yes. A file chooser dialog box opens.
3. Select the directory to which you want to export visual models.
4. Click OK.

3.2.13 Reloading Visual Models in the VR-Forces GUI

If you update the visual models associated with simulation objects and you have a scenario open, you can refresh the models used in the scenario without shutting down VR-Forces.

- To refresh visual models, in the VR-Forces GUI choose **Settings > Reload Visuals**.

3.3. Creating a New Simulation Object

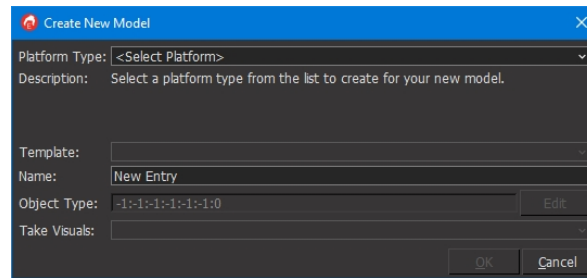
You can create a new simulation object by using an existing object as a template or by building a completely new object.

❗ The procedure for creating a new unit is largely the same as for creating a new entity. The main difference is that units do not have 3D models and, when you create a unit, you specify its subordinates.

► **To create a completely new object:**

1. Choose **Model > New Model**. The Create New Model dialog box opens (Figure 24).

Figure 24. Create New Entity dialog box



2. In the Platform Type list, select a platform type.
3. Optionally, select a template. (Templates are specific to a platform type. There might not be a template for some platforms.)
4. In the Name box, enter a name for the entity.
5. Optionally, for the Object Type, click Edit and specify the object type enumeration. You can do this after you create the entity if you want to. You can also just accept the object type supplied by VR-Forces.
6. In the Take Visuals list, select a specific model or Closest Match. If you select Closest Match, VR-Forces selects the entity model whose object type enumeration is closest to the one you specified.
7. Click OK. The Simulation Object Editor adds a new entry to the object list. The editor shows default values for all the fields.
8. Optionally, edit the enumeration.
9. Specify 2D and 3D visual models.
10. Specify appropriate values for the object parameters.
11. Enter a name to be used in the object's label (marking text) in the Short Name text box.
12. Specify a movement system.
13. Specify a damage model.
14. Optionally, add weapon systems.
15. Optionally, add sensors.
16. Save the simulation model set.

3.3.1 Creating a New Model by Copying an Existing Model

If you create a new simulation object model by copying an existing model, the new model starts with all of the behavioral characteristics and systems of the original one. It increments the object enumeration. You only need to name it and customize it for the new simulation object.

 You can use this procedure to create new units.

► **To create a new simulation object model as a copy of an existing model:**

1. In the object list, select the source object.
2. Choose **Model > New Model from Existing (Copy)**. The Create New Model dialog box opens.
3. Enter a name for the model.
4. Optionally, edit the enumeration.
5. Click OK. The new simulation object is added to the object list. If you did not edit the enumeration, the original enumeration is incremented. An *.entity* file is created based on the name.
6. Edit the parameters to provide the new simulation object the characteristics you want for it.
7. Save the simulation model set.

3.3.2 Inheriting a New Model from an Existing Model

If you create a new simulation object model by inheriting from an existing model, the new model (child) inherits its behavioral characteristics and systems from the original one. You can customize it as needed, but you can also revert it to the settings of the model from which it is inherited (parent).

You can use this procedure to create new units.

 If you want to undo any changes, you can revert them. See "3.1.12 Reverting an Inherited Object's Parameters" on page 43.

► **To create a new simulation object model by inheriting from an existing model:**

1. In the object list, select the source object.
2. Choose **Model > New Model from Existing (Inherit)**. The Create New Model dialog box opens.
3. Enter a unique name for the model.
4. Optionally, edit the enumeration.
5. Click OK. The new simulation object is added to the object list. If you did not edit the enumeration, the original enumeration is incremented. An *.entity* file is created based on the name.
6. Edit the parameters to provide the new simulation object with the characteristics you want for it.
7. Save the simulation model set.

3.4. Deleting an Object Model

► **To delete an object model from a simulation model set:**

1. Select the object in the object list. (You can select multiple models using the **Shift** or **Ctrl** keys.)
2. Choose **Model > Delete Model** or press **Delete**.

3.5. Creating a Screen Capture for the Create Object Palette

In VR-Forces, the Create Object Palette shows a screen capture of the selected simulation object at the bottom of the palette. You can capture a new image to replace the default for an existing object or add a new one for a new object.

► **To capture an image for use in the Create Object Palette:**

1. Change the observer view in the Visual Model Panel as desired.
2. Choose **Edit > Create Screen Grab**.

3.6. Creating a Simulation Object Group

You create simulation object groups (SOGs) in the Simulation Object Editor. You specify the members of the SOG in a version of VR-Forces that is launched from the Simulation Object Editor. When the Simulation Object Editor launches VR-Forces, it needs to use a simulation connection. For details about setting the simulation connection, see "3.6.1 Setting the Simulation Connection when Creating an SOG" on the facing page.

Simulation object groups can include scripted tasks and sets. They become scenario-specific scripts when you add the SOG to a scenario.

Simulation object groups are saved with the extension *.sog*.

► **To create a simulation object group:**

1. Start the Simulation Object Editor.
2. Load the SMS you want to use.
3. Choose **Model > New Object Group**. The New Object Group dialog box opens.
4. Enter a name for the new SOG.
5. Click OK. The new group is added to the objects list and the Simulation Object Editor displays a simulation object group page.
6. Specify the tag, creation palette option, and other generic information as you would for any simulation object.
7. Optionally, change Creation Options and the altitude offset.
8. Click Launch Object Group Editor. You are prompted to save your changes to the SMS.
9. Click Yes. A limited version of VR-Forces opens. It has a grid terrain.

10. Use the Create Object Palette to add objects to the SOG as you would for a scenario. You can change to Stealth observer mode if you want to view the terrain in 3D. It is a flat terrain.
11. Choose **File > Save Object Group** to save the SOG and continue editing. Choose **File > Save Object Group and Exit** to save the SOG and close VR-Forces.

3.6.1 Setting the Simulation Connection when Creating an SOG

When you create an SOG, the Simulation Object Editor starts VR-Forces where you specify the members of the SOG. By default, it uses the DIS localhost connection for *EntityLevel.sms* and the HLA 1516 Evolved connection for the aggregate-level SMSs. You can change the connection if you want. (Aggregate-level scenarios must use HLA 1516 Evolved.)

► **To set the simulation connection when specifying the members of a simulation object group:**

1. In the Simulation Object Editor, choose **Edit > Object Group Connection**. The Choose Connection dialog box opens.
2. Select the connection you want to use.
3. Click OK.

3.7. Editing Tactical Graphics

You can add tactical graphics and edit existing tactical graphics just like you can add and edit simulation objects. You can create a new tactical graphic from an existing one or create a new tactical graphic using one of the basic graphic types: point, route, area, symbol, box, or circle.

When you edit a tactical graphic, you can change the default characteristics of that tactical graphic. When VR-Forces users create one of these graphics, they can edit its specific characteristics as desired.

The basic procedures for adding and editing tactical graphics are the same as for adding and editing simulation objects. This section describes procedures that are specific to tactical graphics.

3.7.1 Editing a Tactical Graphic's Menu Icon

Most objects on the Create menu and the various object creation palettes have icons associated with them. For simulation objects, the icon is used to represent them in the GUI. For tactical graphics, the icon illustrates the type of graphic. You can change the icon associated with a tactical graphic and add icons for newly created tactical graphics.

- To specify the palette icon for a tactical graphic, on the General tab, click the Browse button () for the Menu Icon line (Figure 12 on page 36) and select the icon that you want to use.
- To remove the icon associated with a tactical graphic, click the Delete button () on the Menu Icon line.

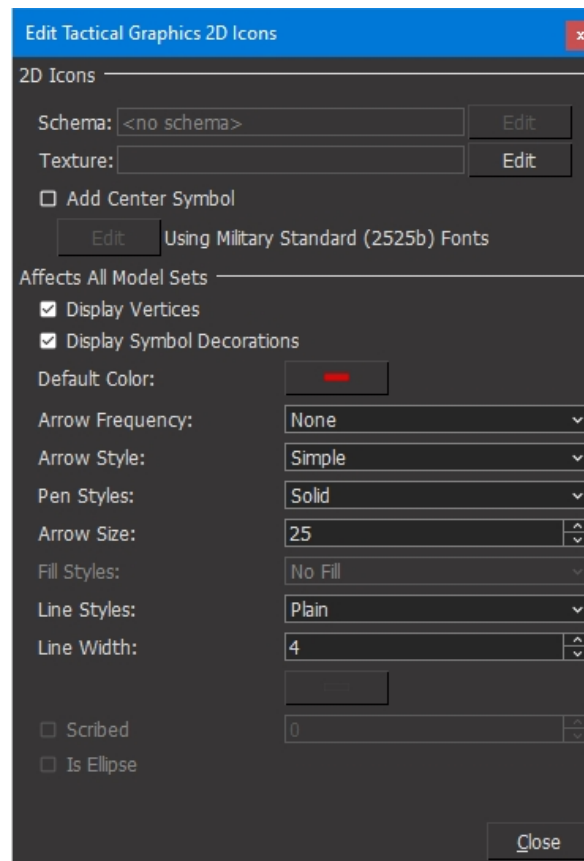
3.7.2 Setting the Default Values for Tactical Graphics Properties

When you select a tactical graphic, the Visual Model group box displays all of the properties that can be set for all of the different types of tactical graphics. Properties that do not apply to the selected tactical graphic are disabled. You can set default values for all of the properties that are applicable to the selected graphic. Setting a default value does not mean that the property will be editable in the VR-Forces GUI. The properties that are editable in the GUI are determined by keywords. For details, see "3.7.3 Specifying Tactical Graphic Properties that Can Be Edited in the GUI" below.

► **To set a default value for a tactical graphic property:**

1. In the Objects List, select the tactical graphic that you want to edit.
2. In the visuals panel, click the Edit button (✎). The Edit Tactical Graphics 2D Icons dialog box opens (Figure 25). The applicable properties are enabled.

Figure 25. Edit Tactical Graphics 2D Icons dialog box



3. Set the default values.

3.7.3 Specifying Tactical Graphic Properties that Can Be Edited in the GUI

When you create a tactical graphic in the GUI, you can edit its properties, such as line style, line width, color, and fill. The properties that you can edit in the GUI are determined by keywords that you specify in the Simulation Object Editor.

The keywords are:

- **arrow.** The user can change the object's arrow style.
- **arrowsize.** The user can change the object's arrow size.
- **canorient.** The user can change the object's orientation (heading).
- **defaultlength=<num>.** Specifies the starting length of the tactical graphic in meters.
- **defaultwidth=<num>.** Specifies the starting length of the tactical graphic in meters.
- **extruded.** The object's width and height can be extruded.
- **fill.** The user can change the object's fill.
- **freehand.** The object can be drawn freehand.
- **initialaltitude=<num>.** Specifies the starting altitude for the tactical graphic in meters.
- **max=<num>.** Specifies the maximum number of points allowed.
- **min=<num>.** Specifies the minimum number of points required to create the object.
- **noaltitude.** The user cannot set an altitude for the object.
- **nostyle.** Do not allow style editing.
- **overlay=<name>.** This is a way to define the default overlay for the tactical graphic.
- **penstyle.** The user can change line style for the object.

When you edit a tactical graphic in the GUI, VR-Forces builds an Edit Style dialog box with the properties specified by the keywords. If no editing is allowed, it removes the Edit icon from the Edit Object dialog box.

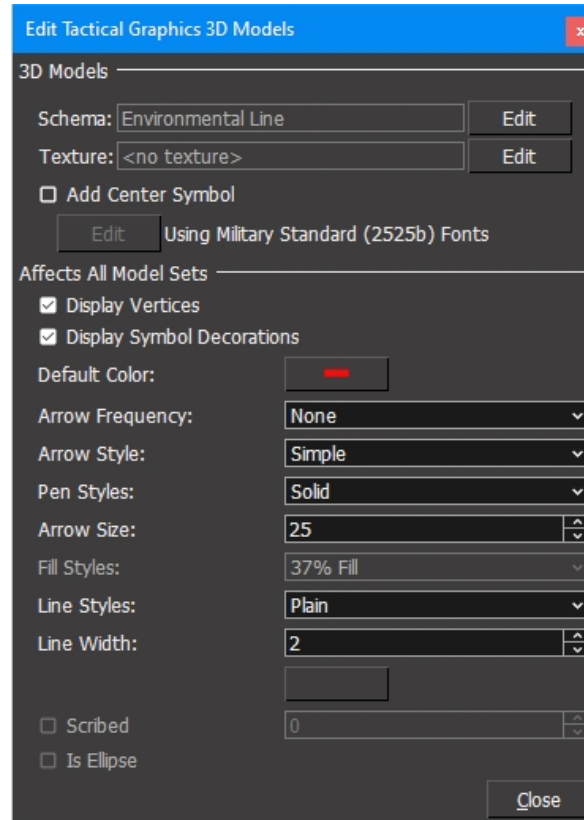
- To allow a tactical graphic's properties to be edited, on the General tab, enter the appropriate keyword in the Keywords text box. Separate each keyword with a semi-colon (;).

3.7.4 Editing the Visual Model for a Tactical Graphic

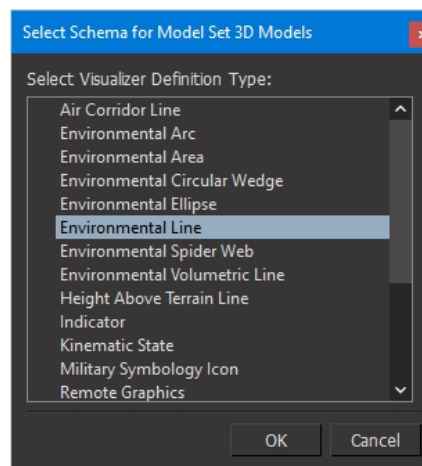
The visual model for a tactical graphic is based on a schema or on a texture. Schemas employ vector graphics and are best for basic lines, points, and areas. Textures apply raster images and are used primarily for specialized linear and areal graphics, such as fortified lines and minefields. (If you use a texture for a graphic, you cannot edit its style properties.)

- **To edit the visual model for a tactical graphic:**

1. In the list of tactical graphics, select the one that you want to edit.
2. In the Visual Model panel, in the Model Sets list, select the model set that you want to edit.
3. Click the Edit button (). The Edit Tactical Graphics *model_set* dialog box opens (Figure 26), where *model_set* is the model set you selected.

Figure 26. Edit Tactical Graphics Models dialog box

4. On the Schema line, click the Edit button. The Select Schema for Model Set *model_set* dialog box opens (Figure 27).

Figure 27. Select Schema for Model Set dialog box

5. Select the schema that you want to use.
6. Click OK.

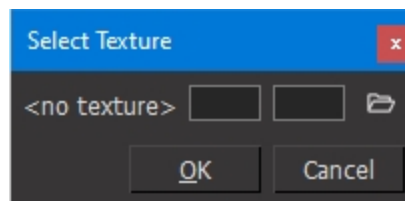
Editing the Texture for a Tactical Graphic

When you use a texture for a tactical graphic you specify a model definition and the height and width of the texture.

► To specify a texture for a tactical graphic:

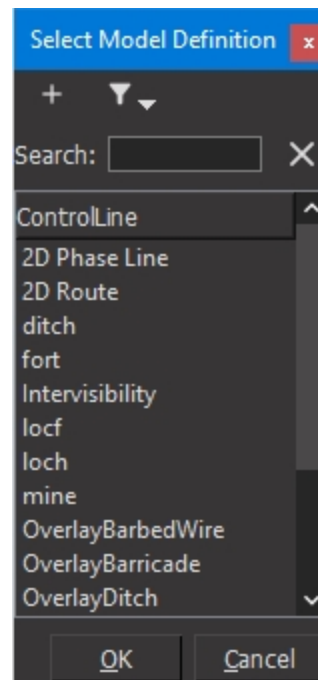
1. In the list of tactical graphics, select the one that you want to edit.
2. In the Visual Model group box, in the Model Sets list, select the model set that you want to edit.
3. Click the Edit button (✎). The Edit Tactical Graphics *model_set* dialog box opens (Figure 26), where *model_set* is the model set you selected.
4. On the Texture line, click Edit. The Select Texture dialog box opens (Figure 28).

Figure 28. Select Texture dialog box



5. Click the Browse button (📁). The Select Model Definition dialog box opens (Figure 29).

Figure 29. Select Model Definition dialog box



6. Select the model definition that you want to use for this texture.
7. Click OK.

8. In the Select Texture dialog box, specify the width and height of the texture, in pixels.
9. If this is an areal graphic, specify whether to use the texture for the outline of the area. If you do not use the texture, a regular line is used.
10. Click OK.

3.8. Creating a Template

When you create a new simulation object, you can select a template to base it on. You can create templates from existing simulation objects. Templates are meant as a starting point. They have standard posture, combat, and defense vales for that type of object. They do not include assemblies or health.

In many ways, basing a new model on a template is not much different from creating a new model from an existing model. However, when you create a template, you can edit it to be more generic than a particular simulation object, which makes it more broadly applicable. Templates are particularly useful for creating aggregate-level units because they are so much more complex than simulation objects in entity-level scenarios.

Templates are named with the format [Template] *object_name*, saved in the `./vrfSim/templates` folder of the SMS's directory, and included in the Template list of the Create New Model dialog box.

► **To create a template:**

1. In the objects list, select the simulation object that you want to use as a template.
2. Choose **Model > Make Template**. A template is created.
3. Optionally, select the template and edit it as you would any simulation object.

3.9. Editing Systems in the Simulation Object Editor

You can create and edit systems in the Simulation Object Editor. The support for different system types varies. Details of the use of the SOE to edit particular systems is described in the chapters devoted to those system types.

When you select a system, the SOE presents information and options on one or more tabs. All systems display the General tab, which includes:

- Name. System name.
- Description. A description of the system.
- Supported entity kinds. A list of the entity types that can use this system.
- Objects that use the system. This list is read-only. It updates when you add or remove a system from a simulation object.
- Parameters. A list of the parameters that you can edit for individual simulation objects. This field is read-only.
- Tags. The tags that you want to apply to the system.

Most systems have a set of parameters that have default values. When you add a system to a simulation object, you can edit the parameters for that simulation object. The Attributes tab lists the system parameters for which you can set default values. Some systems have additional tabs for parameters, largely as a convenience to keep the window size manageable.

Weapon systems have a Targets tab that lets you configure hit probabilities, ammunition, and other targeting details. Damage systems have a Damage tab where you can configure how entities are damaged by different power types.

Munitions are on a separate tab in the Objects List Panel. They have a Detonation tab for configuring the power types supported by the munition.

The Damage Probabilities dialog box helps you understand the impact of different types of weapons on different types of entities.

3.9.1 Creating a New System

You can create a new system by copying an existing system or by basing it on a template. System templates represent a type of system that has a set of parameters that all systems of that type typically have. The choice between creating a new system by copying an existing one or using a template is a matter of estimating how much you must edit the system once you create it. For example, the parameter values for a 130 mm gun are probably very similar to those of a 125 mm gun, in which case you might want to simply copy the 125 mm gun. But they might be quite different for a 70 mm gun, in which case you might want to create the new system from a template.

Creating a New System from an Existing System

If you want to create a new system that is very similar to an existing system, you can copy the existing system to a new system and then edit the parameters.

► **To create a new system from an existing system:**

1. Select the Systems tab.
2. Select the system that you want to copy.
3. Choose **System > New System from Existing**. The New System from Existing dialog box opens.
4. In the System Name box, enter a name for the new system.
5. In the System Filename box, enter a filename for the new system or click the Browse button () and select a system definition file. If you select a system definition file, the new system will overwrite the existing system.
6. Click OK. The new system is added to the system list. Edit the system as desired.

Creating a New System from a System Template

If you want to create a system that has all the basic elements of a certain type of system, but which will not be very similar to an existing system, you can create it from a system template.

► **To create a new system from a system template:**

1. Choose **System > New System**. The New System from Template dialog box opens.
2. In the System Type list, select the type of system you want to create. If the type of system you want to create is not in the list, VR-Forces does not have templates for that system type.
3. In the System Template list, select the template that you want to use for the new system. The dialog box provides a brief description of the selected system type. It lists the systems that use the selected template and the simulation object types that support that system.
4. In the System Name box, enter a name for the new system.
5. In the System Filename box, accept the default system definition filename or enter the name you want to use.
6. Click OK. The new system is added to the system list.
7. Edit the parameters of the system.

3.9.2 Editing Probability Tables

Some systems, such as weapon systems and damage systems, use probability tables to determine the likelihood that a weapon will strike an entity, an entity will be damaged by a munition, and so on. These tables can be specified as a range list or a coefficient list.

A range list is a step function, a function that increases or decreases abruptly from one constant value to another. A coefficient list is a continuous function. Its values change according to a curve specified by a polynomial. You can specify the probabilities by editing a table. Once you create a table, you can edit it directly or graphically.

Figure 30 illustrates the following range list:

```
(range-list
  (range 1000.000
    (probability 0.9000)
  )
  (range 2000.000
    (probability 0.8000)
  )
  (range 3000.000
    (probability 0.7000)
  )
  (range 4000.000
    (probability 0.7000)
  )
)
```

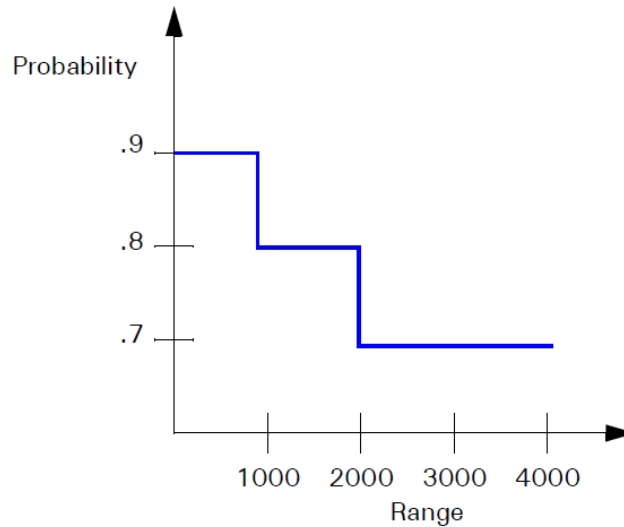
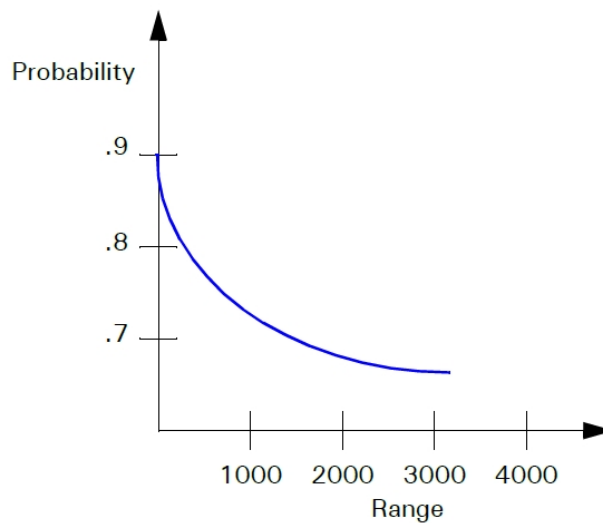
Figure 30. Range list (step function)

Figure 31 illustrates the curve specified by a coefficient list. The hit probability decreases along the curve rather than at specified steps. (If a coefficient list only has two points, it is graphed as a straight line, not a curve.)

Figure 31. Coefficient list polynomial curve

Editing a Range List Probability Table

Range lists are specific to a particular target type, damage power, or other system parameter. This procedure assumes that you have selected the system parameter for which you want to edit a range list.

► To edit a range list table:

1. To add a column to the table (that is, add a range value):
 - a. Click the Add button (+) above the table. The New Range dialog box opens.
 - b. Enter a range value.
 - c. Click OK. A column is added to the table. It is automatically sorted to fit in the correct location relative to the other columns.
2. To edit a a range value:
 - a. Select the column whose range you want to edit. (If you select a row, the column defaults to the first column.)
 - b. Click the Edit button (✎) above the table. The Edit dialog box opens. (You can also double-click the range to open the Edit dialog box.)
 - c. Enter a range value.
 - d. Click OK. The range in the column heading is changed. The table is resorted if necessary.
3. To edit the probability for a given range:
 - a. Select the probability value.
 - b. Enter a probability as a whole number, for example, for 30%, enter 30.

Editing a Range List by Editing its Graph

When you edit a range list graph, you can change the range values and percentage for existing ranges. You cannot add new ranges to the table.

If a system has more than one parameter type, there is a graph for each type. You can edit each of them individually. (Depending on range settings, the graphs might overlap.)

Each graph has a different color. You can change the colors assigned to each graph.

► To edit a range list graph:

1. Move the mouse cursor over the graph. It changes to cross-hairs.
2. Hold down the mouse button above the horizontal or vertical portion of the graph that you want to edit.
3. Drag the line to the new value. The value in the corresponding cell in the table changes as you drag the line.

Editing a Coefficient List Probability Table

The columns in a coefficient list probability table have predetermined values: 1, r , r^2 , r^3 , and so on. You can set the value for each column.

► To edit a coefficient list table:

1. To add a column to the table (that is, add a coefficient value), click the Add button (+) above the table. A new column is added to the table.
2. To edit the probability for a given coefficient:

- a. Select the probability value.
- b. Enter a probability.

Changing the Color of a Probability Graph Line

Each system parameter that is being graphed is listed below the horizontal label for the graph. A color swatch next to the parameter name indicates the color used for that line in the graph.

► **To change the color used for a line in a probability graph:**

1. Click the label for the parameter whose color you want to change. A color picker dialog box opens.
2. Select the color you want to use.
3. Click OK.

Changing the Type of List for a Probability Table

You can change the type of probability table you use for a system parameter:

- To change from a range list to a coefficient list, click the Coefficient List button () above the table. Click Yes in the confirmation dialog box.
- To change from a coefficient list to a range list, click the Range List button () above the table. Click Yes in the confirmation dialog box.

3.9.3 Deleting a System

► **To delete a system:**

1. Select the system in the system list.
2. Choose **System > Delete System**. The system name is struck-through.
3. Save the SMS.

3.9.4 Reverting a System

If you edit a system and have not yet saved the SMS, you can revert all changes to the system since the SMS was last saved.

► **To revert changes to a system:**

1. Select the system in the system list.
2. Choose **System > Revert System**.
3. Save the SMS.



4. Editing Entities for Entity-Level Scenarios

This chapter explains how to use the Simulation Object Editor to edit individual objects in *EntityLevel.sms*. Many of the concepts and procedures also apply to units. For details about editing units, see "Chapter 5. Editing Units for Entity-Level Scenarios" on page 103 and "Chapter 6. Editing Aggregate-Level Objects" on page 115.

4.1. Editing Simulation Object Parameters	75
4.1.1 Editing a Simulation Object's Bounding Volume (Size)	75
4.1.2 Displaying a Simulation Object's Bounding Volume	76
4.1.3 Specifying Support Points	77
4.1.4 Excluding Entities from Batch Bounding Volume and Support Point Updates	78
4.1.5 Editing Behavioral Parameters	78
4.1.6 Adding Object Geometry to a Simulation Object	78
4.1.7 Configuring Cultural Features To Be Embedded in the Terrain	79
4.1.8 Ground Clamping Cultural Features	80
4.1.9 Specifying Entity Properties that Can Be Edited in the GUI	80
4.1.10 Editing a Lifeform's Visual Model and Animation	81
4.1.11 Specifying a Default Appearance	84
4.2. Editing Sensor Signatures	85
4.3. Editing a Model's Articulated Parts	85
4.3.1 Adding Articulated Parts	85
4.3.2 Editing Articulated Parts	86
4.3.3 Animating Articulated Parts	87
4.3.4 Displaying Muzzle Flashes	87
4.3.5 Removing Articulated Parts	87
4.4. Editing a Simulation Object's Systems	88
4.4.1 Adding a System	89
4.4.2 Selecting a Damage Model	90
4.4.3 Selecting a Movement Model	90
4.4.4 Editing a System's Properties	90

4.4.5 Removing a System	90
4.4.6 Replacing a System	91
4.5. Creating and Editing Munitions	91
4.5.1 Creating a Simulated Munition	91
4.5.2 Displaying Munition Detonations	92
4.5.3 Creating a New Ammunition Type	92
4.6. Configuring Wakes	92
4.7. Creating Randomized Simulation Objects	94
4.8. Configuring Scripted Entity Movement	97
4.8.1 Configuring Ballistic Missile Movement	97
4.8.2 Configuring Animated Movement	99
4.8.3 Editing Movement File Configurations	101

4.1. Editing Simulation Object Parameters

This section explains how to edit parameters that are only applicable to simulation objects and cultural features. (Cultural features are objects that you can add to a scenario to simulate human artifacts such as buildings, bridges, and so on. They are modeled as simulation objects because they exist in the physical world.)

4.1.1 Editing a Simulation Object's Bounding Volume (Size)

VR-Forces uses a simulation object's bounding volume to do intersection tests. The bounding volume is specified by the values in the Size/Supports section of the Physical tab. You can display a simulation object's bounding volume in the Visual Model panel. If you change size values, the bounding volume changes accordingly.

To set the bounding volume, you can:

- Edit the value manually.
- Calculate the bounding volume of a particular entity based on the model.
- Calculate the bounding volumes for all entities based on their models.



- The bounding volume is not the same thing as the selection box that is displayed around a simulation object when you select it. The selection box is just part of the GUI display. The bounding volume is used by the simulation engine.
- You can exclude individual entities from the global updates.
- VR-Forces calculates the bounding volume for units based on the locations and bounding volumes of the members of the unit.

► **To change an entity's bounding volume manually:**

1. Select the entity you want to edit in the object list.
2. Select the Physical tab (Figure 32).

Figure 32. Physical tab

Size/Supports

Dimensions (Length, Width, Height)(m): 16.10 17.31 4.55

Origin: Center

Offset (Forward, Right, Up)(m): 0.00 0.00 0.00

Left Support (Forward, Right, Up)(m): -0.14 -2.76 -2.27

Right Support (Forward, Right, Up)(m): -0.14 2.76 -2.27

Other Support (Forward, Right, Up)(m): 4.86 0.29 -2.27

☐ Exclude from batch update

Appearance: 0X166000

Sensor Signatures

Predefined Signatures: Standard Fixed Wing

Active Sonar(km): 0.00

Infrared(km): 20.00

MAD Sensor(km): 0.00

Passive Sonar(km): 0.00

Radar(km): 50.00

Visual(km): 4.00

Articulated Parts

Animate	Fire	Part Type	Part Number	Comment	Associated Component
☐	☐	tank or APC hatch	3232		
☐	☐	Primary gun number 1	4416		
		Landing Gear	3072		movement:landinggear
		Primary launcher 1	4736		maverick-missile-launcher:missile-launcher
		Primary launcher 2	4768		sidewinder-missile-launcher:missile-launcher

3. Change any of the values in the Size/Supports section.

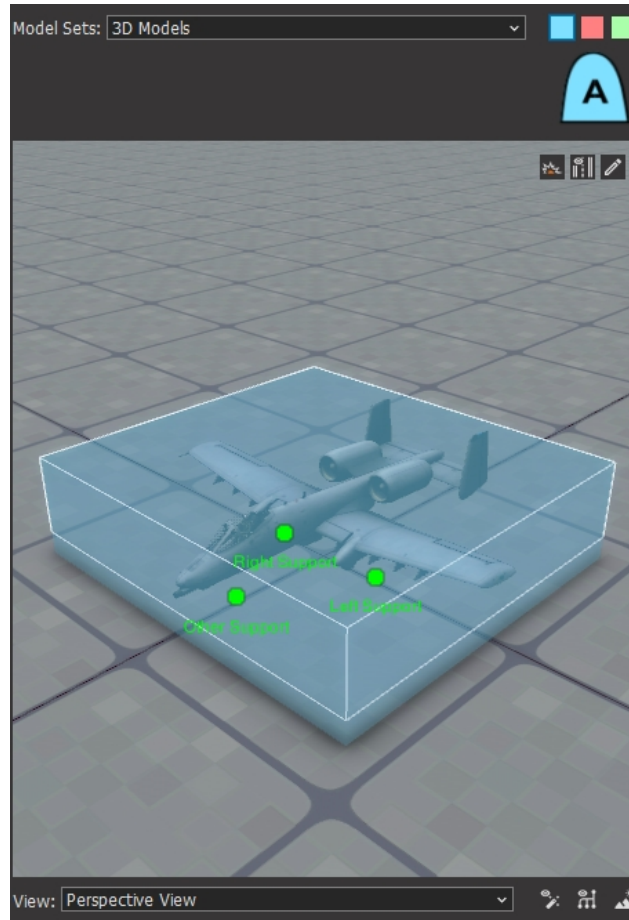
- To calculate the bounding volume of an entity from its model, click the Calculate Support Points and Bounding Volume from Displayed Model button (🌐) or the Calculate Bounding Volume from Displayed Model button (🌐).
- To calculate the bounding volume for all entities, choose **Model > Check Bounding Volume and Support Points**.

4.1.2 Displaying a Simulation Object's Bounding Volume

i You must run VR-Forces at least once before you can display bounding volumes in the Simulation Object Editor.

- **To display a simulation object's bounding volume:**
 1. Select the entity in the object list.
 2. In the Visual Model Panel, from the Model Sets list, select 3D Models.
 3. Choose **Preferences > Show Bounding Volumes**. The bounding volume displays (Figure 33).

Figure 33. Bounding volume



4.1.3 Specifying Support Points

Support points are the locations on a model that contact the ground. To specify support points, you can:

- Set them manually.
- Calculate them from the 3D model.
- Calculate the support points for all entities based on their models.

► **To specify support points manually:**

1. Select the entity you want to edit in the object list.
2. Select the Physical tab (Figure 32 on the previous page).
3. In the Size/Supports section, enter the desired values.

► To calculate support points based on the model, in the Size/Supports section, click the Calculate Support Points and Bounding Volume from Displayed Model button (🌐) or the Calculate Support Points from Displayed Model button (📐).

► To calculate the support points for all entities, choose **Model > Check Bounding Volume and Support Points**.

4.1.4 Excluding Entities from Batch Bounding Volume and Support Point Updates

If you run the Compute Bounding Volume and Support Points command, it updates all entities. If you have specified the bounding volume or support points for an entity manually, this operation might override those settings. You can exclude individual entities from the global update process.

► **To exclude an entity from the global update process:**

1. Select the entity in the object list.
2. Select the Physical tab (Figure 32 on page 76).
3. In the Size/Supports section, select the Exclude From Batch Update check box.

4.1.5 Editing Behavioral Parameters

The Simulation Object Editor lets you edit some of the parameters that affect simulation object behavior.

i The Sensor Signature parameters specify the distance at which other simulation objects can sense the simulation object you are editing. By contrast, the sensor systems on the simulation object control its ability to sense other simulation objects. For more information, see "4.4. Editing a Simulation Object's Systems" on page 88.

- To edit simulation object parameters, edit any of the parameters in the Physical and Movement tabs. (The tabs reconfigure as you select different types of simulation objects.)

4.1.6 Adding Object Geometry to a Simulation Object

Object geometry is to a simulation object as terrain is to the earth. It allows embarked entities to move about on the entity on which they are embarked and it allows entities to move about on cultural features that are embedded in the terrain or embarked on other simulation objects. For example, adding object geometry to an aircraft carrier lets planes taxi on the deck of the carrier.

You can add object geometry to simulation objects in the Simulation Object Editor. Object geometry must be in OpenFlight, MEDF, or GDB format. OpenFlight geometry is converted to GDB and saved. For complete details about adding object geometry and generating navigation data, see "Generating Navigation Data for Entities".

Ordinarily, you would specify an object geometry file for an entity. However, some entities have multiple main model visualizers in their element definitions. If an entity has multiple main models, VR-Forces can scan the element definition to identify all of the relevant object geometry files.

i When you edit an entity's embarkation points and slots, the model used is the visual model. This will not necessarily line up with the object geometry for that entity, if it has any.

► To add object geometry to a simulation object:

1. Select the simulation object in the Object List.
2. Select the Movement tab (Figure 34).

Figure 34. Movement tab for A-10 Thunderbolt

The screenshot shows the 'Movement' tab for the 'Fighter Jet' entity. The parameters are as follows:

Parameter	Value
Mass(kilograms):	19000.00
Maximum Flight Speed (Over Ground)(km/h):	700.00
Maximum Altitude(m):	13700.00
Minimum Air Speed(km/h):	220.00
Default Air Speed(km/h):	560.00
Maximum Thrust Acceleration(m/s ²):	4.50
Maximum Turn Acceleration(m/s ²):	49.00
Maximum Deceleration(m/s ²):	20.00
Maximum Climb Rate(m/min):	1800.00
Maximum Yaw Rate(deg/s):	40.51
Maximum Pitch Rate(deg/s):	28.65
Maximum Roll Rate(deg/s):	85.94
On Ground Tolerance(m):	2.50
Default Ground Speed(km/h):	10.00
Maximum Taxi Speed(km/h):	360.00
Maximum Ground Deceleration(m/s ²):	30.00
Ground Turning Radius(m):	5.00
Flaps Deceleration(m/s ²):	10.00
Can Embark	☒
Can Be Moved Onto For Embark	☐
Can Be Embarked Upon	☐
Protect Embarked Entities from Collateral Damage	☒
Use Detailed Geometry in Simulation Engine	☒
Use Object Visuals	☐
Geometry File	\$(SHARED_DATA_DIR)\ModelData\Vehicles\FixedWing\A-10\A-10. (Select...)
NavigationData	None (Generate) (Clear)

3. Select Use Detailed Geometry in Simulation Engine.
4. If the entity has multiple main models, select the Use Object Visuals option.
5. If it has just one main model, select the option that lists an medf file.
6. Optionally, to change the object geometry file:
 - a. Click Select. The Select Object Geometry dialog box opens.
 - b. Click the Browse button. The Object Geometry dialog box opens.
 - c. Select the file that contains the object geometry for this entity.
 - d. Click Open.
 - e. Click OK.

4.1.7 Configuring Cultural Features To Be Embedded in the Terrain

If a cultural feature, such as a house, has object geometry, you can specify that it can be embedded in the terrain. When you place such an object into a navigation area, VR-Forces automatically generates navigation data for that object. This allows entities to walk around and inside the building (if it has interior geometry) as if it were part of the terrain. For more information, see "Embedding Cultural Features in the Terrain".

i You do not need to generate navigation data in the Simulation Object Editor for cultural features that you want to embed in the terrain. The navigation data is generated when you place the cultural feature in a scenario. Generating navigation data for an object in the Simulation Object Editor is for when you want to embark an entity onto that object.

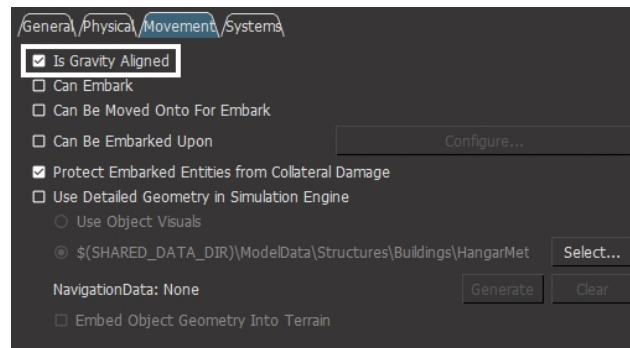
► **To allow a cultural feature to be embedded in the terrain:**

1. Select the cultural feature in the Objects List.
2. Select the Movement tab (Figure 35 below).
3. Select Use Detailed Geometry in Simulation Engine.
4. Click Select. The Select Object Geometry dialog box opens.
5. Click the Browse button. The Object Geometry dialog box opens.
6. Select the file that contains the object geometry for this entity.
7. Click Open.
8. Click OK.
9. Select Embed Object Geometry Into Terrain.

4.1.8 Ground Clamping Cultural Features

You can specify that cultural features be aligned to the terrain (ground clamped). If they are not aligned to the terrain, they might end up sticking out at odd angles.

Figure 35. Is Gravity Aligned check box



- To align cultural features to the terrain, on the Movement tab, select the Is Gravity Aligned check box (Figure 35).

4.1.9 Specifying Entity Properties that Can Be Edited in the GUI

When you create a simulation object in the GUI, you can edit its properties, such as pitch. The properties that you can edit in the GUI are determined by keywords that you specify in the Simulation Object Editor.

The keywords are:

- **canbearborne.** The object can be in the air. This is used for things such as fire and smoke; it is not needed for aircraft.
 - **cansetpitch.** The user can change the object's pitch.
 - **no-orbat.** This object cannot be added to the Order of Battle panel for future creation through the Order of Battle panel.
 - **nocopy.** Once placed, the entity cannot be copied.
 - **noedit.** Once placed, the user cannot edit the object.
- To allow a simulation object's properties to be edited, on the General tab, enter the appropriate keyword in the Keywords text box. Separate each keyword with a semi-colon (;).

4.1.10 Editing a Lifeform's Visual Model and Animation

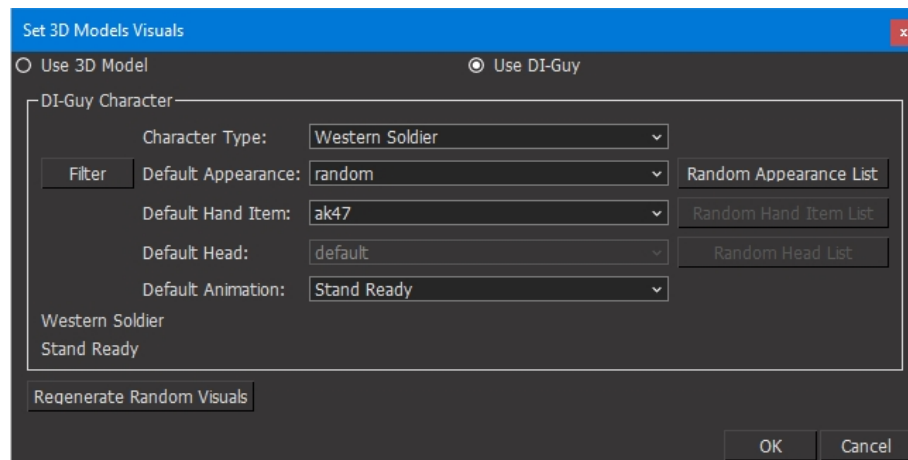
VR-Forces can specify the character, appearance, hand item, head, and animation values for DI-Guy models used in the 3D view and visualization applications such as VR-Vantage. A model's character determines the kinds of movement it can make. The appearance determines what it looks like. The animation specifies the movements that the character makes. In the VR-Forces GUI, you can specify the animation with the DI-Guy Animation task. You can set the appearance with the DI-Guy Appearance set data request. This feature is valid only for lifeforms (the Human and Animal tags).

► **To specify the default DI-Guy characteristics for a lifeform:**

1. Select the Human or Animal tag.
2. Select a lifeform in the object list.
3. In the Visual Model Panel (Figure 36), on the Model Sets list, if 3D Models is not selected, select it.

Figure 36. Visual Model Panel

4. In the 3D display, click the Edit button (). The Set 3D Models Visuals dialog box opens (Figure 37). The Use DI-Guy option should be selected. If it is not selected, select it.

Figure 37. DI-Guy Character group box

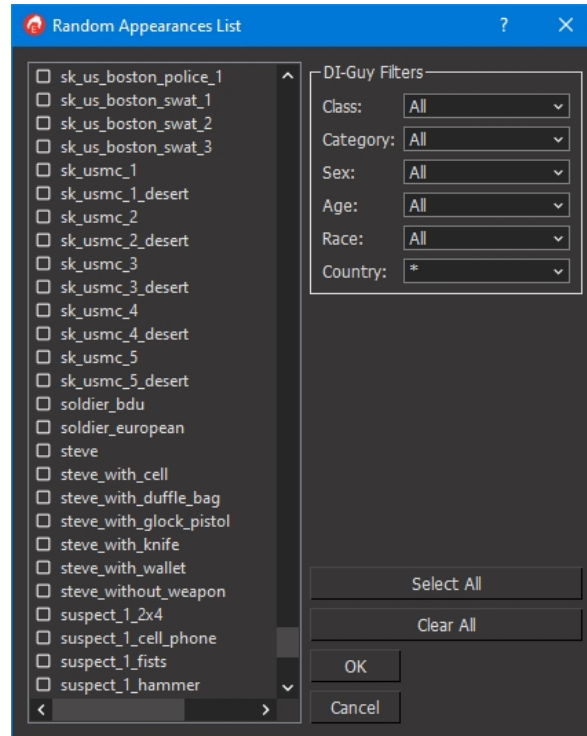
5. Optionally, filter the options available in the lists:
 - a. Click Filter. The DI-Guy Filters dialog box opens.
 - b. Select an option for each characteristic that you want to filter on.
 - c. Click OK.
6. Select the DI-Guy character you want to use from the Character Type list.
7. Select the default appearance from the Default Appearance list. One of the default options is to use a random appearance.
8. If you selected random appearance, optionally, specify the list of appearances to choose from. For details, see "Configuring the Random DI-Guy Characteristics Lists" below.
9. Optionally, in the Default Hand Item list, select a hand item. If you choose random, click Random Hand Item List to specify the hand items that can be chosen.
10. Optionally, in the Default Head list, select a head. If you select random, you can specify the list of heads to choose from. If you choose random, click Random Head List to specify the heads that can be chosen.
11. Select the default animation from the Default Animation list. The default animation is the animation the entity uses if it is not tasked and if it has a posture and speed that is appropriate to the animation.

Configuring the Random DI-Guy Characteristics Lists

When you configure the visual model for a DI-Guy, you can specify that a lifeform use a random appearance, hand item, head, or some combination of these. When VR-Forces creates a lifeform, it chooses from the random characteristic lists. By default, a random list has all options available for the chosen character. However, you can specify a subset of the available characteristics to use.

► To configure the DI-Guy characteristics lists:

1. Select the Human tag (or Animal tag).
2. Select a lifeform in the object list.
3. In the Visual Model Panel (Figure 36), on the Model Sets list, if 3D Models is not selected, select it.
4. In the 3D display, click the Edit button (). The Set 3D Models Visuals dialog box opens (Figure 37). The Use DI-Guy option should be selected. If it is not selected, select it.
5. Click the random list button for the characteristic you want to configure. A dialog box opens (Figure 38). The options available for the character type are listed.
6. Optionally, filter the list by selecting options in the filter lists.

Figure 38. Random Appearances List dialog box

7. Select the options that you want to include in the random list.
8. Clear any options that you do not want to include in the random list.
9. Click OK.
10. Click OK to close the Modify Visuals dialog box.

4.1.11 Specifying a Default Appearance

You can specify the appearance bits for a simulation object that should be used when the object is first created in a scenario. For example, you might want a launcher to default to raised and operational.

► **To specify the simulation object's default appearance:**

1. Select the entity you want to edit in the object list.
2. Select the Physical tab (Figure 32 on page 76).
3. Next to the Appearance box, click the Edit button (✎). The Set Appearance dialog box opens.
4. Set the appearance values as desired.
5. Click OK.

i The controller also affects some appearance bits. For example, the Power Plant On might be set to No in the Simulation Object Editor, but the controller sets to Yes when creating the entity.

4.2. Editing Sensor Signatures

Sensor signatures specify the distance at which other entity sensors can locate a simulation object. VR-Forces includes a set of predefined sensor signatures for each of the sensor domains supported. You can edit the predefined sensor signatures in `./data/simulationModelSets/EntityLevel/extra/sensor-signatures.csv`.

 The VR-Forces simulation engine does not read *sensor-signatures.csv*. The file is only used to set predefined sensor signatures in the Simulation Object Editor. If you make change to the predefined sensor signatures, you must update objects in the Simulation Object Editor to put the changes into effect.

► **To specify sensor signatures:**

1. Select the simulation object.
2. Select the Physical tab (Figure 32 on page 76).
3. In the Sensor Signatures section, specify a value in each domain in which the simulation object can be sensed or select a set of predefined signatures in the Predefined Signatures list.

4.3. Editing a Model's Articulated Parts

The Simulation Object Editor allows you to manage a simulation objects' articulated parts. This applies to articulated parts that are defined for the model, such as propellers, rotors, flaps, wheels, landing gear, tracks, radar, turrets, gun barrels, missile launchers, doors, hatches, and so on.

The default configuration for articulated parts comes from sim 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.

4.3.1 Adding Articulated Parts

You can add articulated parts to a simulation object. If they are associated with a part on the model that does not articulate visually, they can still have articulation information.

► **To add an articulated part:**

1. Select the simulation object.
2. Select the Physical tab (Figure 32 on page 76).
3. In the Articulated Parts section, click Add Part (+). The Add Articulated Part dialog box opens.
4. You can hover the cursor over an available part to see the source file information.
5. Select the part to add and then click OK. The part is added to the list of articulated parts.

6. You can edit the part as needed; see "4.3.2 Editing Articulated Parts" below.
7. Save the simulation model set.

4.3.2 Editing Articulated Parts

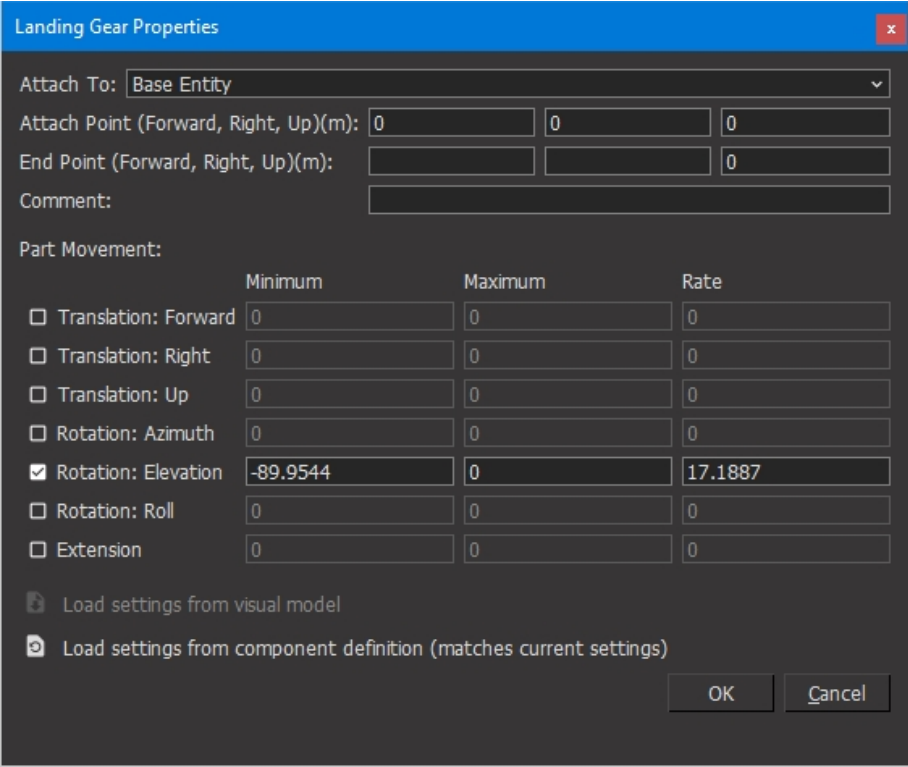
You can make changes to articulated parts on the simulation object. Your edits override any other settings. The articulated parts on the simulation object come from the system definition files, the model definition, and any articulated parts added to the simulation object.

 Hover the cursor over a part in the list to see the source file information.

► **To edit an articulated part:**

1. Select the simulation object.
2. Select the Physical tab.
3. In the Articulated Parts section, select the articulated part you want to edit.
4. Click Edit Part Properties (). The properties dialog box opens (Figure 39).

Figure 39. Properties dialog box for articulated parts



Part Movement:			
	Minimum	Maximum	Rate
☐ Translation: Forward	0	0	0
☐ Translation: Right	0	0	0
☐ Translation: Up	0	0	0
☐ Rotation: Azimuth	0	0	0
☒ Rotation: Elevation	-89.9544	0	17.1887
☐ Rotation: Roll	0	0	0
☐ Extension	0	0	0

5. Make changes as needed.
6. Click OK.
7. Save the simulation model set.

4.3.3 Animating Articulated Parts

You can see the animations of articulated parts that are associated with a part of the model that can be animated.

► **To animate an articulated part:**

1. Select the simulation object.
2. In the Visual Model Panel, from the Model Sets list, select 3D Models.
3. Select the Physical tab.
4. In the Articulated Parts section, select the Animate check box for the articulated part. The part animates in the Visual Model Panel.

4.3.4 Displaying Muzzle Flashes

You can preview the particle system configured for a weapon system to check whether it looks correct. The particle system displays at the point where it is defined. Munition types are configured with a specific particle system.

Select the part type to see the placement of the points. These are defined in the entity element, entity type map, and so on.



- The preview does not animate the articulated part in any way. It only shows the associated particle system.
- This does not work for lifeforms equipped with weapons.

► **To display a muzzle flash:**

1. Select the simulation object.
2. Select the Physical tab (Figure 32 on page 76).
3. In the Articulated Parts section, click Fire () for the weapon that you want to see. The particle system displays.

4.3.5 Removing Articulated Parts

You can remove articulated parts that you added to a simulation object. You cannot remove articulated parts that are part of the model.

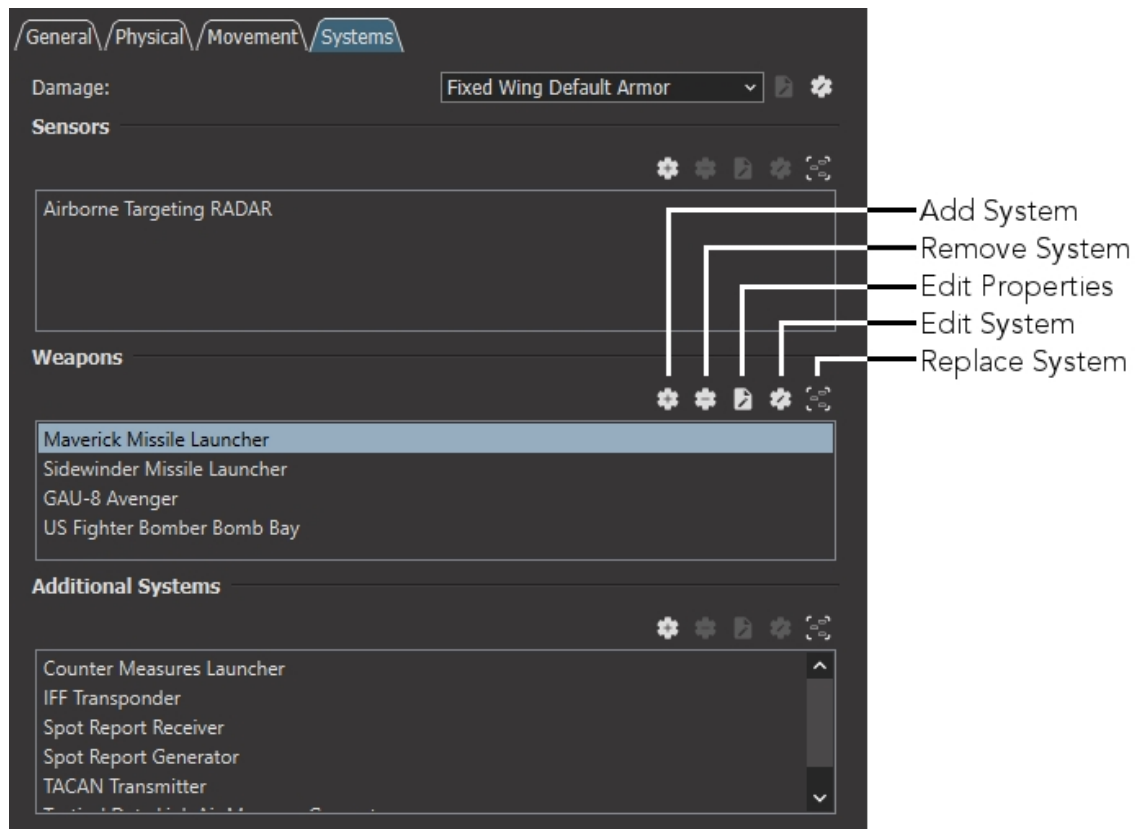
► **To remove a custom articulated part:**

1. Select the simulation object.
2. Select the Physical tab.
3. In the Articulated Parts section, select the articulated part you want to remove.
4. Click Remove Selected Part (). The part is removed from the entity.

4.4. Editing a Simulation Object's Systems

Simulation objects have complex capabilities, such as the ability to move, to take damage, to sense other simulation objects, and to fire munitions. These capabilities are organized as systems. You can quickly change how a simulation object functions by adding, removing, and changing its systems. The Simulation Object Editor lists the systems available in each category and only lists systems that are appropriate for the entity type (Figure 40). For example, a lifeform can have an M-16 or an RPG, but not a cruise missile.

Figure 40. Systems tab



Some systems are interdependent. For example, if you add a weapon system to a simulation object, it probably cannot use it unless it also has a sensor system so that it can detect simulation objects. If you change or add systems, it is up to you to know if they have dependencies.

The components that make up a system have parameters and you can edit the parameters of system components by editing the properties of the system.

Use the Simulation Object Editor to edit these types of systems:

- **Damage.** You can specify the type of damage model that a simulation object uses. When you select a damage model, the Simulation Object Editor configures the damage files that are appropriate for the object type.

- **Sensors.** You can specify the sensors that a simulation object uses to locate other simulation objects in the simulation. (By contrast, sensor signatures, which you can also set, specify the distance at which other entity sensors can locate this simulation object. For a list of the sensor signatures used for VR-Forces by category, see Table 19 on page 205.) When you add a weapon, the Simulation Object Editor configures the components and connections necessary for the sensor to work.
- **Weapons.** The Simulation Object Editor lists the weapon systems that are appropriate for the object type. When you add a weapon, the Simulation Object Editor configures the components, targets, ammunition, and connections necessary for the weapon to work.
- **Movement.** You can specify the type of movement system that a simulation object uses, for example, tracked, wheeled, pedestrian, and so on. When you select a movement system, the Simulation Object Editor configures all of the actuators and controllers that the simulation object needs to implement the system.
- **Additional Systems.** This category includes spot reports, IFF transponders, emergency transponders, and refueling systems.

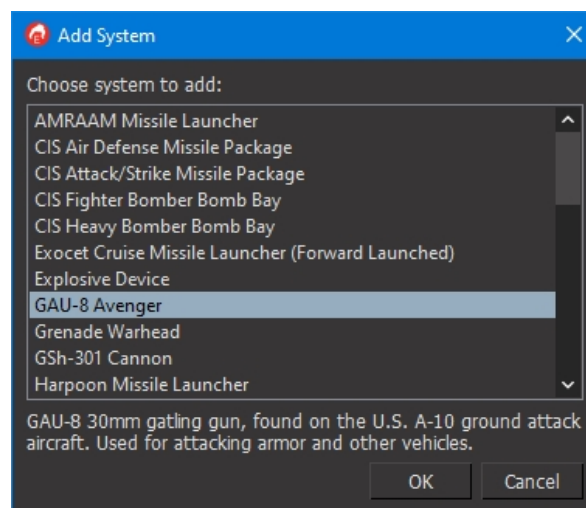
4.4.1 Adding a System

There is no specific limit to the number of sensors, weapons, or other systems that you can configure for a simulation object.

► **To add a system:**

1. Select a simulation object in the list.
2. Select the Systems tab (Figure 40 on the previous page).
3. Click the Add System button (⚙️) for the system type you want to add. The Add System dialog box opens (Figure 41). It lists the systems of this type that are available for the simulation object you are editing. If you select a system in the window, the dialog box displays a brief description.

Figure 41. Add System dialog box



4. Select the system you want to add.
5. Click OK. The system is added to the list in the Simulation Object Editor.

4.4.2 Selecting a Damage Model

You can only have one damage model specified for a simulation object.

- To specify a damage model, on the Systems tab (Figure 40 on page 88), select an option from the Damage list.

4.4.3 Selecting a Movement Model

You can only have one movement model specified for a simulation object.

- To specify a movement model, on the Movement tab (Figure 34 on page 79), select an option from the Movement Dynamics list.

4.4.4 Editing a System's Properties

Some systems have properties that you can edit.

- **To edit a system's properties:**

1. Select the simulation object whose system you want to edit.
2. Select the Systems tab (Figure 40 on page 88).
3. For sensors, weapons, or other systems, select the system whose properties you want to edit. (You do not have to select the damage or movement model, because there can only be one.) If the system has properties, the Properties button is enabled.
4. Click the Edit Properties button (). A Properties dialog box opens. The 3D model in the Visual Model Panel displays a red dot on the simulation object where the system is located. In some cases there are multiple locations.
5. Edit the properties as desired.
6. Click OK.

4.4.5 Removing a System

You can remove sensor, weapon, and additional systems.

 If you remove a system that another system depends on, the other system will not work. It is your responsibility to know these dependencies.

- **To remove a system:**

1. On the Systems tab, select the system you want to remove.
2. Click the Remove System button (.

4.4.6 Replacing a System

You can swap systems on a simulation object by selecting a system you do not want and replacing it with one that you do. This writes the new system entry with the same string-id that the existing one had and simply hides the existing one. For a simulated object that you created as the child of another (described in "3.3.2 Inheriting a New Model from an Existing Model" on page 59), this is a system inherited from the parent.

► **To replace a system:**

1. Select a simulation object in the list.
2. Select the Systems tab (Figure 40 on page 88).
3. Select the system you want to replace.
4. Click the Replace System button (). The Add System dialog box opens (Figure 41 on page 89).
5. Select the system you want the simulation object to have.
6. Click OK. That system is added to the list and the replaced system is removed.

4.5. Creating and Editing Munitions

Munitions fall into two broad categories: simulated munitions and non-simulated munitions. As the name implies, simulated munitions are simulated as individual objects. This category includes missiles, bombs, mines, and hand grenades. Since they are simulated, they have visual and 2D models. They have all of the various tabs for configuring their parameters that other simulation objects have, plus a Firing tab for adjusting their accuracy and a Detonation tab for configuring the damage power.

Non-simulated munitions are not simulated as individual objects. This category is for ammunition that is fired by some of the direct fire weapons, such as rifles, machine guns, and some ballistic rounds. These munitions have a General tab for basic identifying information and a Detonation tab for configuring their damage power.

4.5.1 Creating a Simulated Munition

Simulated munitions have visual models and are configured just like other simulation objects.

► **To create a simulated munition:**

1. Choose **Model > New Munition**. The New Munition dialog box opens.
2. Select the New Munition option.
3. Click OK. The Create New Model dialog box opens.
4. In the Platform Type list, select the platform type for the munition. The Description field provides information about the type of munition represented by this platform.
5. In the Template field, select the template you want to use. It is possible that there are no templates for the platform type.
6. In the Name box, enter a name for the munition.
7. Optionally, edit the Object Type.

8. In the Take Visuals list, select the model you want to use for this munition.
9. Click OK. A new entry is added to the munitions list.
10. If applicable, on the General tab, specify the object types that this munition can target.
11. Edit the Detonation power, as described in "12.2.2 Configuring Detonation Power in the SOE" on page 244.
12. Edit other parameters for this munition as desired.

4.5.2 Displaying Munition Detonations

There is a button for simulated munitions that allows you to preview the detonation in different environments.

► **To display the detonation for simulated munitions:**

1. Select the Munitions tab.
2. Select the munition.
3. Click the new Detonate button () in the 3D display. The Display Detonations dialog box opens.
4. Select the detonation type you want to display.
5. Click the Detonate button.

4.5.3 Creating a New Ammunition Type

Ammunition munitions are not simulated as individual objects.

► **To create a new ammunition type:**

1. Choose **Model > New Munition**. The New Munition dialog box opens.
2. Select the New Ammunition option.
3. Click OK. The Add New Ammunition dialog box opens.
4. Specify the object type for the new ammunition by selecting options from the lists of typing the object type directly.
5. Click OK.
6. Click OK. A new entry is added to the munitions list.
7. On the General tab specify the object types that this munition can target.
8. If this munition has a tracer type, specify it on the General tab.
9. Edit the Detonation power, as described in "12.2.2 Configuring Detonation Power in the SOE" on page 244.

4.6. Configuring Wakes

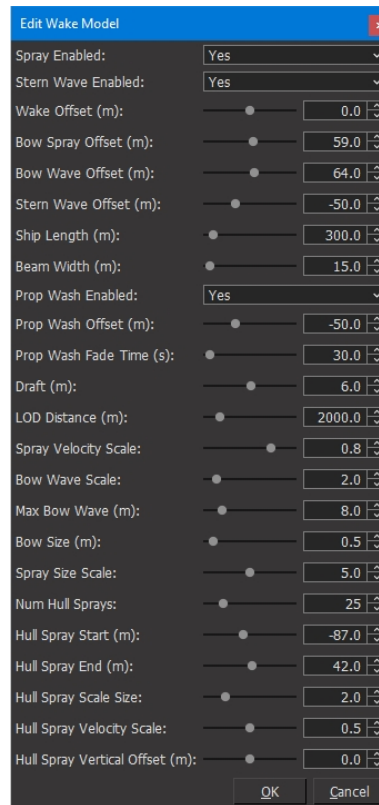
You can configure the wake parameters for surface entities in the Simulation Object Editor. The model window shows the ocean state and you can vary the wind speed, entity speed, and chopiness to see how the wake looks under varying ocean conditions (Figure 42).

Figure 42. Wake display

► **To configure wakes:**

1. Select a surface entity in the object list.
2. Click the Edit Wake button (🌊) on the Visual Models Panel (Figure 18 on page 44). The Edit Wake Model dialog box opens (Figure 43).

Figure 43. Edit Wake Model dialog box



3. Change wake parameter values as desired. For details about wake parameters, see "[Configuring Wakes](#)" in the *Adding Content to MAK ONE Applications Reference Manual*.
4. To see the effect of parameter changes under different sea conditions:
 - a. Click the Environment Settings button (🌊) on the Visual Models Panel. The Environment Settings dialog box opens.
 - b. Adjust the wind speed, wind direction, choppiness, and ocean quality values.
 - c. Click Close.
5. Click OK.

i To view the wakes and propeller wash in the Visual Models Panel, choose **View > Visual Effects** and then adjust the speed.

4.7. Creating Randomized Simulation Objects

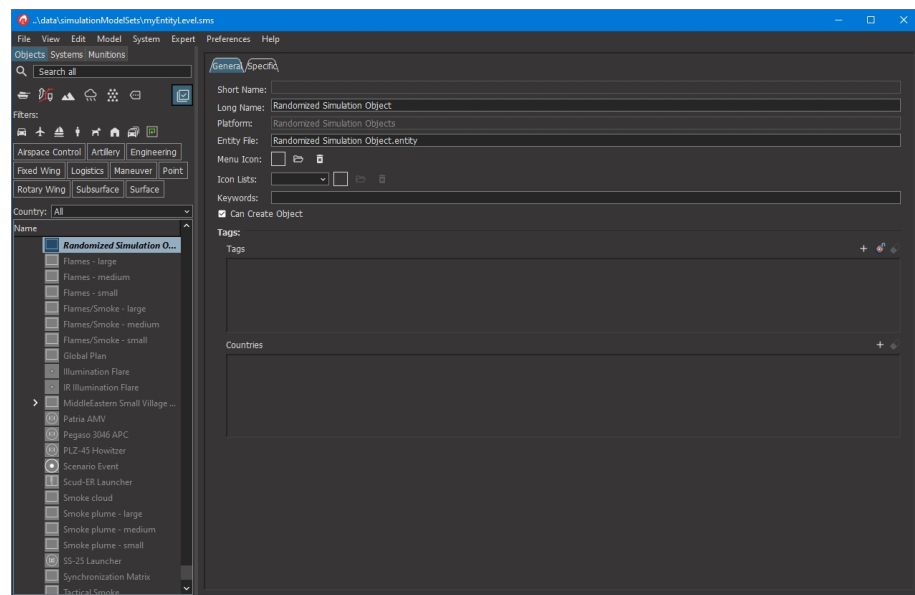
A randomized simulation object is an object that you can use to quickly add a generic simulation object type, such as a civilian vehicle or commercial aircraft, using randomly chosen specific models of these object types. For example, if you want to add several cars to a scenario, you could individually select a Chevrolet Caprice, a Ford Escape, a Toyota Pickup, and so on, and add them to the scenario. Or you could select a randomized simulation object that references a list of civilian vehicles and click multiple times on the terrain. With each click, a randomly chosen vehicle is added to the scenario.

You create randomized simulation objects in the Simulation Object Editor. You can create them in entity-level and aggregate-level scenarios.

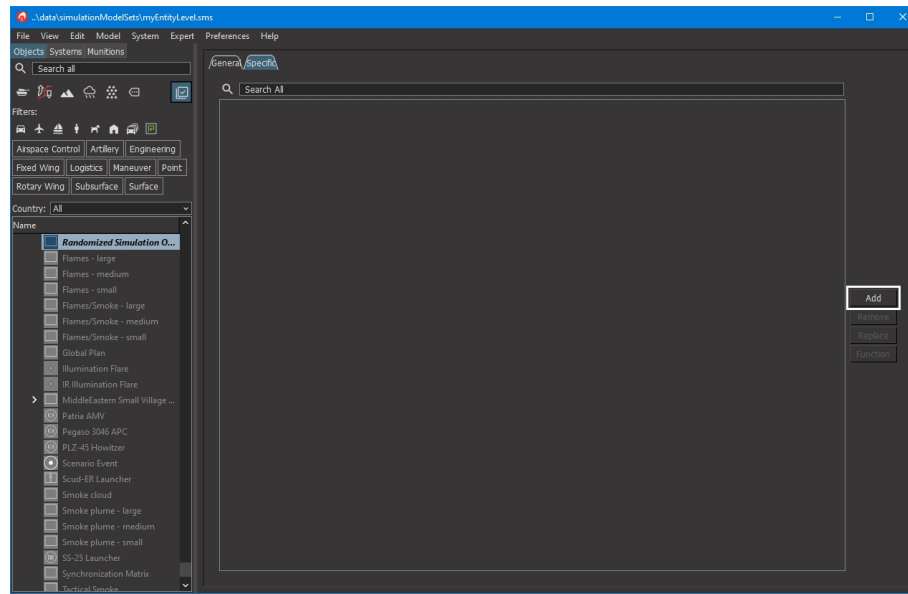
► **To create a randomized simulation object:**

1. Start the Simulation Object Editor.
2. Load the SMS you want to use.
3. Choose **Model > New Randomized Simulation Object**. The Randomized Simulation Object dialog box opens.
4. Enter a name for the simulation object and then click OK. The new object is added to the object list.
5. Select the new object in the object list (Figure 44).

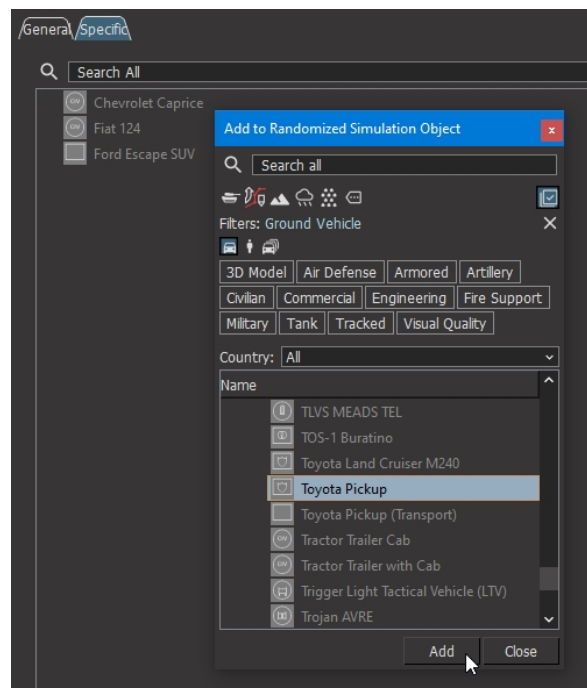
Figure 44. Randomized simulation object



6. Optionally, edit the common simulation object attributes, such as tag, palette, and so on.
7. Select the Specific tab (Figure 45).

Figure 45. Randomized object, Specific tab

8. Click the Add button (Figure 45). The Add to Randomized Simulation Object dialog box opens.
9. Select the simulation objects that you want to be part of this randomized simulation object. To select an object, double-click it or select it and click Add. As you select objects, they are added to the list (Figure 46). If you want the probability of a particular simulation object being created to be higher than others, add multiple instances of the same simulation object to the list.

Figure 46. Randomized object list

10. Click Close.
11. Save the SMS.

4.8. Configuring Scripted Entity Movement

In most cases, simulation objects move based on the intelligence built into their components – sensors, controllers, and actuators. However, there are two instances in which entity movement follows a predetermined sequence, or script. These cases are ballistic missile movement and animated movement tasks.

When you launch a ballistic missile (by creating a missile target in VR-Forces), it does not follow a route or do any path planning; it flies based on the parameters in the scripts it is configured to use. Similarly, if you give an animated movement task to an entity, it moves in whatever way the script prescribes.

Ballistic missile movement is configured using comma-delimited value (CSV) files. Animated movement tasks can be configured with CSV files or 3DS ASCII Export (ASE) files. We refer to these files generically as movement files.

A movement file specifies a time sequence and a set of parameters that define the entity's movement for each time entry. Ballistic missile movement files define lateral-distance, altitude, and range. Animated movement movement files define lateral distance, altitude, range, heading, pitch, and roll.

i Ballistic missile movement files are configured in a dialog box that is shared with animated movement movement files ("4.8.2 Configuring Animated Movement" on page 99). Animated movement movement files have more parameters than ballistic missile movement files. If a ballistic missile is configured with a movement file that has these additional parameters, they are ignored.

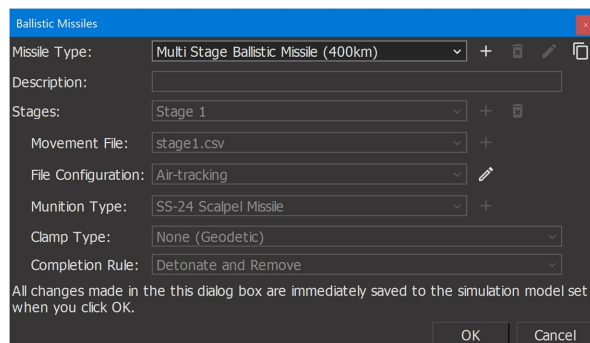
4.8.1 Configuring Ballistic Missile Movement

When you create a Missile Target event in VR-Forces, you specify a missile type. Missile types are configured in the Simulation Object Editor.

► **To configure ballistic missiles:**

1. Choose **Edit > Ballistic Missiles**. The Ballistic Missiles dialog box opens (Figure 47).

Figure 47. Ballistic Missiles dialog box



2. In the Missile Type list, select the missile type that you want to edit. You can add, delete, and rename missile types using the Add (+), Delete (✕), and Rename (✎) buttons.
3. Optionally, in the Description text box, enter a description. This description is used as a tooltip for the Missile Type in the Create Missile Target and Edit Missile Target dialog boxes.
4. In the Stages list, select the stage that you want to configure. Missiles can be single stage or multi-stage. For single stage missiles, configure Stage 1. For multi-stage missiles, use the Add and Delete buttons to add or delete stages.

For each stage, specify:

- Movement File. The CSV file to use. The list lists the files in `./data/simulationModelSets/<model_set>/scriptedObjectMovement`.
If you want to use a CSV file that is not listed, click the Add button (+) next to the Movement File list. A file chooser dialog box opens, where you can select the CSV file that you want to use. The file is copied to `./data/simulationModelSets/<model_set>/scriptedObjectMovement`.
- File Configuration. Select a file configuration from the list. File configurations specify the type of value in each column in the movement file. For details, see "4.8.3 Editing Movement File Configurations" on page 101.
- Munition Type. The missile to use.
- Clamp Type. Determines how to calculate altitude for the animation. Table 3 describes the clamp type options.
- Completion Rule. The action to take when the movement in the CSV file is complete.

5. Click OK.

Table 3. Clamp type options

Option	Description
None (Geodetic)	Altitude in the animation is used as altitude above the altitude of the animation reference location. A constant altitude in the animation results in a path that follows the curvature of the earth, when a round earth terrain is being used.
None (Topographic)	Altitude in the animation is used as altitude above a plane defined by the animation reference orientation. A constant altitude in the animation results in a straight path, even as the earth curves, when a round earth terrain is being used.
High Fidelity (Including Orientation)	Altitude in the animation is ignored, and the object is ground clamped to the terrain using three support points, which also sets the orientation of the object.
Low Fidelity (Terrain Clamp Only)	Altitude in the animation is ignored, and the object is ground clamped to the terrain using only its origin point. Orientation is set solely from the animation.

Table 3. Clamp type options (continued)

Option	Description
Altitude Above Terrain (High Fidelity)	Altitude in the animation is used as the altitude above the terrain height.
Ground Clamp if Altitude Offset is 0 (High Fidelity)	Acts like High Fidelity for frames where altitude is 0, and acts like None (Geodetic) for frames where altitude is non-zero.
Radar Face Cartesian	Similar to None (Topographic) , except the Y coordinate is inverted. The animation would typically be started with an orientation reference corresponding to the angle of a radar face. Useful for cases where animation data is captured radar data.

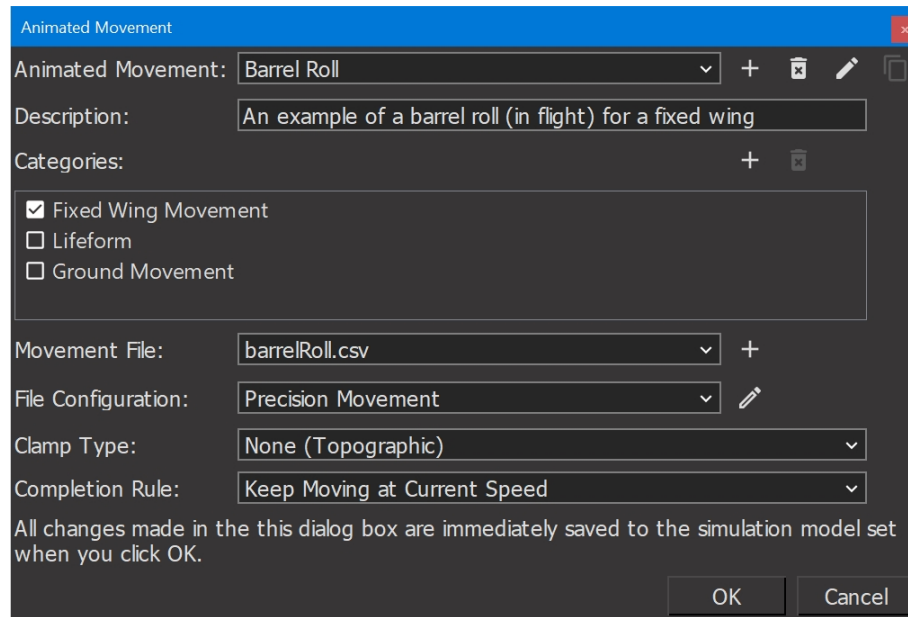
4.8.2 Configuring Animated Movement

The animated movement feature lets you script a sequence of entity movements. You can do this using a CSV file or a 3DS ASCII Export (ASE) file. Typically, movement are created using telemetry data. VR-Forces includes some sample animated movement files. You configure the animated movement options in the Simulation Object Editor.

Animated movement is assigned to an entity using the Animated Movement task, which causes an entity to follow a predefined path defined in an external file. The entity moves between extrapolated points without the ability to respond to external stimuli. For example, this could be illustrated by a ballistic missile flying along a prerecorded telemetry data path.

► **To configure animated movement:**

1. Choose **Edit > Animated Movement**. The Animated Movement dialog box opens (Figure 48).

Figure 48. Animated Movement dialog box

2. To edit an existing animated movement, select it in the Animated Movement list. To create a new animated movement:
 - a. Click the Add button (+) next to the Animated Movement list. The New Configuration dialog box opens.
 - b. Enter a name for the new animated movement.
 - c. Click OK.
3. Optionally, enter a description for the animated movement. This description is displayed in the Animated Movement dialog box when an Animated Movement task is assigned.
4. In the Categories window, select the categories that this animated movement applies to. Use the Add and Delete buttons to add and delete categories.
5. In the Movement File list, select the movement file to use. The list lists the files in `./data/simulationModelSets/<model_set>/scriptedObjectMovement`. If you want to use a CSV file or ASE file that is not listed:
 - a. Click the Add button (+) next to the Movement File list. A file chooser dialog box opens.
 - b. Select the file that you want to use. The file is copied to `./data/simulationModelSets/<model_set>/scriptedObjectMovement`.
6. If you chose an ASE file, the dialog box displays a field called Animation. Select an animation from the list.

If you chose a CSV file, it displays a field called File Configuration. Select a file configuration from the list. File configurations specify the type of value in each column in the movement file. For details, see "4.8.3 Editing Movement File Configurations" on the facing page.

7. In the Clamp Type list, select an option for calculating the altitude. For details, see Table 3 on page 98.
8. In the Completion Rule list, select the action you want the entity to take after it completes the animated movement task.

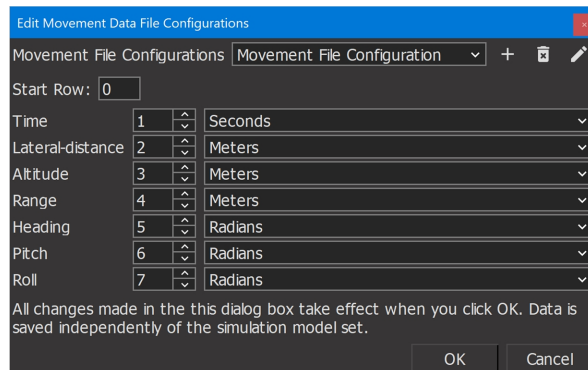
4.8.3 Editing Movement File Configurations

VR-Forces assumes that a ballistic missile CSV file contains four columns of data that describe time, lateral distance, altitude, and range. It assumes that animated movement files have seven columns that describe time, lateral distance, altitude, range, heading, pitch, and roll. VR-Forces includes file configurations for the sample CSV files provided. They are organized in the order that the columns are listed in this paragraph. If a CSV file you are using is organized differently, you must create a file configuration that matches your file or edit one of the provided configurations.

► **To create or edit a movement file configuration:**

1. In the Ballistic Missiles dialog box or the Animated Movement dialog box, click the Edit button (✎) next to the File Configuration list. The Edit Movement Data File Configurations dialog box opens (Figure 49).

Figure 49. Edit Movement Data File Configurations dialog box



- Ballistic missile file configurations have four rows. Animated movement file configurations have seven rows. If you specify a seven-row configuration for a ballistic missile, it ignores the values that it does not use.
- If you are editing an animated movement that uses an ASE file, there is no File Configuration option. This only applies to CSV files.

2. In the Movement File Configurations list, select the configuration you want to edit. If you are creating a new configuration, click the Add button (+). The New Configuration dialog box opens.
 - a. Enter a name for the configuration.
 - b. Click OK.

3. Optionally, change the Start Row value. This is the row in the CSV file from which VR-Forces should start reading values for entity movement.
4. Specify the column in the CSV file that has the Time value.
5. Optionally, select the time unit from the list.
6. Repeat steps 4 and 5 for each movement parameter.
7. Click OK.



5. Editing Units for Entity-Level Scenarios

This chapter explains how to edit units for entity-level scenarios (*EntityLevel.sms*) in the Simulation Object Editor.

5.1. Introduction to Editing Units	104
5.2. Creating a Unit for Entity-Level Scenarios	104
5.3. Specifying an Aggregate As Option	104
5.4. Editing Unit Composition	105
5.4.1 Adding Subordinates	106
5.4.2 Removing Subordinates	107
5.4.3 Replacing Subordinates	107
5.4.4 Changing the Order of Subordinates	108
5.4.5 Editing a Subordinate Unit	108
5.4.6 Defining Functions for Subordinates	108
5.5. Editing Formations	109
5.5.1 Adding Formations	110
5.5.2 Removing Formations	111
5.5.3 Renaming Formations	111
5.5.4 Specifying the Default Formation	111
5.5.5 Expanding and Collapsing the Display of Formations	111
5.5.6 Displaying Bounding Boxes	111
5.5.7 Copying Formations	112
5.5.8 Laying Out a Formation Automatically	113
5.5.9 Changing a Formation's Layout by Hand	113

5.1. Introduction to Editing Units

VR-Forces can simulate units in two different ways:

- In entity-level scenarios, VR-Forces models the individual entities of a unit. Units for entity-level scenarios are configured in *EntityLevel.sms* or SMSs based on it.
- Units in aggregate-level scenarios do not model individual entities. Combat operations are data-driven and are not based on individual entity interactions. Units for aggregate-level scenarios are configured in *AggregateLevelBase.sms*, *AggregateLevel.sms*, or SMSs based on them.

Units in entity-level scenarios and aggregate-level scenarios have many of the same parameters as individual entities. Therefore, for the most part, configuring them in the Simulation Object Editor involves the same procedures as those described in "Chapter 2. The Simulation Object Editor and SMSs" on page 3 and "Chapter 4. Editing Entities for Entity-Level Scenarios" on page 73.

The two major differences in configuring the different types of units is that units in entity-level scenarios have subordinates and formations and do not have sensors. Units in aggregate-level scenarios do not move in formation. They have sensors and they have a large amount of information about their combat capabilities, personnel, equipment, and supplies.

5.2. Creating a Unit for Entity-Level Scenarios

► **To create a unit for entity-level scenarios:**

1. Follow the procedures for creating a new entity in "3.3. Creating a New Simulation Object" on page 57.
2. Add subordinates, as described in "5.4.1 Adding Subordinates" on page 106.
3. Optionally, configure formations.

5.3. Specifying an Aggregate As Option

When you create a unit using the Aggregate As command, the Aggregate As dialog box has a list of unit types that you can select for the unit that is to be created. VR-Forces is configured with just a short list of generic unit types. You can specify that any unit configured in the Simulation Object Editor be included in the list of Aggregate As options.

 This procedure is valid for entity-level and aggregate-level SMSs.

► **To specify that a unit can be an option in the Aggregate As list:**

1. Select the unit in the object list.
2. Select the General tab (Figure 50).

Figure 50. General tab, entity-level unit

General Movement Systems

Short Name: AR Cav

Long Name: Armored Cavalry Platoon

Description:

Echelon Level: PLT

Maximum Disaggregation Distance(m): 4000.00

Platform: Vehicle Unit

Entity File: Armored Cavalry Platoon.entity

Object Type: 11:1:0:3:6:0:0 Edit

Menu Icon:

Icon Lists:

Keywords:

☒ Can Create Object

☒ Allow as Aggregate As Option

Tags:

Tags: Armored^x Empty Unit^x Entity^x Ground Vehicle^x Military^x Unit^x

Countries:

3. Select the Allow as Aggregate As Option check box.
4. Save the SMS.

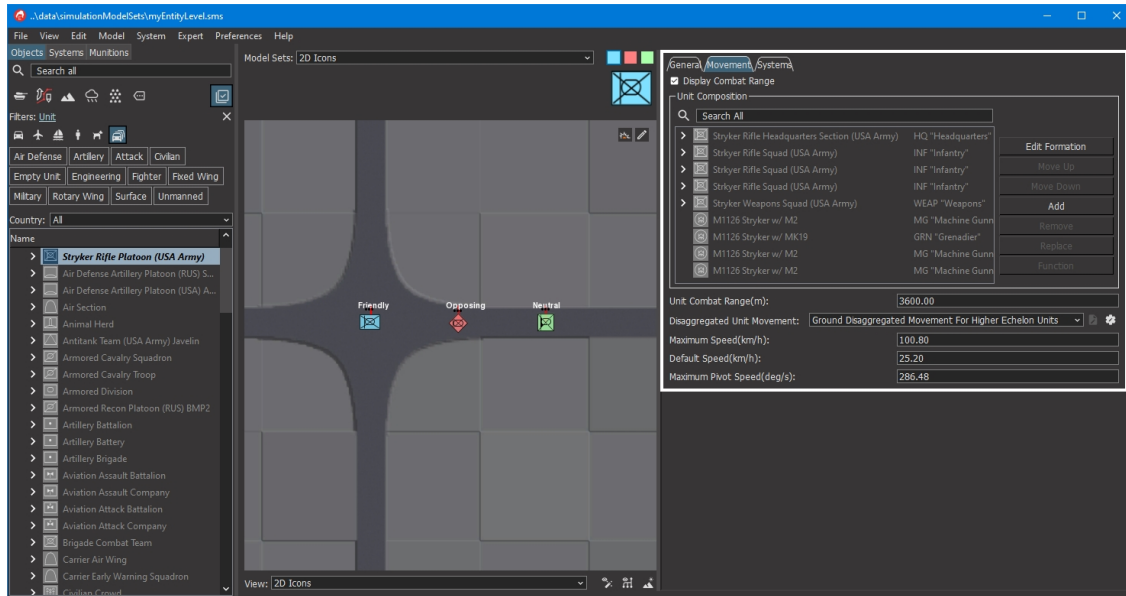
5.4. Editing Unit Composition

Many units available in VR-Forces have predefined sets of subordinates, which you can modify. (Some units are configured as "empty" units — they have no subordinates. These empty units are still available to the list of choices in the Aggregate As dialog box and for mapping to units when you import MSDL scenarios.)

i A few units (and groups that are managed like units) do not allow aggregation — that is, you cannot combine the subordinates into a single object. These include Crowds, Convoys, Carrier Air Wings, and Animal Herds.

Figure 51 illustrates the Simulation Object Editor with a unit selected. The Unit Composition group box is highlighted.

Figure 51. Unit Composition

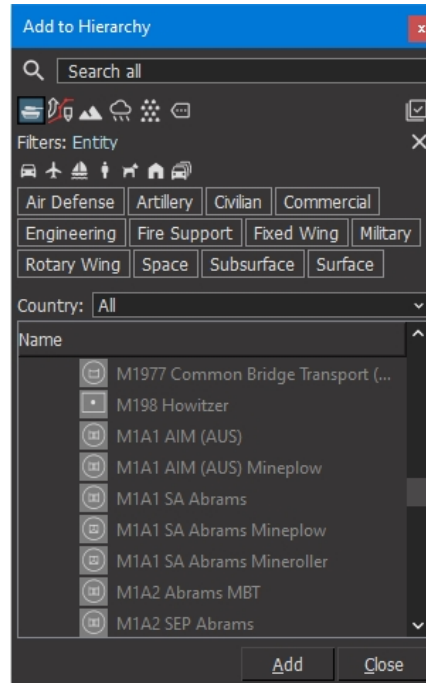


5.4.1 Adding Subordinates

When you create a new unit without copying an existing unit, you must specify its subordinates. You can also edit subordinates for existing units.

► **To add a subordinate to a unit:**

1. In the Objects List, select the unit.
2. Select the Movement tab.
3. In the Unit Composition group box (Figure 51 above), click Add. The Add to Hierarchy dialog box opens (Figure 52).

Figure 52. Add to Hierarchy dialog box

4. Optionally, click the Entity filter list.
5. In the list of simulation objects, select the simulation object that you want to add.
6. Click Add. The simulation object is added to the list in the Unit Composition window. It is also added to each formation using a default layout algorithm.
7. Continue to add as many subordinates as desired.
8. Click Close.

5.4.2 Removing Subordinates

► **To remove a subordinate from a unit:**

1. In the Unit Composition window (Figure 51 on the previous page), select the subordinate that you want to remove.
2. In the Unit Composition group box, click Remove. The subordinate is removed from the unit.

5.4.3 Replacing Subordinates

Replacing a subordinate is a convenient way to remove a subordinate and insert a different subordinate in the same location in one step.

► **To replace a subordinate:**

1. In the Unit Composition window (Figure 51 on the previous page), select the subordinate that you want to replace.

2. In the Unit Composition group box, click Replace. The Add to Hierarchy dialog box opens (similar to Figure 52 on the previous page).
3. In the list of simulation objects, select the object that you want to add as a replacement.
4. Click OK. The new object replaces the selected simulation object in the Unit Composition window. It is also added to each formation in the location of the replaced simulation object.

5.4.4 Changing the Order of Subordinates

The order of subordinates in the Unit Composition window determines which entity is the leader, the order in which units are placed into formation, and the succession when units are reorganized.

► **To change a subordinate's position in the unit composition:**

1. In the Unit Composition window (Figure 51 on page 106), select the subordinate that you want to move.
2. In the Unit Composition group box, click Move Up or Move Down.

5.4.5 Editing a Subordinate Unit

Units can contain entities and other units as subordinates. When you are editing a unit, you might decide that you want to edit one of its subordinates. You can jump to the subordinate quickly without selecting it in the main list of simulation objects.

 If you edit a subordinate unit, the changes affect all uses of that subordinate, not just the one in this particular unit.

► **To edit a subordinate:**

1. In the Unit Composition window or in the Formation Editor, right-click the subordinate that you want to edit. A menu displays.
2. In the menu, choose **Edit**. The Simulation Object Editor refreshes to show the selected subordinate. (If the Formation Editor is open, you must close it before you can edit the subordinate.)

5.4.6 Defining Functions for Subordinates

You can specify a subordinate's function within the unit. These are text-based handles that identify a role or function that the subordinate performs in that unit, which are then used in unit behaviors. Some behavior tasks are included with VR-Forces, which are described in "Unit Behavior Tasks".

► **To assign a function to a subordinate:**

1. In the Unit Composition window, select the subordinate.
2. Click Function. The Assign Subordinate Function dialog box opens.

3. Select the function you want to assign to the subordinate.
4. Click OK.

Advanced developers can use the VR-Forces behavior engine to write their own behavior trees and make use of functions; see "Unit-Specific Extensions to Standard Behavior Trees" for details. You can define a function for a behavior you are scripting if it does not fit into one of the available functions.

► **To define a function:**

1. Choose **Edit > Subordinate Functions**.
2. Click the Add button. A row is added to the bottom of the list.
3. Select the Handle field. Enter an abbreviation for the function.
4. Select the Display Text field. Enter the display text for the function.
5. The Model Set specifies the location of the subordinate functions for the SMS you are working in. You cannot edit this field.
6. Click OK.

5.5. Editing Formations

Each unit in *EntityLevel.sms* has the following defined formations:

- Column
- Line
- Vee
- Wedge

 The Convoy unit provided with VR-Forces is an exception to the discussion of units in this section. It does not have any formations because there is no preconceived membership in a convoy.

You can add, remove, and rename formations. You can also:

- Change the layout of subordinates in a formation.
- Create new formations.
- Specify the default formation.

The Formation Editor allows you to change the display of formations:

- Expand and collapse the display of subordinate units.
- Zoom in and out on the display.
- Display subordinate bounding boxes.

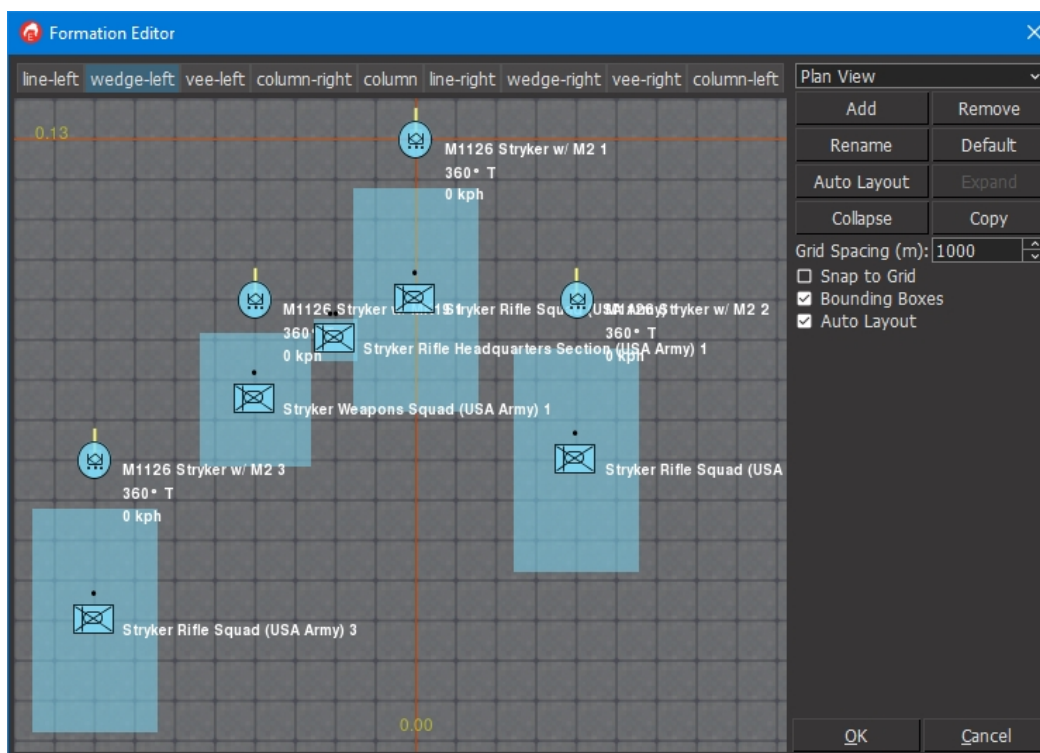
You can also edit formations in formation files. For details, see "13.1. Configuring Formations" on page 260.

5.5.1 Adding Formations

► To add a formation:

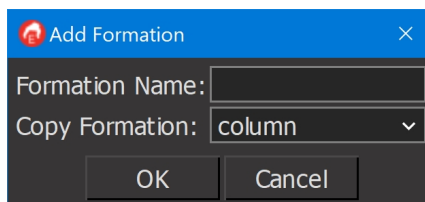
1. In the Unit Composition group box (Figure 51 on page 106), click Edit Formation. The Formation Editor opens (Figure 53).

Figure 53. Formation Editor



2. In the Formation Editor, click Add. The Add Formation dialog box opens (Figure 54).

Figure 54. Add Formation dialog box



3. In the Formation Name box, enter a name for the formation.
4. In the Copy Formation list, select an existing formation to copy the new formation from.
5. Click OK.
6. Edit the layout of the formation as described in "5.5.9 Changing a Formation's Layout by Hand" on page 113.

5.5.2 Removing Formations

► **To remove a formation:**

1. In the Unit Composition group box (Figure 51 on page 106), click Edit Formation. The Formation Editor opens (Figure 53 on the previous page).
2. In the Formation Editor, select the tab for the formation that you want to remove.
3. Click Remove. The tab is removed.

5.5.3 Renaming Formations

► **To rename a formation:**

1. In the Unit Composition group box (Figure 51 on page 106), click Edit Formation. The Formation Editor opens (Figure 53 on the previous page).
2. In the Formation Editor, select the tab for the formation that you want to rename.
3. Click Rename. The New Name dialog box opens.
4. Enter a name for the formation.
5. Click OK.

5.5.4 Specifying the Default Formation

The default formation is the formation assigned to a unit when you create it in VR-Forces. The default formation is the leftmost tab in the Formation Editor.

► **To specify the default formation:**

1. In the Unit Composition group box (Figure 51 on page 106), click Edit Formation. The Formation Editor opens (Figure 53 on the previous page).
2. In the Formation Editor, select the tab of the formation that you want to specify as the default formation.
3. Click Default. The selected tab moves to the leftmost position.

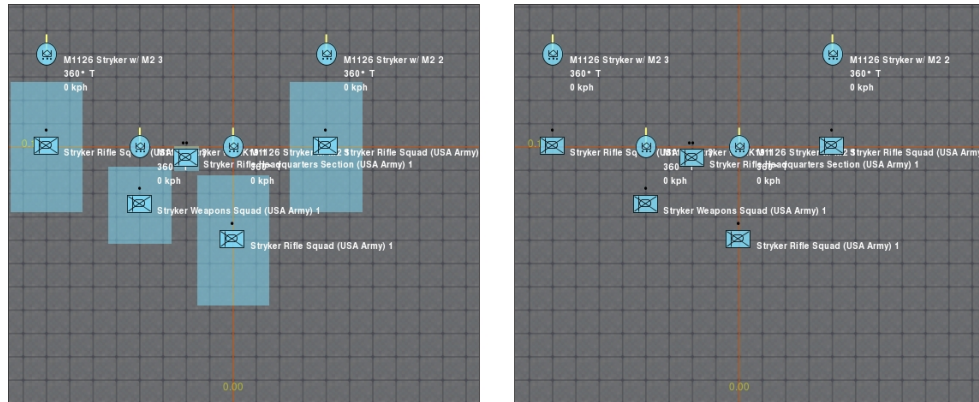
5.5.5 Expanding and Collapsing the Display of Formations

You can expand and collapse the display of formations in the Formation Editor window. This enables you to see how simulation objects are dispersed over the terrain. It can help you quickly identify any unintended variations in subordinate formations.

- To expand or collapse the display of formations in the Formation Editor, click the Expand or Collapse buttons.

5.5.6 Displaying Bounding Boxes

You can display bounding boxes for the subordinate units in a unit. This may be useful when you collapse the views of individual units, but you still want to have a sense for the area that they occupy. Figure 55 illustrates a unit with the display of bounding boxes off and on.

Figure 55. Unit bounding boxes in Formation Editor - off and on

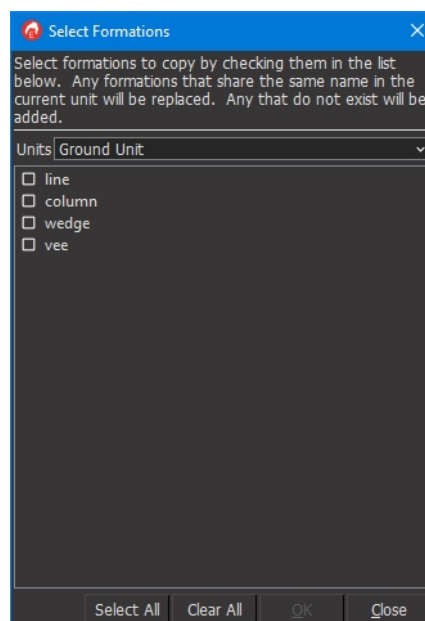
- To display or hide bounding boxes, in the Formation Editor (Figure 53 on page 110), select or clear the Bounding Boxes check box.

5.5.7 Copying Formations

You can copy formations from one unit to another. When you copy formations, the subordinates in the unit are laid out in each formation according to their order in the Unit Composition window.

► To copy a formation:

1. In the object list, select the unit that you want to copy formations to.
2. In the Unit Composition group box, click Edit Formation. The Formation Editor opens (Figure 53 on page 110).
3. In the Formation Editor, click Copy. The Select Formations dialog box opens (Figure 56).

Figure 56. Select Formations dialog box

4. In the Units list, select the unit whose formations you want to copy.
5. In the window, select the formations that you want to copy.
6. Click OK. The formations are copied from the selected unit to the currently displayed unit.

5.5.8 Laying Out a Formation Automatically

You can automatically distribute the simulation objects in a unit into one of the built-in formations. You can specify the distance between each entity in the formation.

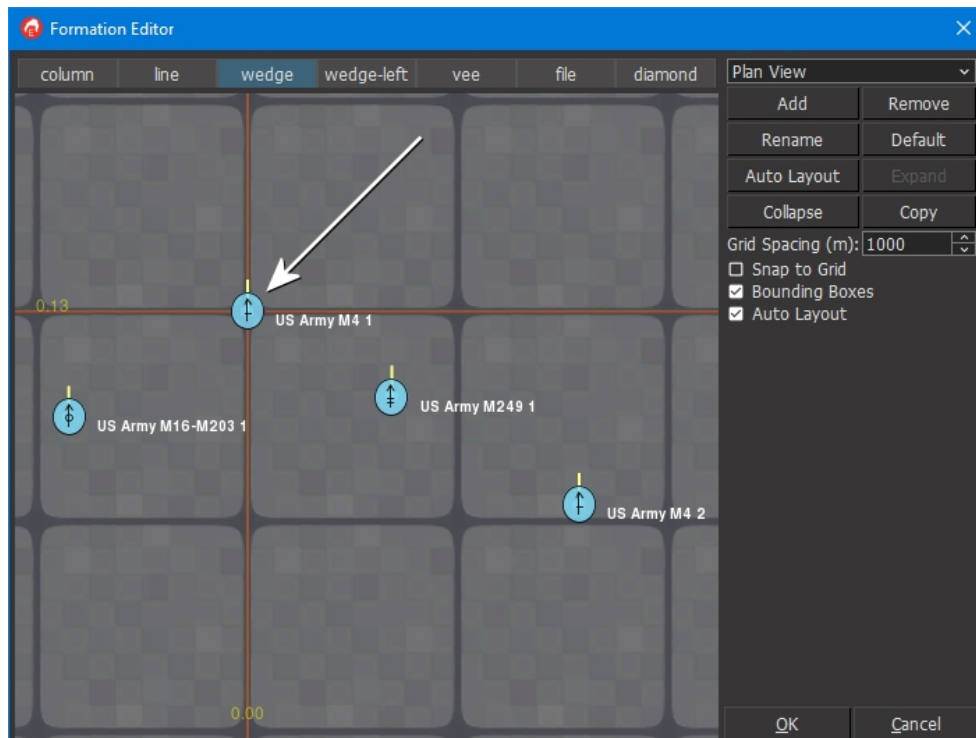
► **To lay out a formation automatically:**

1. In the Unit Composition group box, click Edit Formation. The Formation Editor opens (Figure 53 on page 110).
2. In the Formation Editor, select the tab for the formation that you want to lay out.
3. Click Auto Layout. The Auto Layout dialog box opens.
4. In the list, select a formation type.
5. Specify the distance, in meters, between each entity in the formation.
6. Click OK. The simulation objects in the unit are reorganized to the new layout.

5.5.9 Changing a Formation's Layout by Hand

You can lay out a formation by hand. The gridline in the Formation Editor indicates the center of the formation (Figure 57). This is where the unit icon is placed when the unit is collapsed.

Figure 57. Formation Editor cross-hairs



► **To change a formation's layout:**

1. In the Unit Composition group box, click Edit Formation. The Formation Editor opens (Figure 53 on page 110).
2. In the Formation Editor, select the tab for the formation that you want to lay out.
3. To move a subordinate, double-click it and then move it to the location where you want it to be.
4. Click again to place it in the new location.

Snapping Subordinates to the Grid

When you lay out a formation, you can specify that simulation objects snap to the grid. That is, they can only be placed at the intersection of vertical and horizontal grid lines. When snap-to-grid is enabled, if you want to place simulation objects more closely together than the grid allows, you can change the scale factor.

- To enable snap-to-grid for entity placement, in the Formation Editor, select Snap To Grid.

Changing the Grid Spacing

Grid spacing determines the distance, in meters, between grid lines. The default is 1000 meters.

- To change the grid spacing, enter a value in the Grid Spacing box.



6. Editing Aggregate-Level Objects

This chapter explains how to edit simulation objects and combat engineering objects for aggregate-level modeling.

6.1. Creating Objects for Aggregate-Level Modeling	116
6.2. Configuring Simulation Objects for Aggregate-Level Modeling	116
6.2.1 Aggregating Units into Higher Echelon Units	117
6.3. Configuring Ammunition, Weapons, and Equipment	118
6.3.1 Defining Ammunition, Weapons, and Equipment Items	120
6.4. Assemblies	121
6.4.1 Creating Assemblies	122
6.4.2 Copying an Assembly to Your SMS	122
6.4.3 Editing Assemblies	123
6.4.4 Creating an Assembly from a Unit	124
6.4.5 Adding Assemblies to a Unit	124
6.4.6 Removing Assemblies	125
6.5. Rolling Up Assemblies	125
6.5.1 Creating Roll Up Rules	125
6.6. Adding Activities	126
6.7. Object Parameters for Aggregate-Level Modeling	127
6.7.1 General Parameters	128
6.7.2 Physical Parameters	128
6.7.3 Movement Parameters	129
6.7.4 Sensor Parameters	131
6.7.5 Attack Parameters	132
6.7.6 Defense Parameters	133
6.7.7 Combat Parameters	134
6.7.8 Electronic Warfare (EW) Parameters	134
6.7.9 Personnel and Equipment Parameters	135
6.7.10 Supplies Parameters	135
6.8. Configuring Combat Engineering Objects	136

6.1. Creating Objects for Aggregate-Level Modeling

You create new aggregate-level simulation objects and combat engineering objects the same way you create entity-level objects, by creating a new model or creating an object from an existing model.

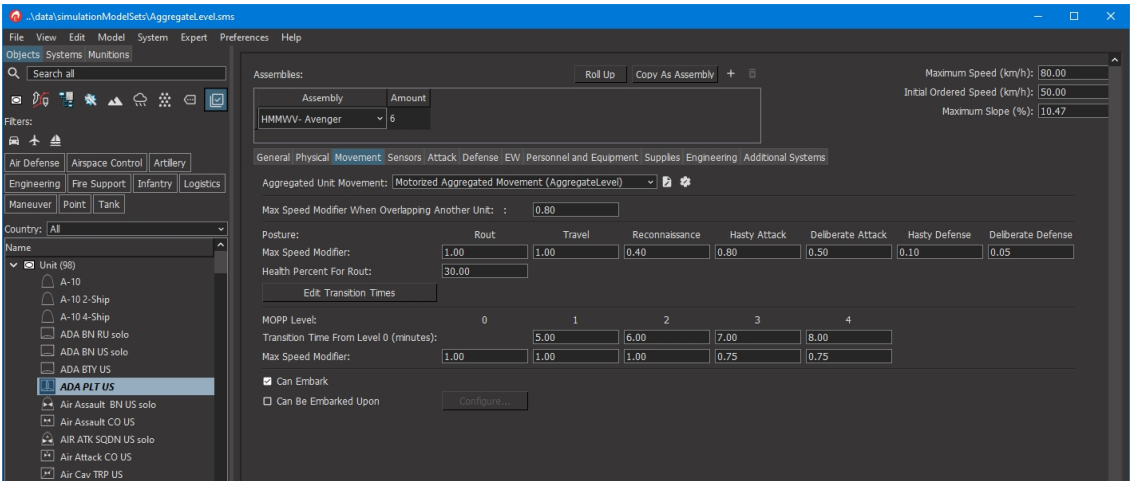
Simulation objects in an aggregate-level scenario have a more complex set of parameters than simulation objects in an entity-level scenario. Ensuring consistency among similar types of simulation objects could become difficult if, after creating new models, you had to configure all of the parameters from scratch, or if, after creating a model from an existing model, you had to back out all of the assemblies and parameters that you wanted to change. Therefore, VR-Forces includes templates for common entity types. When you create a new model from a template (by selecting a template and then choosing **Model > New Model From Existing (Copy)** or **New Model From Existing (Inherit)**), you know that you will have a simulation object with similar characteristics to other simulation objects of its type, and that is ready to customize with assemblies or direct specification of equipment, weapons, and personnel.

See "3.3. Creating a New Simulation Object" on page 57 to learn more about creating new simulation objects.

6.2. Configuring Simulation Objects for Aggregate-Level Modeling

Simulation objects for aggregate-level modeling (those used in the aggregate warfare model) are configured in the Simulation Object Editor in aggregate-level SMSs like *AggregateLevelBase.sms* and *AggregateLevel.sms*. Some aspects of their configuration, such as name, enumeration, and tags, are the same as for simulation objects in an entity-level scenario. Other parameters, the focus of this chapter, are used in the aggregate warfare model (Figure 58).

Figure 58. Unit in *AggregateLevel.sms*



The parameters for simulation objects in aggregate-level SMSs are organized into these groups:

- General. Basic identification information for the unit.
- Physical. Overall health of the simulation object, its footprint, and sector size for each posture.
- Movement. The movement system to use and how speed is modified by different postures and MOPP levels. (Does not apply to air units.)
- Sensors. Sensor systems and signatures (similar to simulation objects in an entity-level scenario) and how signatures are modified by different postures and MOPP levels.
- Attack. Specifies combat power, ammunition use, weapon systems, and modifiers to combat power. (Does not apply to air units.)
- Defense. Specifies vulnerability to different types of attack and modifiers to vulnerability. (Does not apply to air units.)
- Combat. For air units only. Specifies weapon systems, combat systems, loadout, and other attributes.
- EW. Specifies electronic warfare systems, EW defense, and dependence.
- Personnel and Equipment. Lists personnel and equipment. This information is not used by the sim. engine in simulations. It is reported as part of the simulation object's state information.
- Supplies. Specifies the supplies a simulation object needs to move, support personnel, and carry out combat engineering operations.
- Engineering. Lets you specify engineering systems. (Does not apply to air units.)
- Additional Systems. Similar to simulation objects in an entity-level scenario, this tab lets you add other systems such as spot report systems.

The individual parameters in each group are described in "6.7. Object Parameters for Aggregate-Level Modeling" on page 127.

To configure a simulation object, you specify values for as many of the parameters as are appropriate for the object type. However, you cannot configure ammunition, weapons, and equipment (aircraft, ships, and vehicles), without first defining them.

6.2.1 Aggregating Units into Higher Echelon Units

You can aggregate simulation objects into higher echelon units. And just as with simulation objects in entity-level scenarios, you specify that a simulation object can be listed in the Aggregate As list by selecting the Allow as Aggregate As Option check box. However, the aggregate-level simulation objects that are configured to use the aggregate warfare model do not have the configuration options for specifying subordinates. Therefore, *AggregateLevelBase.sms* has a platform, Aggregate Container, for use when aggregating simulation objects. The SMS has several simulation objects configured with that platform for use when aggregating simulation objects. If you want to create additional hierarchical units for aggregation, it is recommended that you use the Aggregate Container platform.

6.3. Configuring Ammunition, Weapons, and Equipment

To engage in combat, simulation objects need appropriate amounts of ammunition, weapons, and equipment. When you create a simulation object in the Simulation Object Editor, you specify the amounts that it should have by selecting them from lists of the available types of these resources. The items in these lists are also configured in the Simulation Object Editor. You can configure the ammunition, weapons, and equipment for simulation objects directly for each entity, or you can simplify the process by creating assemblies.

For example, a U.S. armored cavalry troop (AR CAV TRP US) has nine M1A2s, thirteen M3A3s, and two M1064 vehicles. An M1A2 has a 120 mm gun, a 12.7 mm machine gun, and two 7.62 mm machine guns. An M3A3 has two TOW missiles, a 25 mm gun, and a 7.62 mm machine gun. An M1064 has a 120 mm mortar and a 12.7 mm machine gun. Therefore, the entity as a whole has twenty-six TOW missiles, thirteen 25 mm guns, nine 120 mm guns, eleven 12.7 mm machine guns, thirty-one 7.62 mm machine guns, and two 120 mm mortars (Figure 59).

Figure 59. Anatomy of a unit without using assemblies



To configure this unit directly, you would go to the Personnel and Equipment tab in the Simulation Object Editor, select all these weapons in the Weapons list, and then enter the amounts for each. You also need to add up all the ammunition required by these weapons and enter them on the Attack tab.

That configuration method works well for simple units and individual entities. If you had to calculate weapons and ammunition at this level of granularity for all units, however, it could become a tedious and error-prone activity. There is an alternative to directly configuring all ammunition, weapons, and equipment individually: you can create assemblies. Assemblies are resources such as complete weapons and vehicles, or even small units, such as platoons. They are a convenient way to package ammunition, weapons, and equipment to more easily add them to units.

Each assembly configures the ammunition and weapons it needs for one instance of the resource, such as an M1A2. When you create a unit, you simply give it the assemblies it needs. Their capabilities are "rolled up" to define the total amount of resources for the entity.

Figure 60 illustrates the anatomy of an AR CAV TRP US entity that uses assemblies to represent M1A2, M3A3, and M1064 vehicles. Each assembly specifies its weapons and ammunition. When you configure the AR CAV TRP US, you simply specify that it has nine M1A2, thirteen M3A3, and two M1064 assemblies. You roll up the assemblies and the Simulation Object Editor calculates the number of each type of weapon needed (Figure 61).

Figure 60. Anatomy of a unit using assemblies

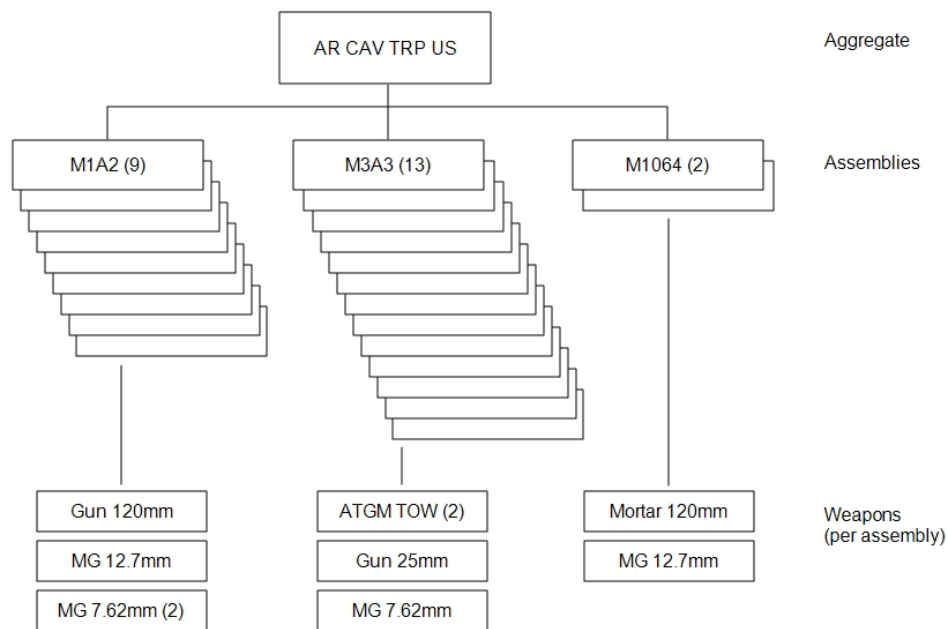


Figure 61. AR CAV TRP US weapons rolled up on Personnel and Equipment tab

General Physical Movement Sensors Attack Defense EW Personnel and Equipment Supplies Engineering Addition			
Personnel-Officers:	5		
Personnel-WOs:	0		
Personnel-NCOs:	30		
Personnel-Enlisted:	27		
Equipment:			
Equipment	Amount	Pacing/Tracking	Category
M1A2	9	Pacing	Ground Vehicle Category
M3A3	13	Pacing	Ground Vehicle Category
M1064	2	Tracking	Ground Vehicle Category
Weapons:			
Weapon	Amount	Pacing/Tracking	Category
Gun, 120mm	9	None	Weapon Category
MG, 12.7mm	11	None	Weapon Category
MG, 7.62mm	31	None	Weapon Category
Launcher, TOW	26	None	Weapon Category
Gun, 25mm auto	13	None	Weapon Category
Mortar, 120mm	2	None	Weapon Category

Using assemblies, the process for configuring units is to:

1. Define the weapons, equipment, and ammunition that units can use.
2. Define assemblies.
3. Create a unit.
4. Specify the unit's assemblies.
5. Roll up the assemblies.
6. Configure all other parameters.

For more information about assemblies, see "6.4. Assemblies" on the facing page.

6.3.1 Defining Ammunition, Weapons, and Equipment Items

Each ammunition, weapon, and equipment item that you want to be available for use by simulation objects is defined as a key/value pair. The key can be any text string you want to use. There is no relationship to any underlying VR-Forces code. The value is a Display Name – the string used in lists when creating assemblies or adding resources directly to a simulation object.

For each ammunition key/value pair you also must specify a type, where the types are:

- Anti-air
- Anti-personnel
- Anti-ship
- Anti-tank
- Biological
- Chemical
- High explosive
- Nuclear
- For weapon system

Use the anti-air, anti-personnel, anti-tank, and anti-ship categories for ammunition that will be used by assemblies or added directly to a simulation object. Use For Weapon System for any type of ammunition that will be used by a weapon system that must be added to a simulation object, such as a missile system, a guided bomb, or indirect fire system.

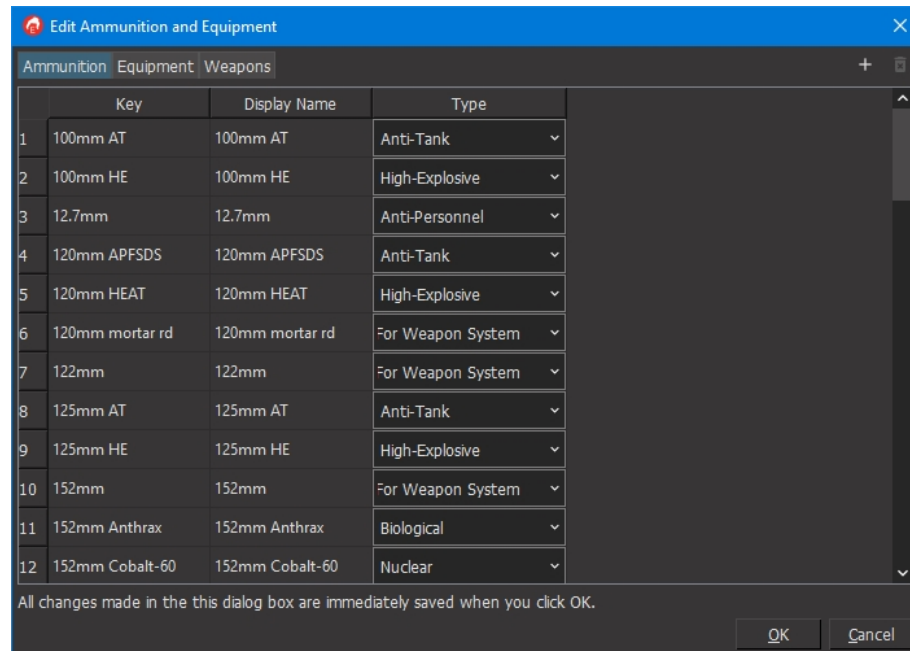
For each weapon and equipment key/value pair, you also must specify a category, where the categories are aircraft, ships, vehicles, and weapons.

The ammunition, weapons, and equipment defined in *AggregateLevelBase.sms* are representative of the weapons and equipment in common use by the United States and Russia. However, you can add any sort of resource you want. For example, you could create entries for knives, slingshots, and shovels. You can add them to assemblies or directly to a simulation object. The only real requirement for making them useful in a scenario is to assign an appropriate level of combat strength when you add them to a simulation object.

► To define an ammunition, equipment, or weapon item:

1. Choose **Edit > Ammunition and Equipment**. The Edit Ammunition and Equipment dialog box opens (Figure 62).

Figure 62. Edit Ammunition and Equipment dialog box



2. Select the tab for the type of resource you want to add.
3. Click the Add button (+). A row is added to the list. The Key cell is editable.
4. Enter the key value.
5. Click Tab or double-click the Display Name cell to make it editable.
6. Enter the text you want to be visible in the lists where this item will be included.
7. For ammunition items, select an option in the Type list.
For equipment items, select the appropriate category.
For weapon items, select Weapon Category.
8. Click OK.

6.4. Assemblies

Assemblies are a convenient way to provide a simulation object with a set of resources without individually configuring each resource.

At the aggregate level, a unit does not model individual entities. It represents some number of vehicles and personnel. For example, a mechanized infantry company has personnel, armored personnel carriers, and so on, which carry a variety of weapons and ammunition. Rather than explicitly configuring a unit to have, for example, 20 APCs, 20,000 rounds of ammunition, 10 howitzers, and 200 infantry, you can configure the ammunition and weapons for one APC as an assembly. Then you would assign a unit 20 APC assemblies. You would not need to tally all the resources associated with an APC.

This becomes even more beneficial when multiple units use an assembly. For example, the Scout Hvy PLT US unit uses M3A3 assemblies. The Tank CO US uses M1A2 assemblies. The AR CAV TRP US uses M1A2s, M3A3s, and M1064 assemblies. While it might be a manageable process to configure M1A2s for one unit and M3A3s for another unit, it becomes tedious and error-prone to use them in multiple units and have to calculate the sum of all of their weapons and ammunition each time. Using assemblies makes it easy to assign multiple units of each type to a unit.

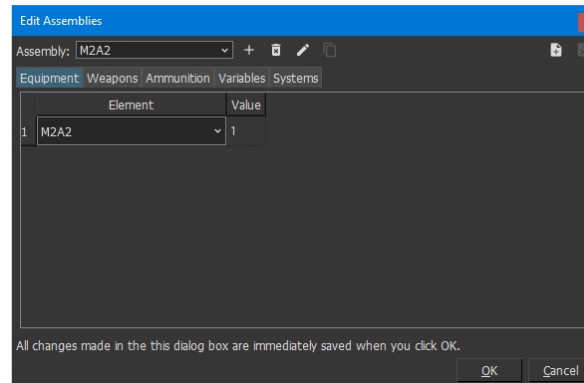
6.4.1 Creating Assemblies

When you create an assembly, you specify the equipment, weapons, ammunition, variables, and systems that are configured for a simulation object when you add the assembly to it.

► **To create an assembly:**

1. Choose **Edit > Assemblies**. The Edit Assemblies dialog box opens (Figure 63).

Figure 63. Edit Assemblies dialog box



2. Click the Add button (+). The New Assembly dialog box opens.
3. Enter a name for the assembly.
4. Optionally, select an assembly to base the new one on.
5. Click OK. The new assembly is added to the Assembly list.
6. Edit the assembly.

6.4.2 Copying an Assembly to Your SMS

When you create a new SMS by extending or including an existing one, the assemblies from the included SMS are read-only. If you want to customize an assembly, you can copy it from the included SMS to your new SMS and make changes. When you load your SMS, the customized assembly overwrites the one from the included SMS.

► **To copy an assembly:**

1. Choose **Edit > Assemblies**. The Edit Assemblies dialog box opens (Figure 63 above).
2. In the Assembly list, select the assembly that you want to copy from the included SMS.

3. Click the Copy button (). The assembly is copied to your SMS.
4. Edit the assembly.

 If you delete an assembly that you copied, your SMS returns to using the one from the included SMS.

6.4.3 Editing Assemblies

An assembly specifies equipment, weapons, ammunition, variables, and systems. Variables represent entity resources that do not fit into weapons, ammunition, and equipment. These include tangible resources, such as fuel, water, and personnel, and also intangible variables, such as health and combat power. If you do not specify particular variables for an assembly, you can specify them directly for a simulation object.

When you create an assembly, you can edit the properties of its systems. You can also edit system properties when you edit a simulation object. Many system properties do not change when you roll up a unit, but some do. For example, indirect fire weapons have a maximum sheaf parameter. The default maximum sheaf value for the 155 mm howitzer is 80 meters. If an artillery unit has 6 howitzers and you roll it up, the maximum sheaf value for the howitzers in that entity becomes 480.

 If you specify variables for an assembly, when you roll up assemblies, the variables in the assembly overwrite the values that have been set directly. Therefore, it is recommended that you use them consistently in all of your assemblies and that you not edit these values directly in a unit if you know that you will use assemblies that might overwrite them.

► **To edit an assembly:**

1. Choose **Edit > Assemblies**. The Edit Assemblies dialog box opens (Figure 63 on the previous page).
2. In the Assembly list, select the assembly that you want to edit.
3. For the Equipment, Weapons, Ammunition, and Variables tabs:
 - a. Click the Add New Assembly Item button (). A new row is added to the list of resources on the tab. In the Element column, it lists the first resource in the list of available resources.
 - b. From the Element list, select the resource you want to add to this assembly.
 - c. Double-click the Value cell to make it editable.
 - d. Enter the amount of this resource that you want to be in this assembly.
4. For the Systems tab, add a system as described in step 3. Instead of entering a number in the Value column to complete, edit the system's properties. To edit properties:
 - a. Click the Edit Properties button (). A Properties dialog box opens.
 - b. Set the property values as desired.
5. Click OK.

6.4.4 Creating an Assembly from a Unit

In addition to creating a new assembly from the Edit Assemblies dialog box (as described in "6.4.1 Creating Assemblies" on page 122), you can use an existing unit as the basis for a new assembly. Once it exists as an assembly, you can add it to other units and edit it using the Edit Assemblies dialog box (Figure 63 on page 122). After creating the assembly, you can also copy any changes to the unit and update the assembly but that does overwrite any changes you made to the assembly itself.

► **To create or update an assembly based on a unit:**

1. Create or edit the unit as you would any aggregate-level simulation object. See "6.1. Creating Objects for Aggregate-Level Modeling" on page 116.
2. Click Copy As Assembly.
3. Click Yes to confirm that you want to create or update the new assembly.
4. Click OK in the message notifying you of the assembly keyword.
5. Click OK in the message notifying you that the process was successful. The assembly is now available in the Edit Assemblies dialog box.

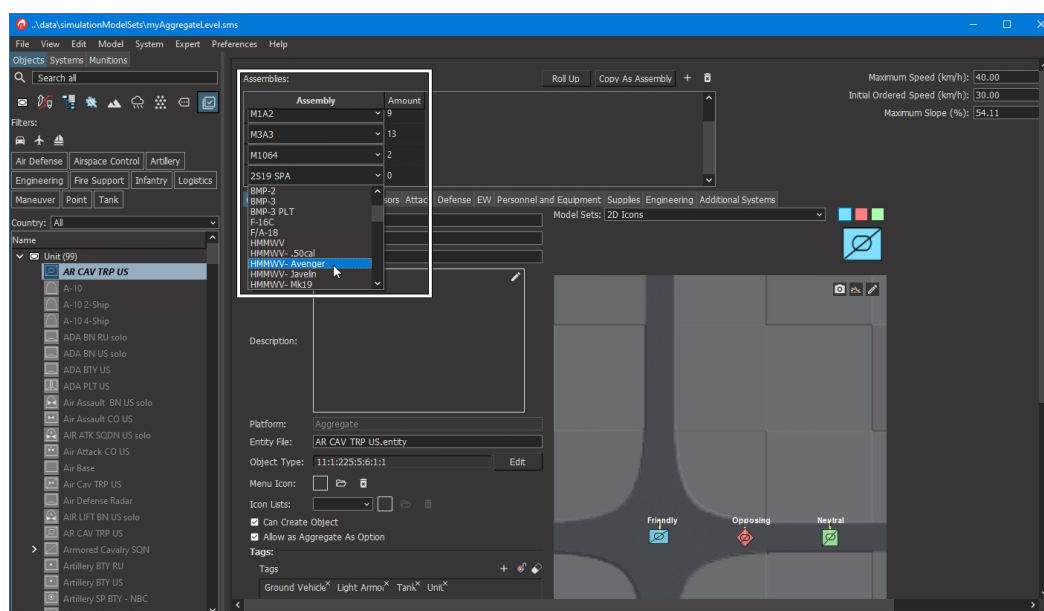
6.4.5 Adding Assemblies to a Unit

You can configure individual vehicles and weapons for a unit, or you can add assemblies, which add all the relevant parameters and combine them for the number of instances of each assembly that you add.

► **To add an assembly to a unit:**

1. Select the entity in the object list.
2. Click the Add button (+) above the Assemblies list (Figure 64). A new row is added to the list.

Figure 64. Assemblies table



3. In the Assembly column, select an assembly from the list.
4. Double-click the Amount value to make it editable.
5. Enter how many of this assembly that you want to add.

6.4.6 Removing Assemblies

► **To remove an assembly:**

1. In the Assemblies list, select the index number or any cell in the assembly you want to remove.
2. Click the Delete button ()

6.5. Rolling Up Assemblies

Once you assign a set of assemblies to a unit, you roll them up so that the full set of resources is applied to the unit. Assemblies are rolled up according to roll up rules.



When you roll up assemblies, the resources and modifiers in the assemblies overwrite the current values for the unit. If you have edited any of these resources or modifiers manually, your changes are lost. Any parameters that are not part of an assembly or roll up rules do not change.

► **To roll up assemblies for a unit:**

1. Select the unit.
2. Choose **Edit > Roll Up All** or, in the Assemblies area of the Simulation Object Editor window, click Roll Up. You are prompted to confirm the command.
3. Click Yes.

6.5.1 Creating Roll Up Rules

VR-Forces rolls up assemblies based on a set of rules. Roll up rules are not specific to simulation objects; they are global for an SMS. A roll up rule specifies the assemblies to roll up and the formula to use to roll them up:

- Sum. Add up the values for all the assemblies.
- Average. Use the average value for identical variables in the assemblies.
- Maximum. Use the highest value specified for identical variables in the assemblies.
- Minimum. Use the lowest value specified for identical variables in the assemblies.

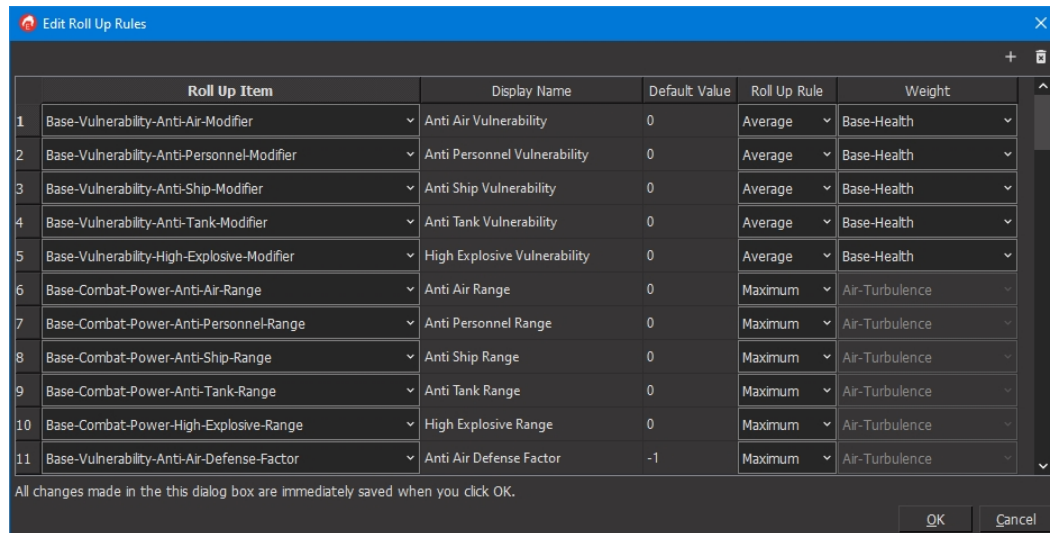


Ammunition, weapons, equipment, and systems are all rolled up by summation. There are no user-editable roll up rules for these resources.

► To create roll up rules:

1. Choose **Edit > Roll Up Rules**. The Edit Roll Up Rules dialog box opens (Figure 65).

Figure 65. Edit Roll Up Rules dialog box



2. Click the Add button (+). A new row is added to the list.
3. Select the type of roll up item from the list.
4. Specify a Display Name.
5. Specify a default value.
6. Select an option in the Roll Up Rule list.
7. If you are adding an Average roll up type rule, select an option in the Weight list. This creates a weighted average based on the item you want to give greater weight, such as health or vulnerability.
8. Add additional items as desired.
9. Click OK.

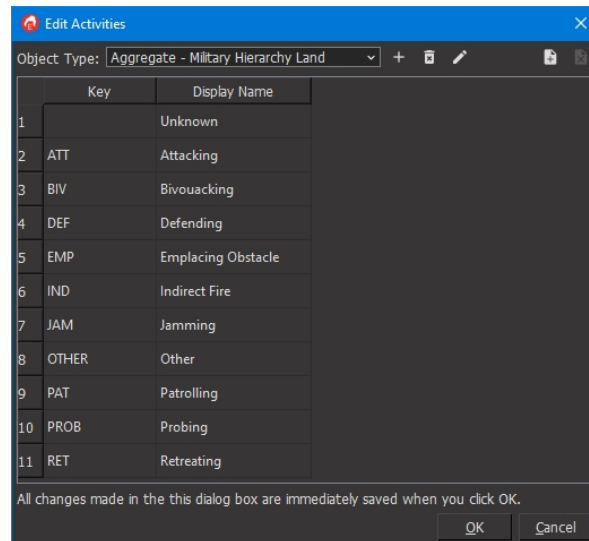
6.6. Adding Activities

You can assign simulation objects an activity. The Simulation Object Editor lets you add activities. When you add an activity, it is included in the list of activities for the Activity set data request. However, unless you add code or a Lua script that takes advantage of the new activity setting, assigning an activity to a simulation object will have no effect.

► **To add an activity to the Activities list:**

1. Choose **Edit > Activities**. The Edit Activities dialog box opens (Figure 66).

Figure 66. Edit Activities dialog box



2. Select an object type in the Object Type list, or click the Add button (+) to add a new type.
3. Click the Add Activity button (+). A new entry is added to the list of activities. The Key cell is active for editing.
4. Enter a key value for the activity. This is the value that the code or Lua script uses to identify the activity.
5. Press **Tab** or double-click in the Display Name cell to make it editable.
6. Enter the text to display in the Activities dialog box.
7. Click OK.

6.7. Object Parameters for Aggregate-Level Modeling

The aggregate warfare model is data driven. It relies on the comparative strengths of opponents given their resources and movement posture. The data used is from an extensive set of parameters that are configured for each simulation object in the Simulation Object Editor.

Most parameters are in one of these categories:

- Real-world absolute values, such as liters of diesel fuel, number of personnel, and amount of equipment and ammunition.
- Modifiers. Modifiers change the base values for the various indicated conditions, such as posture, MOPP level, or footprint. All modifiers are multipliers. For example, speed modifiers are applied to the maximum speed. If the maximum speed for a simulation object is 20 km/hr and the modifier for overlapping another unit's footprint is .9, the entity will not be able to go faster than 18 km/hr while it

overlaps another unit.

- Systems. Movement system, sensor systems, other systems.
- Abstract values. Strength and health are abstract values. They represent the relative ability of simulation objects to damage other simulation objects and absorb damage.

For the simulation objects in aggregate-level SMSs provided with VR-Forces, real-world values are based on publicly available information about the types of simulation objects. The various modifiers applied to simulation object behavior are derived from customer requirements for the aggregate warfare model. Modifier values have been set from experimentation to provide what we believe to be realistic results. Values for health and strength have also been developed on a relative scale to obtain reasonable results.

If you add new simulation objects to an aggregate-level SMS, it is important that you try to set health and strength to values that fit with the existing simulation objects. If you create an entirely new SMS for aggregate-level scenarios, you are responsible for developing your own consistent way to represent the relative strengths of the simulation objects.

Parameters are organized in a set of tabbed pages. The following sections describe the parameters on each tab. The Additional Systems tab does not include aggregate warfare parameters. Use it to specify systems that did not logically belong on the other tabs.

6.7.1 General Parameters

Use the General tab to specify identifying information for a unit, such as its name, enumeration, tags, and model.

6.7.2 Physical Parameters

Physical parameters describe a simulation object's footprint and how it faces the external world for each posture, which is a simulation object's overall mode of interaction with its environment. Air units and ground units have different parameters.

Table 4. Physical parameters for ground units

Parameter	Description
Health	A quantification of the overall health of the entity.
Physical Footprint	The area notionally occupied by the simulation object. The footprint can vary for different postures based on the Footprint Modifier.
Height	Height of the physical footprint.
Footprint Modifier	The amount by which the footprint changes for the indicated posture.
Front Sector Size	The number of degrees, centered on the heading, within which forward-facing sensors and interactions take place.
Rear Sector Size	The number of degrees, centered on the heading -180°, within which rear-facing sensors and interactions take place.
Flank Sector Size	The number of degrees, centered on the heading +/- 90°, within which side-facing sensors and interactions take place.

Table 5. Physical parameters for air units

Parameter	Description
Aggregated Unit Movement	The movement system used by the unit.
Can Embark	Specifies whether the unit can embark on another unit.
Can Be Embarked Upon	Specifies whether other units can embark on this unit.
Health	The health value for this unit.
Physical Footprint	The area notionally occupied by the simulation object.
Mass	The weight of the unit.
Runway Length Required for Takeoff	The runway length this unit needs to take off.
Runway Length Required for Landing	The runway length this unit needs to land.
Runway Length Required for Landing with Drag Chute	The runway length required if this unit is landing with a drag chute deployed.
Has Drag Chute by Default	Whether this unit type should be equipped with a drag chute when it is created.

6.7.3 Movement Parameters

A unit has a movement system. You can specify the movement system on this tab and edit its parameters. You can also specify posture transition times.

 Aircraft units do not have a Movement tab.

Movement can be affected by:

- Environmental factors. These properties are specified per movement system.
- Posture. A simulation object's posture can affect its speed.
- MOPP level transition time. The amount of time it takes for a unit to change from MOPP level 0 to one of the other MOPP levels.

The Movement tab also has settings for embarkation. These settings are the same as for simulation objects in an entity-level scenario. For details, see "Chapter 7. Configuring Embarkation" on page 139.

Table 6 describes the movement parameters.

Table 6. Movement parameters

Parameter	Description
Max Speed Modifier When Overlapping Another Unit	The amount to modify a simulation object's maximum speed when its footprint overlaps another simulation object's footprint.
Max Speed Modifier (Posture)	The amount to modify the maximum speed based on the posture.
Health Percent For Rout	The amount to modify health when in the rout posture.
Transition Time from Level 0	The amount of time it takes to transition from MOPP Level 0 to a higher MOPP level.
Max Speed Modifier (MOPP Level)	The amount to modify the maximum speed based on the MOPP Level.

Editing Movement System Properties

Table 7 describes movement system properties. Systems do not all have the same properties.

Table 7. Movement system properties

Property	Description
Rain Modifier by Intensity	The amount speed is affected by the intensity of rain. The lower the modifier value, the more slowly the simulation object moves as rain intensity increases.
Snow Modifier by Intensity	The amount speed is affected by the intensity of snow. The lower the modifier value, the more slowly the simulation object moves as snow intensity increases.
Visibility Degrades Max Speed When Below	If visibility drops below this value, maximum speed is reduced.
Maximum Amount Visibility Can Degrade Speed	The maximum amount that speed decreases as visibility decreases. Higher values decrease speed more.
Wind Degrades Max Speed When Above	If wind speed exceeds this value, the maximum speed of the entity is reduced. (Aircraft only.)
Wind Degrades Until	Degrade speed due to wind until the wind speed reaches this value. Then degrade speed at the maximum rate. (Aircraft only.)
Maximum Amount Wind Can Degrade Speed	The maximum amount that wind can degrade speed. (Aircraft only.)
Retrograde Movement Penalty	Speed is reduced by this amount when moving in retrograde. (Ground movement only.)

► **To edit movement system properties:**

1. Click the Edit Properties button (🔧). The Edit Properties dialog box opens.
2. Edit the values for the various properties.
3. Click OK.

Editing Posture Transition Times

VR-Forces models the amount of time it takes for a unit to transition from one posture to another. You can specify the transition times for each movement system.

► **To edit posture transition times:**

1. Select the Movement tab.
2. Click Edit Transition Times. The Edit Transition Times dialog box opens (Figure 67).

Figure 67. Edit Transition Times dialog box

	Rout	Travel	Reconnaissance	Hasty Attack	Deliberate Attack	Hasty Defense	Deliberate Defense
To Rout Posture Transition Time	0.00	0.00	0.00	0.00	0.00	0.00	0.00
To Travel Posture Transition Time(Minutes):	30.00		30.00	30.00	60.00	30.00	60.00
To Reconnaissance Posture Transition Time(Minutes):	60.00	60.00		60.00	60.00	60.00	60.00
To Hasty Attack Posture Transition Time(Minutes):	30.00	5.00	5.00		5.00	5.00	15.00
To Deliberate Attack Posture Transition Time(Minutes):	120.00	120.00	120.00	120.00		120.00	120.00
To Hasty Defense Posture Transition Time(Minutes):	90.00	90.00	90.00	90.00	90.00		90.00
To Deliberate Defense Posture Transition Time(Minutes):	180.00	180.00	180.00	180.00	180.00	90.00	

3. Change the times, in minutes, as desired. Changes take effect immediately.
4. Click Done.

6.7.4 Sensor Parameters

Use the Sensors tab to specify the sensors that a simulation object has. You can add and remove sensor systems and set their properties using the standard Simulation Object Editor procedures. (For details, see "4.4. Editing a Simulation Object's Systems" on page 88.)

You can specify the simulation object's sensor signatures by choosing from a preconfigured set of signatures or by defining a custom set. The Sensor Signatures parameters specify the distance at which other simulation objects can sense the simulation object you are editing using the different types of sensors. The preconfigured signatures make it easy to consistently apply the same set of signatures to similar object types.

As with movement, a unit's posture and MOPP level affect its ability to use its sensors. You can specify signature modifiers for each posture and MOPP Level. (Posture and MOPP level modifiers do not apply to air units.)

6.7.5 Attack Parameters

Simulation objects have combat power in five domains: Anti-Air, Anti-Tank, High Explosive, Anti-Personnel, and Anti-Ship. The Attack tab specifies the parameters that determine a unit's ability to attack an opponent in these domains. It allows you to add weapon systems to the simulation object.

 Aircraft units do not have an Attack tab.

Table 8. Attack parameters

Parameter	Description
Range	The distance at which a simulation object can attack in each domain.
Hit Factor	When compared to the defense factor for a target simulation object, determines the probability of the attack hitting the target.
Strength	The simulation object's strength in each type of combat. This is an abstract value.
Ammunition Usage	The number of rounds per minute used for each type of combat. This value is apportioned among ammunition of the same type. For example, if the ammunition usage for Anti-Personnel combat is 1000 rounds per minute and a simulation object has 12.7 mm ammunition and 7.62 mm ammunition, both of which are type Anti-Personnel, then the simulation object uses 500 rounds per minute of each type of ammunition.
Ammunition Auto-Resupply Rate	The amount of ammunition that is automatically resupplied each day if auto-resupply is enabled for this simulation object.
Ammunition	The types of ammunition this simulation object has, the amount of each type, and whether pacing or tracking is enabled.
Distribution	The proportion of casualty types from an attack.
Sector Combat Power Modifiers	The amount the base combat power is modified depending on which sector is being attacked.
Posture Combat Power Modifiers	The effect on combat power of each posture.
Combat Range Modifiers	The effect on the range at which a simulation object can attack based on its posture.
CID Level Combat Power Modifiers	The effect on combat power of the enemy detection and classification level.
MOPP Level Combat Power Modifiers	The effect on combat power of the MOPP Level.
If Morale is Less Than	A percentage of morale below which combat power is affected.

Table 8. Attack parameters (continued)

Parameter	Description
Adjust Combat Power By	The combat power modifier to apply if morale is less than the specified amount.
Environmental Conditions Combat Power Modifier	The combat power modifier to apply if precipitation intensity is at its maximum or if there is no illumination.
Weapon System Auto-Resupply Rate	The number of rounds per day supplied to each weapon system if auto-resupply is enabled.
Weapon System Attack Interval	The amount of time between each weapon system attack by this unit.

6.7.6 Defense Parameters

Use the Defense tab to specify modifiers to a simulation object's vulnerability. Vulnerability is specified within each weapon domain.

 Aircraft units do not have a Defense tab.

Table 9. Defense parameters

Parameter	Description
Vulnerability	The amount that a simulation object is vulnerable to the different types of attack.
Defense Factor for Guided Munitions	The ability to defend against guided munitions.
NBC Vulnerability	Vulnerability to nuclear, biological, or chemical contamination.
Posture Vulnerability Modifier	The effect of different postures on the simulation object's ability to defend itself.
Sector Vulnerability Modifier	The ability of the simulation object to defend itself to the front, rear, and flanks.
MOPP Level Vulnerability Modifier	The ability of the simulation object to defend against NBC contamination at different MOPP Levels.
Health Percent for Firepower Kill	The level of health below which this unit will not be able to fire.
Health Percent for Mobility Kill	The level of health below which this unit will not be able to move.
Health Percent for Sensor Kill	The level of health below which this unit will not be able to sensor other objects.

6.7.7 Combat Parameters

Air units have a Combat tab. This tab lets you configure weapon systems, combat systems, loadout, and health parameters.

Table 10. Combat parameters

Parameter	Description
Weapon Systems	Add, remove, and edit properties for the weapon systems this unit has.
Combat Systems	Add, remove, and edit properties for other combat-related systems.
Loadout	The type of loadout this unit uses. You can specify a default by clicking the star, and then lists showing the loadouts indicate the default with a star. (Loadouts are specific to each unit. There is no library of preconfigured loadouts available to select from.)
Loadout Items	The specific munitions in the loadout.
Distribution	The percent chance that a casualty caused by this unit will be of a specific casualty type.
Defense Factor for Guided Munitions	The defense factor to apply when this unit is struck by a guided munition.
Weapon System Attack Interval	The amount of time between each weapon system attack by this unit.
Health Percent for Firepower Kill	The level of health below which this unit will not be able to fire.
Health Percent for Mobility Kill	The level of health below which this unit will not be able to move.
Health Percent for Sensor Kill	The level of health below which this unit will not be able to sensor other objects.

6.7.8 Electronic Warfare (EW) Parameters

The EW tab lets you specify a simulation object's ability to defend against electronic warfare and how much it depends on communications. You can add EW systems.

6.7.9 Personnel and Equipment Parameters

Use the Personnel and Equipment tab to specify the order of battle for the simulation object. These values are not used by the sim. engine in the simulation. They are for reporting purposes. The configured personnel and equipment amounts are multiplied by the simulation object's health to arrive at the current amount.

Table 11. Personnel and Equipment parameters

Parameter	Description
Personnel Officers	Number of officers.
Personnel WOs	Number of warrant officers.
Personnel NCOs	Number of non-commissioned officers.
Personnel Enlisted	Number of enlisted persons.
Equipment	List of equipment types, the number of each, and whether it is flagged for pacing, tracking, or none.
Weapons	List of weapon types, the number of each, and whether it is flagged for pacing, tracking, or none. (Ground units only.)

6.7.10 Supplies Parameters

For ground units, the Supplies tab allows you to specify the amount, use per day, use per kilometer, pacing/tracking, and auto-resupply rate for these supplies:

- Food.
- Water.
- Oil.
- Lubricant.
- Motor gas.
- Aviation fuel.
- Diesel fuel.
- Other Supplies. You can add any other type of supply that you want. The list of items in the Supply drop-down list is drawn from `./data/simulationModelSets/AggregateLevel/gui/rwKeywordValuePairs.xml`. (For information about this file, see "6.9. Editing Reader/Writer Key/Value Pairs" on page 137.) You can enter in any other supply type that you want to.

The supplies added here are only used by engineering objects. To use a supply, you configure it here and in the engineering object. For example, a combat engineering company has other supplies called Construction-Material and Explosives.

This tab also lets you specify the conditions under which a simulation object can receive supplies.

For air units, you can specify the amount of aviation fuel, the aerial refueling system, and other supplies.

6.8. Configuring Combat Engineering Objects

Combat engineering objects (CEOs) can affect mobility of simulation objects and can cause attrition. Therefore, their simulation models include the information needed for these effects. Since CEOs can be constructed by units, their models include parameters that specify the types of materials needed to create them and sizing information.

The actual process of configuring CEOs is similar to that of other objects. Table 12 describes the parameters specific to CEOs. Some parameters do not apply to all CEOs.

Table 12. Combat engineering object parameters

Parameter	Description
Short Name	The name displayed on the map and sent as marking text over the network.
Height	The height of the object.
Length	Maximum length of single constructed objects. If a longer object is specified, it is created as several objects.
Width	The width of a line object.
Minimum Completion Percentage for Effect	The minimum completion percentage before the object affects simulation objects.
Minimum Completion Percentage to Detect	The minimum completion percentage before simulation objects can detect the object.
Attrition	The rate of attrition to an affected simulation object in each domain.
Mobility Modifiers	The amount that mobility of an affected simulation object is modified in each type of movement.
Vulnerability Modifiers	The amount that an affected simulation object's vulnerability is modified in each domain.
Signature Modifiers	For object types that can be affected by this object, the amount that each type of sensor signature is affected.
Resource Required	The type of resource needed to construct this object.
Resource Amount	The amount of the required resource needed to build this object.

Table 12. Combat engineering object parameters (continued)

Parameter	Description
Is Concealed	If selected, the object is detected based on a random probability in the detecting sensor. If cleared, the object is always detected by the appropriate sensor.
Maximum Relative Size	For areal objects, the maximum size of a simulation object, relative to the object, for it to be affected by the object. For example, if the maximum relative size for a fortified area is .75, a simulation object must be 75% of the size of the area or smaller for it to be protected.

6.9. Editing Reader/Writer Key/Value Pairs

Some display text is not automatically included in the *untranslated.ts* file provided with VR-Forces. The Simulation Object Editor lets you map some internal keywords to display text so that you can translate it. These internal keywords are called reader/writer keys and they are managed using key/value pairs. These mappings are stored in `./data/simulationModelSets/AggregateLevel/gui/rwKeywordValuePairs.xml`. If you edit this file, do not change the **paramName**; change the string mapped to it.

Besides allowing for translation, you can add new entries to this file. Table 13 describes the effect of adding new keyword:value pairs.

Table 13. Effect of editing rwKeywordValuePairs.xml parameters

Parameter	Result
Type	Added to the Type list for the Edit Ammunition and Equipment dialog box, Ammunition tab, in the Simulation Object Editor.
Category	Added to the Category list for the Edit Ammunition and Equipment dialog box, Weapon tab and Equipment tab, in the Simulation Object Editor.
Posture	Added to the list of Postures in the Posture set data request. However, selecting such a posture would not have any effect unless complementary code is written to update the simulation engine.
Resource-Item	Added to the Supply list for configuring Other Supplies on the Supplies tab for simulation objects in the Simulation Object Editor.



7. Configuring Embarkation

This chapter explains how to configure entities for embarkation. It includes a tutorial for configuring embarkation slots.

7.1. Enabling Embarkation	140
7.2. Adjusting the View While Configuring Embarkation	141
7.3. Configuring Slots	142
7.3.1 Adding Slots	142
7.3.2 Configuring Slot Parameters	144
7.3.3 Configuring Slot Exclusions	147
7.3.4 Specifying Ingress and Egress Techniques	148
7.4. Adding Ingress and Egress Points	149
7.5. Embarkation Slot Tutorial	151
7.5.1 Configure the Super Tanker for Embarkation	151
7.5.2 Configure Slots for the Super Tanker	153
7.5.3 Add Ingress Points	156
7.5.4 Add Egress Points	157
7.5.5 Save the New Configuration	158
7.5.6 Test the Embarkation Configuration	158

7.1. Enabling Embarkation

If an entity allows other entities to embark on it, you can configure:

- Allowed entity types. The types of entities that can embark.
- Slots. The location on an entity where embarked entities move to. The number of slots on an entity controls how many entities can embark on it using the Embark task. For entities that are visible when they are embarked, such as humans on a truck or aircraft on an aircraft carrier, it is important to specify sensible-looking slots.

Slots can have a type and a name. Use of slot types and names allows you to tell a simulation object to embark in a specific slot.

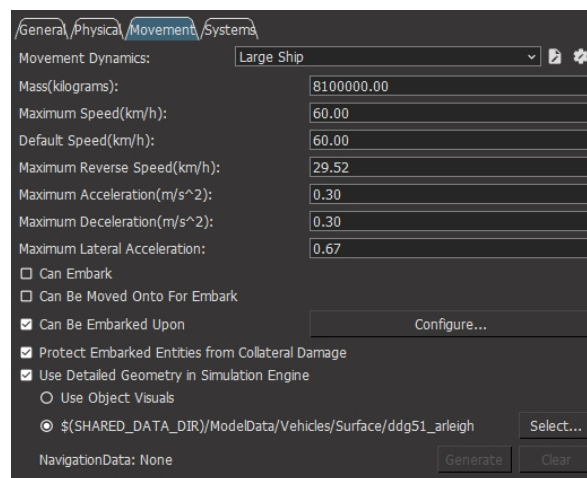
- Ingress points. The locations on an entity where entities embark before moving to a slot.
- Egress points. The locations on an entity from which entities disembark.

i Units can support embarkation, but the usual use of embarkation is for entities to embark on other entities. In entity-level scenarios, if a unit embarks, the individual members of the unit carry out the embarkation task.

► **To enable embarkation for an entity:**

1. In the Simulation Object Editor object list, select the entity that you want to configure embarkation for.
2. Select the Movement tab (Figure 68).

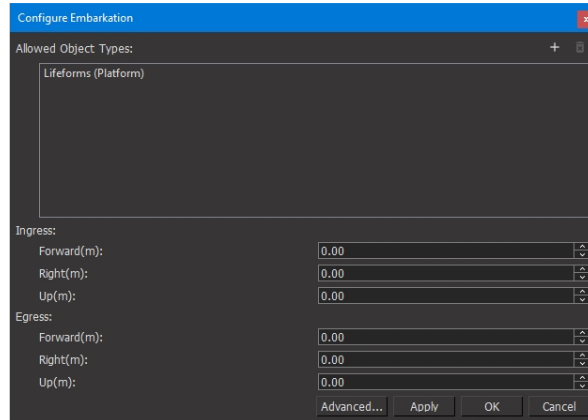
Figure 68. Movement tab



3. If Can Be Embarked Upon is not selected, select it.
4. Optionally, select Can Be Moved Onto For Embark. This option lets ground-clamped entities identify entities that they can embark on. They can then walk or drive onto the entity and automatically become embarked.

5. Optionally, select Protect Embarked Entities from Collateral Damage. If selected, embarked entities are not damaged if the parent entity sustains collateral damage.
6. Click Configure. The Configure Embarkation dialog box opens (Figure 69).

Figure 69. Configure Embarkation dialog box



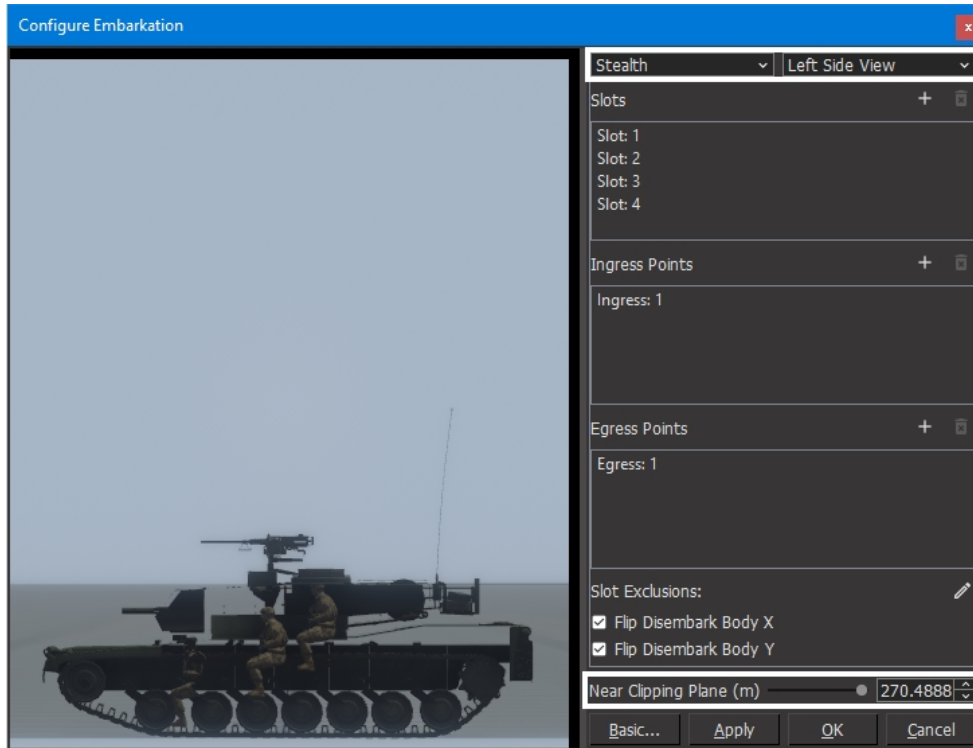
7. Click Add. The Add Type dialog box opens.
8. Select a type of entity that can embark.
9. Optionally, add additional entity types.
10. For each entity type listed, specify ingress and egress points. Values are in meters relative to the center of the parent entity. For more information, see "7.4. Adding Ingress and Egress Points" on page 149.
11. Optionally, configure slots. For more information, see "7.3. Configuring Slots" on the next page.
12. Click OK.

7.2. Adjusting the View While Configuring Embarkation

You can adjust the perspective of the Configure Embarkation dialog box, allowing you to view the entity as needed to define embarkation slots as well as ingress and egress points.

You are able to:

- Change the view mode to either Plan View (2D) or Stealth (3D).
- Select one of the views for the Stealth view mode, such as Top Down or Left Side View.
- Use the mouse or keyboard to zoom in or out or otherwise change your view of the entity.
- Adjust the Near Clipping Plane to be able to see the inside when you are defining slots within the entity.

Figure 70. M1A2 Abrams MBT with view for configuring embarkation

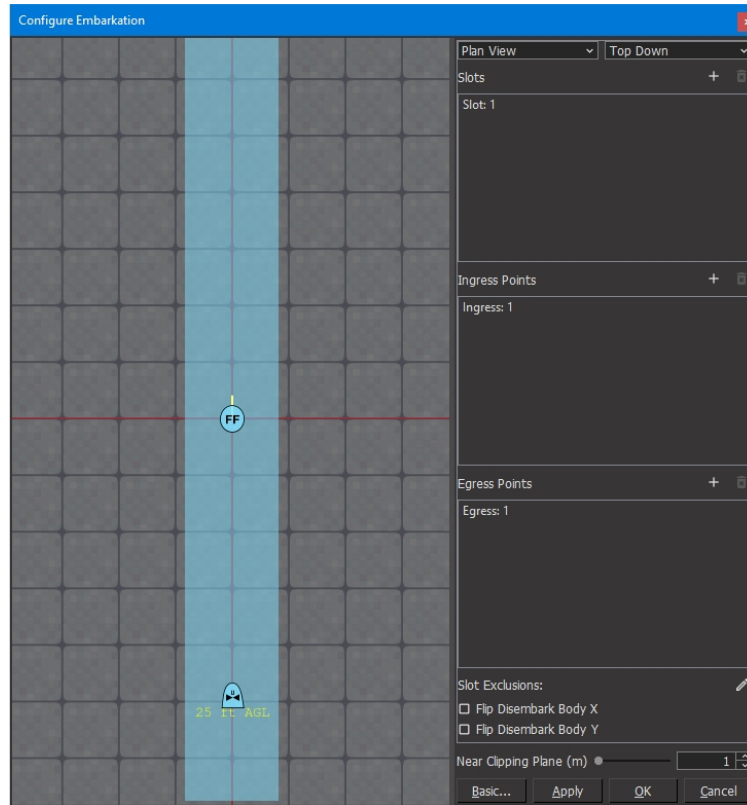
7.3. Configuring Slots

In the simple embarkation configuration process, VR-Forces adds one slot, located at the center of the parent entity. You can edit that slot. You can also add additional slots if you want embarked entities to be in specific locations on the parent.

7.3.1 Adding Slots

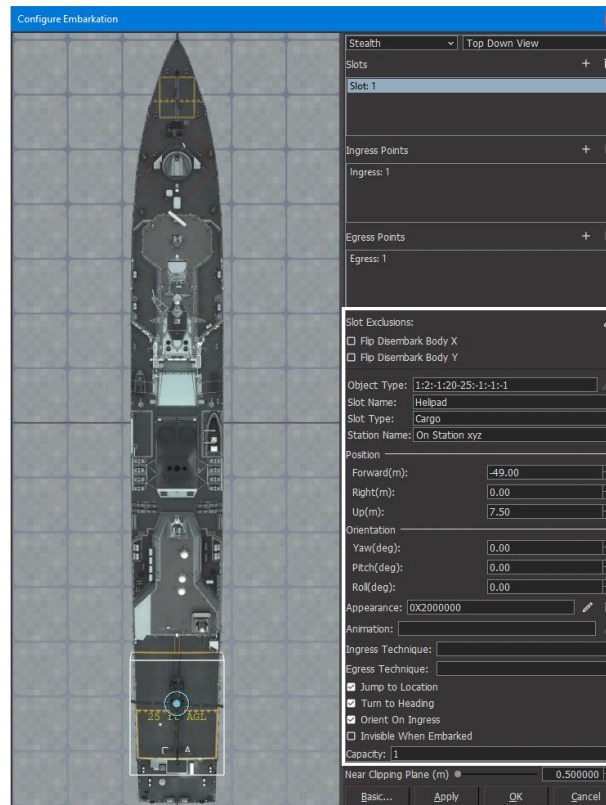
► **To add a slot:**

1. In the Simulation Object Editor object list, select the simulation object that you want to configure embarkation for.
2. Select the Movement tab (Figure 68 on page 140).
3. Select the Can Be Embarked Upon check box.
4. Click Configure. The Configure Embarkation dialog box opens (Figure 69 on the previous page).
5. If you have not already specified the simulation object types that can embark, specify a simulation object type as described in "7.1. Enabling Embarkation" on page 140.
6. Click Advanced. The dialog box redisplay. It shows a 2D view of the bounding volume of the simulation object for which you are configuring embarkation (Figure 71).

Figure 71. Configure Embarkation Advanced

7. Optionally, change the view to see the entity in a way that best allows you to see the slot positions. See "7.2. Adjusting the View While Configuring Embarkation" on page 141.
8. Click the Add button above the Slots window (Figure 71). A slot is added to the list and the dialog box redisplay to show the slot's parameters (Figure 72).

Figure 72. Slot parameters



9. Optionally, configure Slot Exclusions as described in "7.3.3 Configuring Slot Exclusions" on page 147.
10. Optionally, configure other slot parameters as described in "7.3.2 Configuring Slot Parameters" below.

7.3.2 Configuring Slot Parameters

You can specify these parameters for a slot:

- The object type that can use it. You can configure different object types in different slots on a simulation object.

For each field in the object type enumeration, you can specify more than one value. For example, the category value for a helicopter can be 20-25 for military helicopters, 50 for unmanned, and 90-92 for civilian. If you just want to allow a specific category for a given slot, specify that category. If you want to specify a range of values, separate them with a hyphen, for example, 21-24. To specify several specific categories, separate them with commas, for example, 21, 25, 50. You can also combine these options. To specify all helicopters, you could set the object type to 1:2:-1:20-25,50,90-92:-1:-1:-1.

- A name for the slot.
- A slot type.

- The DIS station name. This allows you to indicate the relative position of one entity to another by an identified position rather than an x,y,z offset.
- The position on the parent entity relative to the center of the entity's bounding volume.
- The orientation of the slot relative to the center of the entity.
- The appearance of the simulation object type when it is in the slot. For example, a human embarked on a truck might have its posture set to Sitting.
- For slots that accept humans, an animation to use while embarked, for the ingress, and for the egress. See "7.3.4 Specifying Ingress and Egress Techniques" on page 148
- Whether the embarking entity should just jump to the slot location, rather than using normal entity movement. For example, an aircraft will taxi to a slot on a carrier deck.
- Whether the embarking entity should turn (snap) to the heading of the parent when the entity starts to move to the egress point or starts to move to the embarkation point. (Some movement models do not work well when snapping to a heading. For those models, use the Orient on Ingress option, not Turn to Heading.)
- Whether the embarking entity should smoothly orient (turn) to the heading of the parent when it reaches the embarkation point.
- Whether the embarking entity should be invisible while it is embarked.
- How many simulation objects the slot can hold.

► **To specify slot parameters:**

1. To change the side of a parent simulation object that the embarked simulation objects exit on, select the Flip Disembark Body X check box, the Flip Disembark Body Y check box, or both.
2. If you have added a new slot, in the Object Type field, specify the object enumeration for the type of simulation object that can use this slot. If this is a previously created slot, you can edit the object type. You can configure different object types in different slots on a simulation object.
3. If you want to give the slot a name, enter it in the Slot Name text box.
4. If you want to categorize this slot as a specific named type (a text string, not an object type enumeration), enter it in the Slot Type text box.
5. In the Position group box, specify the location of the slot relative to the center of the entity's bounding volume.
6. In the Orientation group box, specify the orientation of the slot relative to the center of the entity.
7. Optionally, specify the appearance of the simulation object type when it is in the slot. If you want to specify an animation for a human, specify a posture that supports animations, such as Upright, Standing Still. The default is to ignore the appearance.
8. Optionally, specify an animation that the entity will use when it embarks.

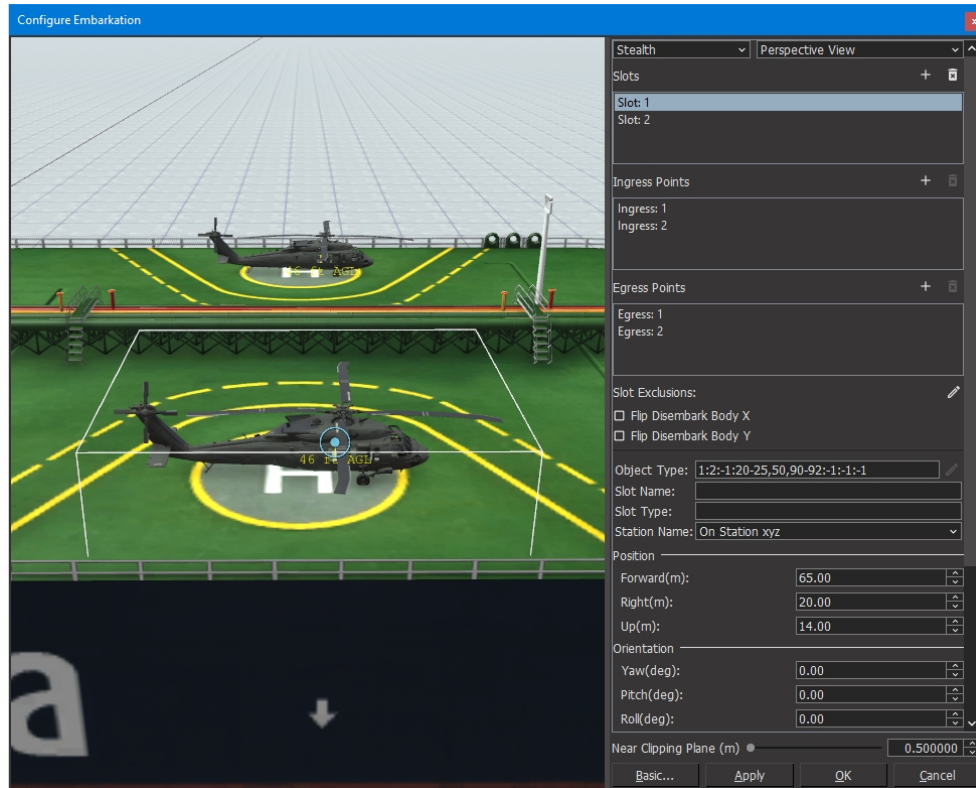
 Animations only apply to humans. The available animations depend on the entity's posture, as specified by the Appearance parameter.

9. Optionally, specify an ingress technique. For details about specifying an ingress technique, see "7.3.4 Specifying Ingress and Egress Techniques" on page 148.
10. Optionally specify an egress technique. For details about specifying an egress technique, see "7.3.4 Specifying Ingress and Egress Techniques" on page 148.
11. Optionally, for lifeforms only, if the Appearance specifies a posture that has animations, click the Edit button and select an animation.
12. If you want the embarked simulation object to immediately jump to the slot from the ingress point, select Jump To Location. If this check box is clear, the simulation object will move to the slot.
13. If you want the embarked simulation object to snap to the same heading as the parent entity, select Turn To Heading.
14. If you want the embarking entity to smoothly orient itself to the heading of the entity when it reaches the ingress point, select Orient on Ingress.
15. If you want the embarked simulation object to be invisible in 3D observer modes, select Invisible When Embarked.
16. In the Capacity text box, specify the number of entities that can use this slot. -1 means an unlimited number.

Previewing the Embarked Entity's Position

In the Stealth view of the Configure Embarkation dialog box, you can move the observer to see the position of the example embarked entity (Figure 73). This can help you position it where you want. Note that this is relative to the visual model; see "4.1.6 Adding Object Geometry to a Simulation Object" on page 78. Testing it in a scenario is the best method for seeing whether it embarks as desired, as covered in the tutorial at the end of this chapter.

Figure 73. Side view of embarked entity



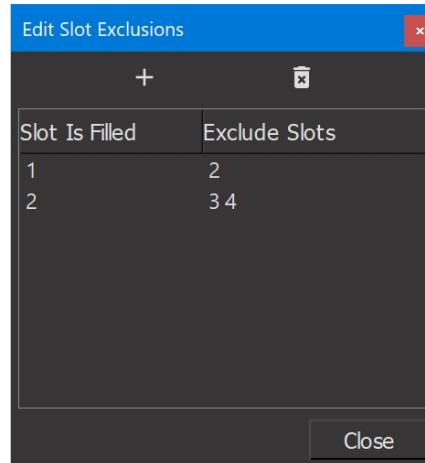
7.3.3 Configuring Slot Exclusions

A slot exclusion specifies that if a given slot is occupied, other slots cannot be occupied. For example, an entity might allow both tanks and dismounted infantry entities to embark. If no tanks are embarked, the entity can hold 20 DI-Guys. But if a tank is embarked, it uses up 10 slots and now only 10 DI-Guys can embark.

► **To configure slot exclusions:**

1. On the Slot Exclusions line, click Edit. The Edit Slot Exclusions dialog box opens (Figure 74).

Figure 74. Edit Slot Exclusions dialog box



2. Click the Add button (+). A blank line is added to the list.
3. Double-click the cell in the Slot Is Filled column. The cell becomes editable.
4. Enter the number of the slot that you want to configure.
5. Double-click the cell in the Exclude Slots column.
6. Enter the numbers of the slots that cannot accept entities if the slot you are configuring is filled. You can enter more than one slot to be excluded. Separate the slot numbers with a space.
7. Repeat this procedure to add additional exclusions. You can also edit existing exclusions and delete them.

7.3.4 Specifying Ingress and Egress Techniques

When you configure a slot, you can specify an ingress and egress technique. An ingress or egress technique associates a string with a script that runs when the entity embarks. The technique string is associated with a script ID in the OPE file for the entity type.

To specify a technique in an OPE file, add the **technique-map** block to the embarkation controller. The *Human.ope* file has this technique map:

```
(embarkation-controller
  ...
  (invisible-at-origin True)
  (technique-map
    (GetInCar "get_in_car")
    (GetOutOfCar "get_out_of_car")
  )
)
```

In this case, putting the string `GetInCar` in the Ingress parameter tells VR-Forces that when the entity embarks, it should run the script that has the script ID `get_in_car`.

VR-Forces includes two scripts, Get In Car and Get Out Of Car (in the Civilian Vehicle folder), that show how to write a script that implements an ingress or egress technique. As shown in the comments in these scripts, there is a set of standard task variables. Do not define the variables. The scripts get their values from the embarkation code.

7.4. Adding Ingress and Egress Points

An ingress point specifies where an entity embarks onto another entity. An ingress point can specify a path that the entity follows to reach the ingress point and how it moves once it is on the entity. For example, a jet embarking on an aircraft carrier can taxi to its slot.

i If you do not specify an ingress point, entities will not move towards the parent before embarking. They will teleport to the slot.

An egress point specifies where an entity disembarks from the parent entity.

You can specify that an ingress point should only support some of the slots on the entity. You might want some types of entities embarking at one ingress point and moving to certain slots, while other entities embark at a different location and use other slots. For example, if you are simulating a ferry, cars and trucks would enter at the front or rear and move below deck. Foot passengers would enter at the sides and move to the passenger decks.

When you add an allowed entity type, the Simulation Object Editor automatically creates a slot for that entity type. The ingress and egress points that you configure for that entity type on the basic version of the Configure Embarkation dialog box are automatically associated with the same slot.

In Advanced mode, you can add additional ingress and egress points and associate them with whichever slot you want. You do not have to add them in pairs.

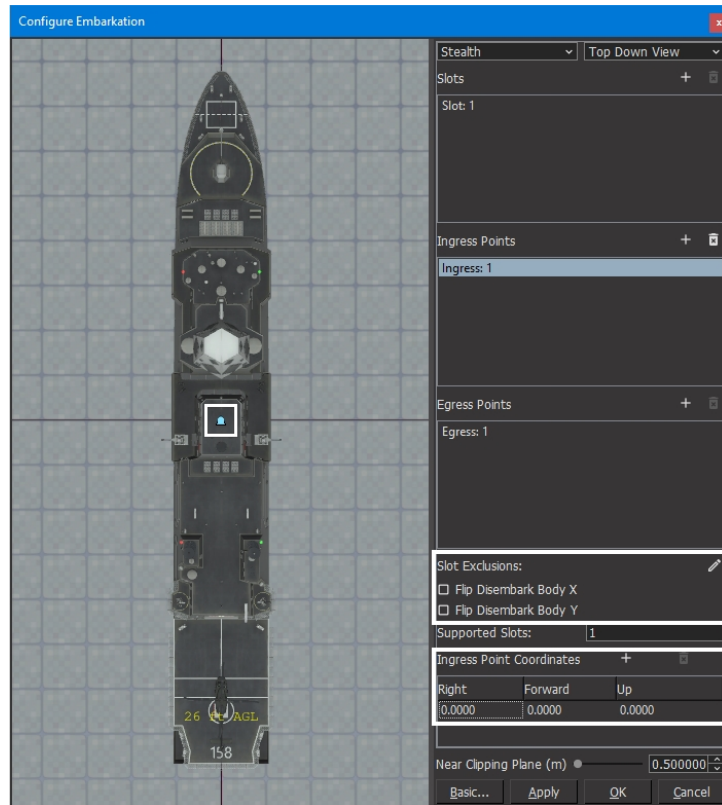
i This procedure describes configuring an ingress point. To configure an egress point, substitute "egress" for "ingress" in each step.

► To add an ingress point:

1. In the Simulation Object Editor object list, select the entity that you want to configure embarkation for.
2. Select the Movement tab (Figure 68 on page 140).
3. Select the Can Be Embarked Upon check box.
4. Click Configure. The Configure Embarkation dialog box opens (Figure 69 on page 141).
5. Click Advanced (if needed). The dialog box redisplay. It shows an entity icon in Plan View mode.
6. Optionally, change the view to see the entity in a way that best allows you to see the slot positions. See "7.2. Adjusting the View While Configuring Embarkation" on page 141.

7. Click the Add button (+) above the Ingress Points list. An ingress point is added to the list. A vertex is added to the display window at the center of the entity bounding volume (Figure 75). The coordinates for the ingress point displays in the table near the bottom of the dialog box.

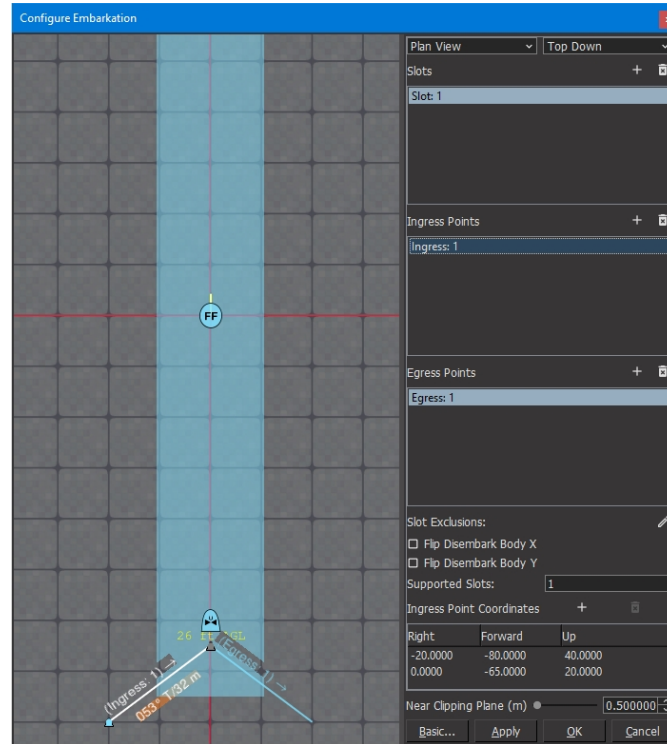
Figure 75. Ingress Point (Stealth View)



8. Double-click the Right value for the Ingress point to make it editable.
9. Enter the Right value.
10. Repeat for the Forward and Up values.
11. To specify a path for the embarking entity:
 - a. Click the Add button for the Ingress Point coordinates list. A new coordinate is added.
 - b. Specify the Right, Forward, and Up values for the new coordinate. The path displays in the window. Entities move along the points in the order they are shown in the table. In the Plan View, an arrow indicates the direction entities

will move along the path (Figure 76).

Figure 76. Ingress point configuration (Plan View)



12. Optionally, in the Supported Slots field, enter the numbers for the slots that should support this ingress point. If you leave the field blank, it supports all slots. To specify more than one slot, separate the numbers with a space.
13. To change the side of a parent simulation object that the embarked simulation objects exit on, select the Flip Disembark Body X check box, the Flip Disembark Body Y check box, or both (Figure 71 on page 143).
14. Click OK.

7.5. Embarkation Slot Tutorial

This tutorial shows how to configure an entity for embarkation. The super tanker does not have slots configured for embarkation in the default VR-Forces *EntityLevel.sms*. However, it has two helipads on the deck, which makes it ideal for embarkation. We will add embarkation slots, ingress points, and egress points to the super tanker. Then we will demonstrate the new slots in a scenario.

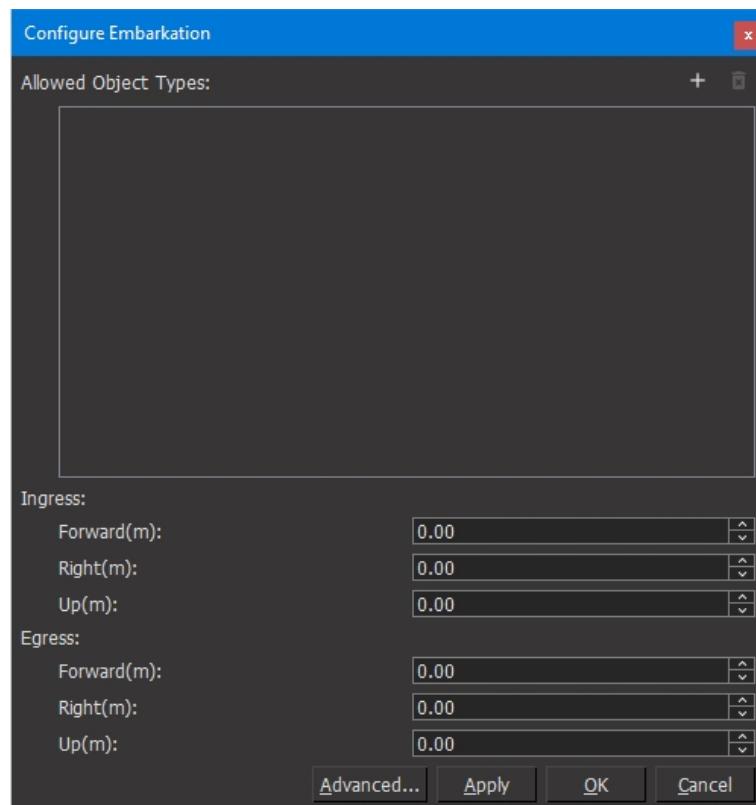
7.5.1 Configure the Super Tanker for Embarkation

► To configure the super tanker for embarkation:

1. Start the Simulation Object Editor.
2. Choose **File > New Simulation Model Set**. The New Simulation Model Set dialog box opens.

3. Select the New Simulation Model Set option.
4. Click Next. The Simulation Model Set Type dialog box opens.
5. Select the Extend Existing Simulation Model Set option.
6. Click Next. The Extend Existing Simulation Model Set dialog box opens.
7. Select *EntityLevel.sms*.
8. Click Next. The New Model Set Name dialog box opens.
9. Enter a name for the new SMS.
10. Click Finish. The new SMS is created and loaded.
11. In the list of entities, select Super Tanker. It is read-only because it is included from *EntityLevel.sms*.
12. At the top of the Simulation Object Editor window, click Promote. You can now edit the Super Tanker.
13. Select the Movement tab (Figure 68 on page 140).
14. Select the Can Be Embarked Upon check box.
15. Click Configure. The Configure Embarkation dialog box opens (Figure 77).

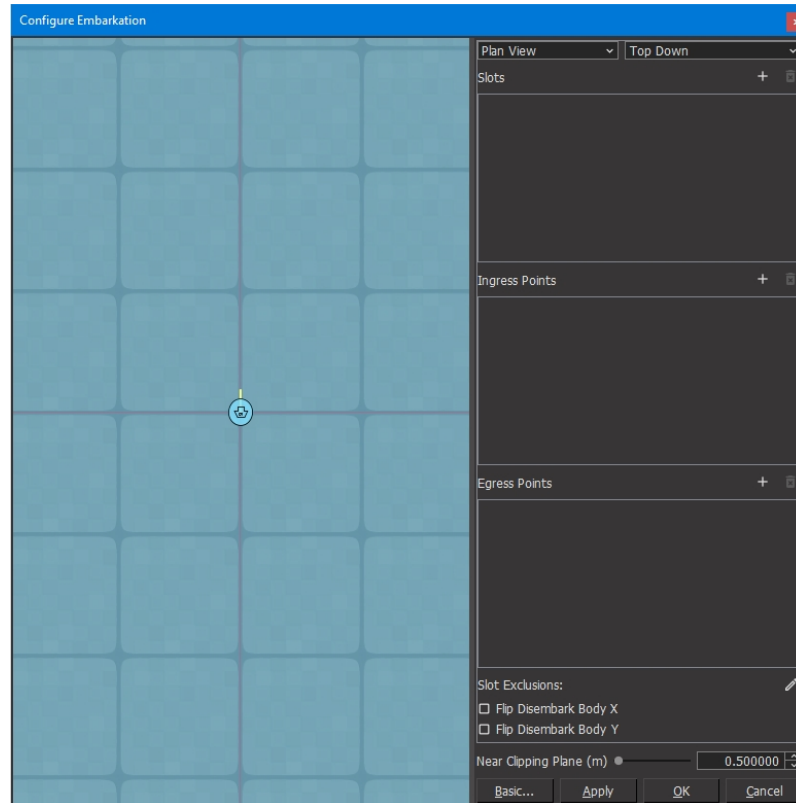
Figure 77. Configure Embarkation dialog box



16. Click Add (+). The Add Type dialog box opens.

17. In the list of entity types, select Rotary Wings (Platform) and then click OK. Rotary Wings (Platform) is listed in the Allowed Object Types window.
18. Click Advanced. The Configure Embarkation dialog box redisplay (Figure 78). We are now ready to configure slots.

Figure 78. Configure Embarkation 2D



7.5.2 Configure Slots for the Super Tanker

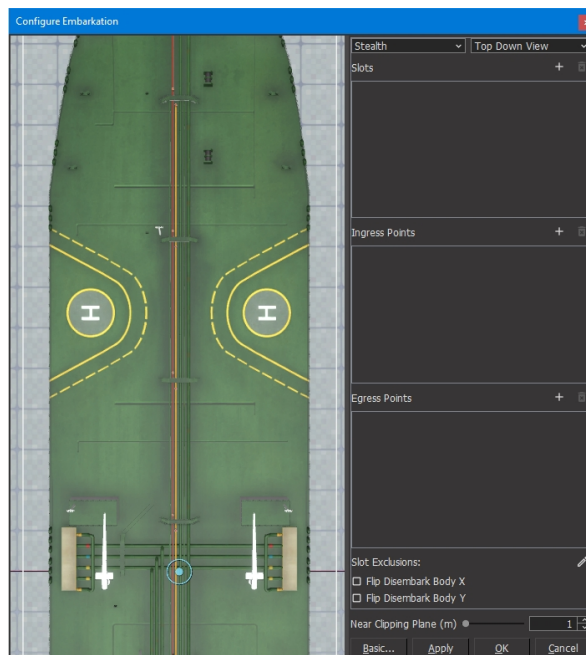
We want to configure specific slots for the super tanker. We cannot do that in the basic Configure Embarkation dialog box. The key parts of configuring a slot are specifying the object type and the location. Other parameters can provide more specific behaviors.

Add a Slot

► To add a slot for the super tanker:

1. In the list at the upper right of the dialog box, change the observer view from Plan View to Stealth. You now see a view of the super tanker model.
2. From the view list at the upper right, select Top Down View.
3. Zoom in so that you can see the forward half of the super tanker. In particular, you want to see the two helipads (Figure 79).

Figure 79. Stealth view



4. Click the Add button (+) above the Slots window. Slot 1 is added to the list.
5. Select Slot 1.

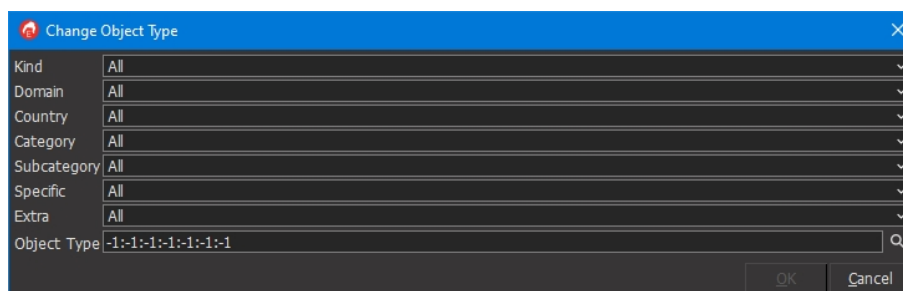
Specify the Object Type

We need to specify the object types of the entities that can use the slot. In this case, we just need to specify helicopters. However, helicopters have several object type enumerations. Helicopters can have category values of 20-25 (military), 50 (UAV), and 90-92 (civilian). You can specify multiple values in an enumeration using dashes for ranges and separating values with commas. For the helicopter, we want an enumeration that looks like this: 1:2:-1:20-25,50,90-92:-1:-1:-1. We will let the Simulation Object Editor build a basic object type. Then we will customize it.

► To: specify the object type:

1. On the Object Type line, click the Select button (✎). The Change Object Type dialog box opens.

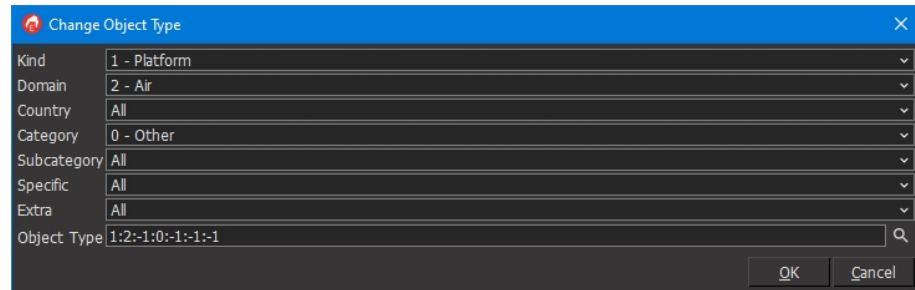
Figure 80. Change Object Type dialog box



2. In the Object Type dialog box, in the Kind list, select Platform (1).

3. In the Domain list, select Air (2).
4. In the Country list, select All (-1).
5. In the Category list, select Other (0).
6. For all the other lists, select All (-1). The results in an enumeration of 1:2:-1:0:-1:-1:-1 (Figure 81).

Figure 81. Completed base enumeration



7. Click OK to close the Change Object Type dialog box.
The enumeration is listed in the Object Type box of the Configure Embarkation dialog box.
8. In the category part of the Object Type, replace 0 with 20-25,50,90-92. The completed object type is: 1:2:-1:20-25,50,90-92:-1:-1:-1 (Figure 82). This object type allows any type of helicopter to land in this slot.

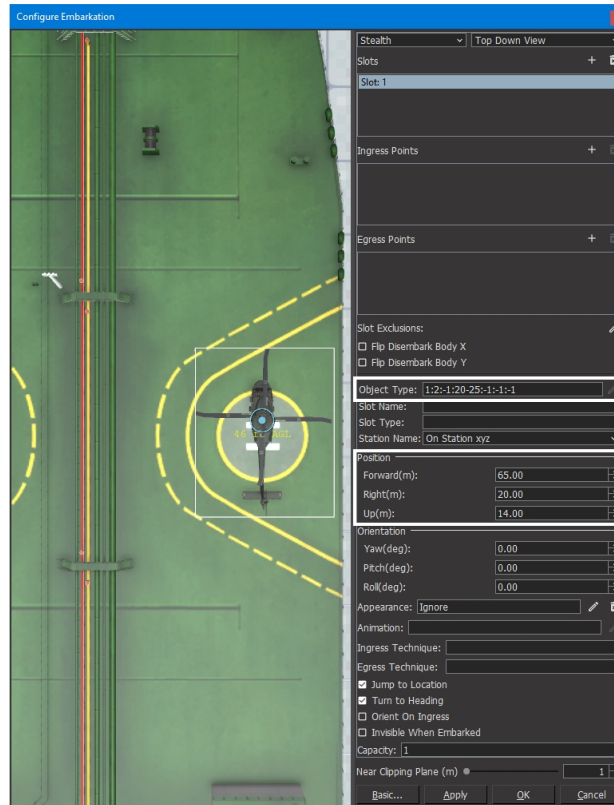
Specify the Location of the Slot

We must specify the X, Y, and Z location of the slot relative to the origin of the super tanker. You will see a blue dot and yellow position text at the center of the ship. Please note that the coordinates specified in this example are good for the super tanker.

► To specify the slot location:

1. Make sure the Station Name is set to On Station xyz. We are going to specify the slot with an x,y,z offset.
2. Take a look at the location of the helipad to the right of the center line of the super tanker (starboard). Estimate the distance from the middle of the ship to the helipad.
3. Enter your estimated distance for the Forward (Y) value. A blue target circle displays on the center line of the ship.
4. Adjust the forward value to reach the proper distance (65 is a good value).
5. For the Right (X) coordinate, enter an estimated value for the distance to the right of center (starboard). (20 is a good value.) You should now see the target circle centered on the H of the helipad.
6. Enter a value for Up (Z). If you leave this blank, the helicopter will clamp to the deck of the tanker, but first it will sink to 0 meters, which doesn't look realistic. (14 is a good value for this coordinate.) An example aircraft displays on top of the helipad.
7. Clear the Jump to Location check box. The dialog box should look like Figure 82.

Figure 82. Slot 1



8. Optionally, click Apply to save your changes.

Add a Second Slot

Add a second slot. Use the same object type and forward and up values. The Right position should be -20.

7.5.3 Add Ingress Points

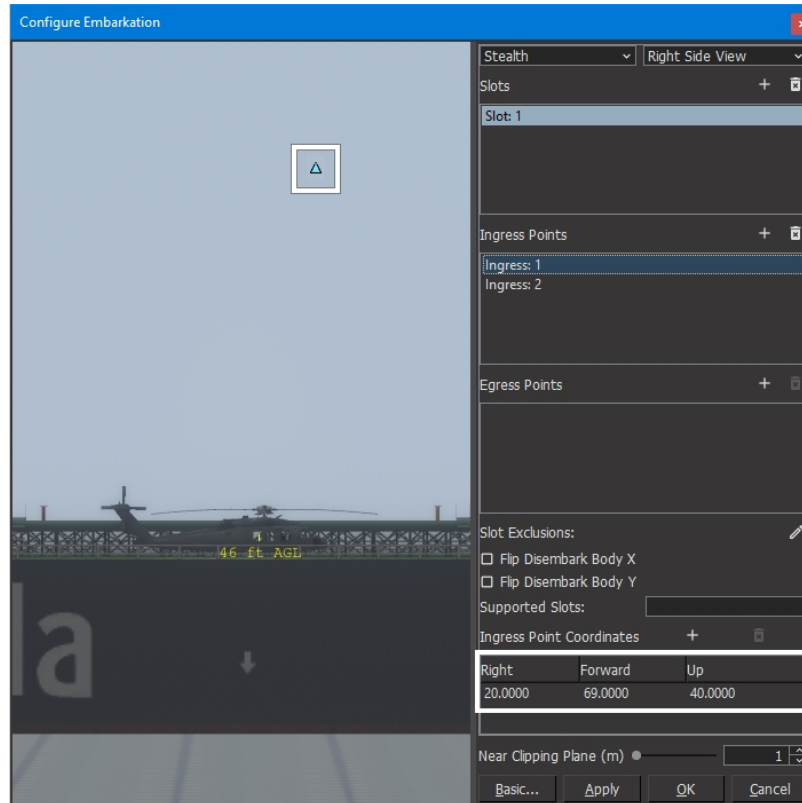
Ingress points define locations relative to the parent entity that an embarking entity moves to before it moves to its slot. While you do not have to define one, you should do so. If there are no ingress points, when you task an entity to embark, it teleports directly to a slot. It does not simulate movement towards the parent.

► To add an ingress point:

1. Click the Add button (+) above the Ingress Points window. A blue point displays at the center of the ship.
2. In the Ingress Points Coordinates table, double-click the Right value to make it editable.
3. Set it to 20 and then press Tab. The point moves to this location.
4. Set the Forward value to 69.

- Set the Up value to 40. This will cause a helicopter to fly to a point above the helipad (the ingress point) and then land on the helipad. You can see the ingress point's position above the deck in one of the side views (Figure 83).
- Add a second ingress point. Set Forward, Right, and Up to -20, 69, and 40.

Figure 83. Completed ingress points



- Optionally, click Apply to save your changes.

7.5.4 Add Egress Points

Egress points define locations relative to the parent entity that a disembarking entity moves to after it leaves its slot. As with ingress points, egress points help manage realistic looking behavior for entities as they disembark. Without them, the entity simply stays where it is in its slot, but changes its state to disembarked.

► **To add an egress point:**

- Click the Add button (+) above the Egress Points window. A red vertex marker is added to the ship.
- Set the Right, Forward, and Up values to 20, 69, and 40. This will cause a helicopter to disembark and fly to a point above the helipad before beginning its next task. The vertex marker moves to show the location of the egress point.
- Add a second egress point. Set Right, Forward, and Up to -20, 69, and 40.

7.5.5 Save the New Configuration

► To save the configuration:

1. In the Configure Embarkation dialog box, click OK.
2. In the main SOE window, choose **File > Save Simulation Model Set**.

7.5.6 Test the Embarkation Configuration

Now you can test the new embarkation configuration for the super tanker.

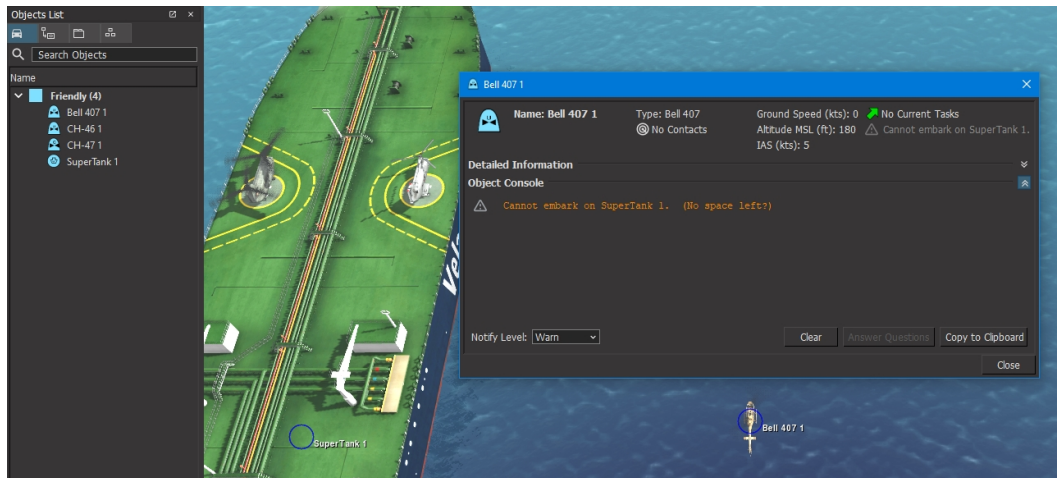
► To test the embarkation configuration:

1. Start VR-Forces.
2. Create a new scenario using the entity-level SMS you created earlier and a terrain that has an ocean area, such as Ground_DB II or Ala Moana.
3. Place a Super Tanker in the water.
4. Add three helicopters. Put one on one side of the tanker and two on the other side. Use a variety of military and civilian helicopters. Set their altitude to about 180 - 300 feet.

Figure 84. Test scenario



5. Give each helicopter a plan to embark on the super tanker. Stagger the times at which they start their embark task so that they do not all start flying at once.
6. Run the scenario. Two helicopters should be able to embark in the two slots. The third should send a message to its console that it cannot complete its task because there are no slots available (Figure 85).

Figure 85. Helicopters embarked on Super Tanker



8. Adding DI-Guy Models to VR-Forces

This chapter explains how to take DI-Guy models created with the DI-Guy SDK and integrate them into VR-Forces.

8.1. Adding New DI-Guy Models to VR-Forces	162
8.1.1 Copy DI-Guy Files to the Data Directory	162
8.2. DI-Guy Configuration Files	162
8.2.1 Adding a New DI-Guy Animation	163
8.3. How VR-Forces Maps DI-Guy Models	164
8.4. Add a New DI-Guy Entity in the SOE	164

8.1. Adding New DI-Guy Models to VR-Forces

VR-Forces supports DI-Guy characters and includes many lifeform characters. However, MAK customers sometimes want to add their own DI-Guy characters. The general process for adding a new character is:

1. Create the new character using DI-Guy software and copy the required files to the appropriate `./SharedData/##/latest` subdirectory.
2. Add a model definition and element definition for the new character.
3. If the character has animations, add them to `character_animations_table.mtl`.
4. Create a new VR-Forces character in the Simulation Object Editor.

8.1.1 Copy DI-Guy Files to the Data Directory

VR-Forces stores files for DI-Guy characters in `./SharedData/##/latest/ModelData/Lifeforms/DIGuy` and its subdirectories. After you create a new character using DI-Guy software, you must copy the files for this character to these directories.

DI-Guy documentation recommends that all customer content be placed in `$(DIGUY)/custom`. To follow this convention, create a directory called `./SharedData/##/latest/ModelData/Lifeforms/DIGuy/custom` and put your files in the appropriate subdirectories under it.

VR-Forces can visualize custom DI-Guy characters based on DIS enumerations mapped to element definitions or by recognizing custom DI-Guy PDUs.

- If you want to map your custom characters to element definitions, then you must add new simulation models in the Simulation Model Editor. This will automatically create the complementary model and element definitions and object type mappings.
- If you want to visualize characters based on custom DI-Guy PDUs without creating element definitions or object type mappings, place the DI-Guy files in the `./SharedData/##/latest/ModelData/Lifeforms/DIGuy/custom` directory as described in the previous paragraph. No further configuration is needed.

8.2. DI-Guy Configuration Files

VR-Forces can specify the character, appearance, and animation values for DI-Guy models used in 3D visualization applications such as VR-Vantage and the VR-Forces GUI. A model's character determines the kinds of movement it can make. The appearance determines what it looks like. The animation specifies the movements that the character makes.

In the VR-Forces GUI, you can specify the animation with the DI-Guy Animation task. You can set the appearance with the DI-Guy Appearance set data request. For details about specifying DI-Guy characteristics in the GUI, see "DI-Guy Animation", "DI-Guy Characteristics (Appearance, Head, Body Weight)", and "DI-Guy Hand Item".

You can specify the default character, appearance, and animation for a lifeform entity in the Simulation Object Editor. (For details, see "DI-Guy Animation" and "4.1.10 Editing a Lifeform's Visual Model and Animation" on page 81.) The DI-Guy mappings are stored in the entity's `.entity` file.

The DI-Guy animations available to lifeforms are controlled by `./appData/settings/vrfSim/character_animations_table.mtl`. This information is used to populate the lists in the Simulation Object Editor. You can edit this file to add additional DI-Guy characters to those already configured for VR-Forces. The syntax of entries is described in the file.

 Any characters you add to `character_animations_table.mtl` must be valid DI-Guy characters. You cannot arbitrarily make up new types of characters. If you create new characters using a supported version of the DI-Guy SDK, MAK ONE applications will support these characters.

8.2.1 Adding a New DI-Guy Animation

This section explains how to add a new DI-Guy animation to VR-Forces. It assumes that you have the DI-Guy software and have created the new animation.

 The new animation must be made with the same version of DI-Guy supported by VR-Forces. Please see the *VR-Forces Release Notes* for the currently supported version.

► **To add a new DI-Guy animation:**

1. Create the new animation in DI-Guy. You will create an animation and add it to a character's action table. Make sure that the name of the animation file matches the name of the action you added to the action table. For example, if the name of the animation is `layEgg`, but you call the action `lay_egg`, it will not work.

 There is a 1-1 relationship between characters and their action tables. You cannot use an animation based on a chicken character in a cow character's action table.

2. Add the `motion_name.bdm` file for the animation to `./data/Lifeforms/DIGuy/motions`.
3. Add the `action_table.cfg` file to `./data/Lifeforms/DIGuy/config/diguy`.
4. Add the new animation to `./appData/settings/vrfSim/character_animations_table.mtl`. This table determines whether an animation can be assigned to a character using the DI-Guy Animation task (**is-taskable-animation True**) or whether it is used based on the entity's behavior. For example, if a character is moving faster than a certain speed, it is given a running animation. An entry for our `lay_egg` example would look like the following:

```
(chicken
  (display-name "Chicken")
  (animations
    (animation
      (animation-name "lay_egg")
      (name-to-be-displayed "Lay an egg")
      (is-taskable-animation True)
      (animation-variant "normal")
    )
  )
)
```

```
(animation
  (animation-name "feeding")
  (name-to-be-displayed "Feed")
  (is-taskable-animation True)
  (animation-variant "normal")
)
...
```

8.3. How VR-Forces Maps DI-Guy Models

Ordinarily, VR-Forces chooses animations to play for a DI-Guy based on the lifeform's speed, position, and similar attributes. However, VR-Forces can specify the DI-Guy state and appearance with the Set DI-Guy Appearance set data request and the Set DI-Guy Animation task. Once you explicitly set the DI-Guy appearance or animation, VR-Forces no longer tries to choose animations and expects to be told how to render that lifeform for the remainder of the simulation.

8.4. Add a New DI-Guy Entity in the SOE

Add a new entity in the Simulation Object Editor, as described in "3.3. Creating a New Simulation Object" on page 57.

When you specify the 3D model, the list of character names is drawn from the **display-name** attribute in the entries in *./appData/settings/vrfSim/character_animations_table.mtl*. Therefore, it must have been updated before you can specify the 3D model in the Simulation Object Editor.



9. The Object Parameter Database

This chapter provides a detailed description of the object parameter database. It is most useful to VR-Forces users who need to develop and configure new components.

9.1. The Object Parameter Database	166
9.1.1 VR-Forces Classes and Object Parameters	167
9.1.2 Object Types	167
9.1.3 Parameter Types	170
9.1.4 Local and Remote Subcomponents	171
9.2. Component Descriptors and Parameters	175
9.2.1 Common Elements of Component Descriptors	175
9.2.2 Configuring the Priority of Components	178
9.3. Platform Files	178
9.4. System Definition Files	179
9.4.1 Creating a System Definition File	179
9.4.2 Configuring System Connections	180
9.5. Adding Variable Bindings to Platform and System Files	183
9.5.1 Adding a Variable Binding	184
9.5.2 Making Variables Editable in the Simulation Object Editor	185
9.5.3 Adding Platform Variable Bindings to the SOE	185
9.5.4 Adding System Definition File Variables to the SOE	187
9.6. State Properties and Parameter Properties	187
9.6.1 Adding State Properties and Parameter Properties	187
9.6.2 Getting and Setting State Properties Using Lua Scripts	190
9.6.3 Displaying State Property Values in the Information Dialog Box	190
9.6.4 An Example of Adding State Properties	190
9.7. Configuring Units	198
9.8. Configuring Tactical Graphics	198
9.9. Pathname Interpretation in Simulation Model Set Files	199

9.1. The Object Parameter Database

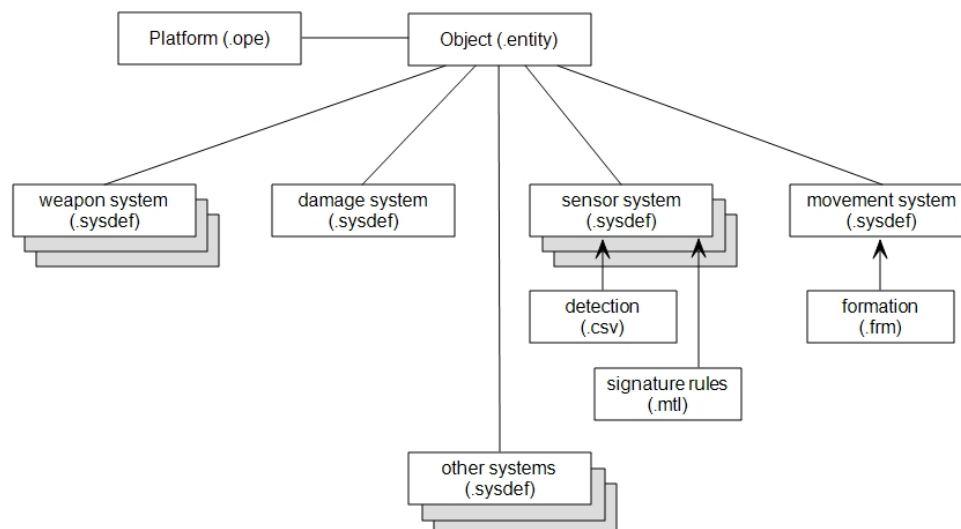
VR-Forces simulation objects and tactical graphics have many parameters. Parameter values are stored in files organized in simulation model sets (SMS) that VR-Forces reads when you load a scenario or create new objects. We refer to these files collectively as the object parameter database (OPD). The object parameter database specifies both the characteristics of a simulation object and starting state values. You can modify the behavior of a simulation object by changing the parameters for that type of simulation object. You can add new types of simulation objects and new behaviors to VR-Forces by adding new files to the object parameter database.

! Before you make any changes to the object parameter database, make a backup copy of the file you are changing.

Figure 86 illustrates the organization of an entry in the OPD. The primary definition of a simulated object, such as an entity, a tactical graphic, or a cultural feature, is stored in a file with the extension *.entity*. The *.entity* file specifies the object type enumeration, marking text, organizational information, and other parameters. It references a platform file (extension *.ope*), that provides a set of generic components for the object type, such as ground platform, surface platform, or line object. Each *.entity* file also specifies the systems that allow it to interact with the simulated environment, such as sensors, weapons, and movement systems.

Systems are configured in system definition files. System definition files group related component and connection specifications into convenient packages. The components reference additional configuration files. The system definition files also make it possible to add systems to simulation objects in the Simulation Object Editor.

Figure 86. Object definition



9.1.1 VR-Forces Classes and Object Parameters

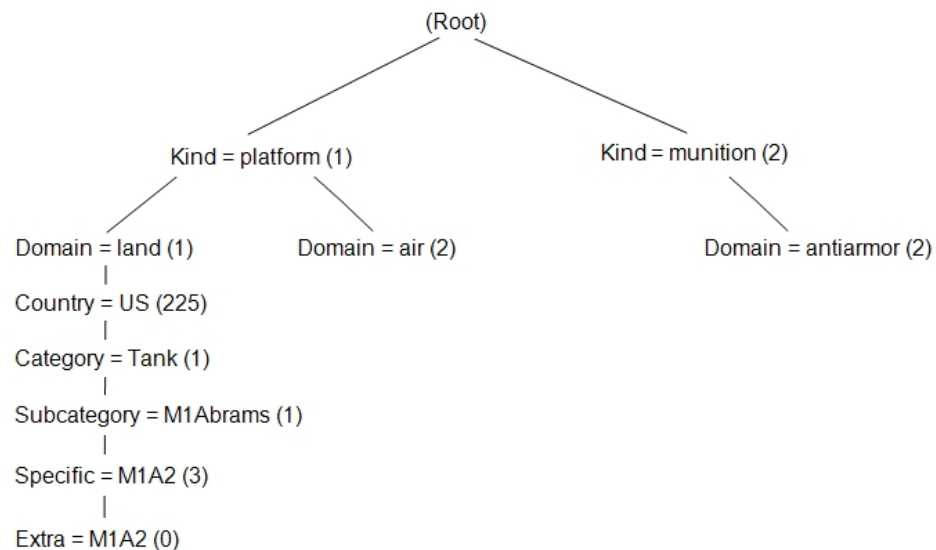
When you create any simulated object in VR-Forces, an instance of *DtVrfObject* is created in the simulation engine. Different types of simulation objects require different parameters for simulation and publishing, which are initialized by a subclass of *DtVrfObjectParameters*, and stored in a subclass of *DtVrfObjectBaseStateRepository*. Each *DtVrfObject* "has a" *DtVrfObjectParameters* and a *DtVrfObjectBaseStateRepository*. The type of parameters object to use for simulation object creation is specified in the simulation object's platform file by the **parameter-type** parameter. The state repository is specified in the **state-repository** parameter in the local-objects block. For example, when an entity that references the platform *Ground_Vehicle.ope*, which has a parameter type **ground-vehicle-param**, is created, a *DtVrfGroundVehicleParameters* object is created to read in the initial values from, and a *DtGroundVehicleStateRepository* is used to store runtime values.

9.1.2 Object Types

VR-Forces uses an eight-digit enumeration scheme for specifying objects. The enumerations are based on the seven-digit enumerations used by DIS and the RPR FOM. The additional field distinguishes between individual and unit objects.

This enumeration scheme describes a tree that goes from a more general specification to a more detailed one. Figure 87 illustrates a partial object type tree. To build an object type enumeration, start at the Kind and work your way down the tree, adding the number for each element. So, for example, if you follow the tree down to M1A2 you build the enumeration: 1 1 225 1 1 3 0. Since M1A2 is a leaf in the tree, the last field is 0. For more information about the object enumeration scheme, see the SISO Enumerations Document (*SISO Reference for Enumerations for Simulation Interoperability*).

Figure 87. Object type enumeration tree



The Object Super-Type

VR-Forces adds a field to the standard 7-digit entity enumeration so that we can manage objects that are not part of the DIS/RPR FOM enumeration scheme, such as units and tactical graphics. We enclose the seven-digit enumeration in parentheses and preface it with the object super-type (Figure 88). For example, the object enumeration for an M1A2, described in the previous paragraph, becomes the object type: 1 (1 1 225 1 1 3 0).

Figure 88. VR-Forces object-type parameter

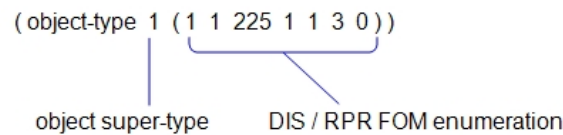


Table 14 lists the values VR-Forces can use for the object super-type. For more information, see *objTypeEnum.h* or "Object Type Enumerations".

Table 14. Object super-type field values

Object super-type	Meaning
0	Other.
1	Individual (platform entities, individual tactical graphics).
2	Unit.
3	Disaggregated unit (a unit composed of other simulation objects).
4	Aggregated unit (a unit that does not have subordinate simulation objects).

Published Object Types and Matching Object Types

Each simulation object in the OPD has two object types – the published object type and the matching object type.

The published object type is used as follows:

- When the GUI sends a create simulation object message to the sim. engine, the sim. engine uses the object type to determine which model to create.
- When the sim. engine publishes a simulation object on the network it uses the published object type as the object enumeration.

Each field of a published object type must have a specific value.

The matching object type is used as follows:

- When the sim. engine receives a request to create a simulation object but it cannot find an exact match for the object type that it is sent, it finds the best match among matching object types to determine which model to create. (For details about the best match method, see "The Best Match Method" on the facing page.)

- When the sim. engine discovers a simulation object on the network, it finds the best match among matching object types so that it can create local representative *DtVrf-Objects*.

The matching type can be the same as the published object type or it can have wildcards (-1).

For example, in VR-Forces, the published object type for an A-10 Thunderbolt is 1:2:225:2:4:1:0. If the matching type for the A-10 Thunderbolt is also 1:2:225:2:4:1:0 and VR-Forces discovers a remote simulation object with that same object type, it can match it to the A-10 Thunderbolt simulation model and can represent it properly in the simulation. However, if VR-Forces discovers a remote simulation object with the object type 1:2:225:2:4:1:1 (which is just a variation of an A-10), it will not be able to match the remote entity to an A-10 model because there is no A-10 match for that object type. (VR-Forces would end up matching the remote entity to a more generic fighter model, as explained in "The Best Match Method" below.) But if the A-10's matching object type is 1:2:225:2:4:1:-1, then VR-Forces will be able to match the remote entity to the A-10 model using the best match method.

Object types are specified in the *.entity* file:

```
<simObject objectType="1:1:2:225:2:4:1:0" matchType="1:1:2:225:2:4:1:-1" platform="@platforms-dir)/Fixed_wing_Aircraft.ope">
```

The **objectType** attribute specifies the published object type; **matchType** specifies the matching object type.

The Best Match Method

As explained in "9.1.2 Object Types" on page 167, object types describe a simulation object in increasing specificity as the object type fields move from left to right. Objects in the object parameter database are specified in a hierarchy in which entries at higher levels of the hierarchy have wildcard values (-1s) in object type fields that do not need to be specified at that level of the hierarchy. When VR-Forces looks up an object type, it begins at the root of the tree, working its way down until no better matches are found (the best match method.)

Suppose the object type hierarchy for fighter jets looks like this:

- -1:-1:-1:-1:-1:-1:-1 - match any object.
- 1:1:2:-1:-1:-1:-1 - match any fixed-wing entity.
- 1:1:2:225:2:-1:-1:-1 - match any U.S. attack fixed-wing.
- 1:1:1:2:225:2:4:1:-1 - match any A-10 Thunderbolt. (In VR-Forces, this is the A10 Thunderbolt matching object type.)
- 1:1:1:2:225:2:4:1:0 - match a specific A-10 Thunderbolt. (In VR-Forces, this is the A10 Thunderbolt published object type.)

As you replace each wildcard value (-1) with a specific enumeration, the scope of the object type narrows. Given this hierarchy, if VR-Forces discovers a simulation object with the object type 1:1:2:225:2:11:0:0, it tries to find the best match:

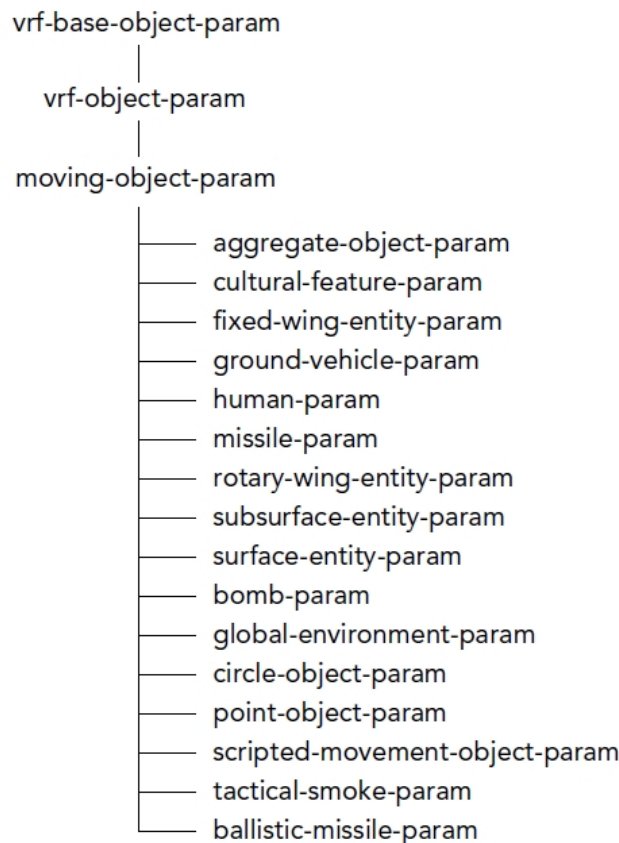
1. The discovered object type matches the top level - any object, because all fields are wild cards.
2. The discovered object type matches the fixed-wing object type, because the first three fields match and the rest are wild cards.
3. The discovered object type matches the U.S. attack fixed-wing object type.
4. The discovered object type does not match the A-10 Thunderbolt matching object type. Although the first five fields match, the sixth is different and is not wild carded.

The best match is the U.S. attack fixed-wing object type, so VR-Forces will use the generic U.S. attack fixed-wing model to represent the discovered simulation object.

9.1.3 Parameter Types

In addition to specifying an **object-type** for each object, you must specify a **parameter-type** for each object. The **parameter-type** determines the parameters and components that can be specified for an object. For example, a rotary-wing or fixed-wing entity can have a **nose-support** parameter, but a ground vehicle cannot. Figure 89 illustrates the object parameter type hierarchy.

Figure 89. Object parameter hierarchy



The `parameter-type` specifies the set of parameters appropriate to a simulation object, and the object state repository maintains state information for a simulation object, so the `parameter-type` and state repository must match. For more information, see "The State Repository Hierarchy" on page 174.

 The **object-type** is specified in an object's `.entity` file. The `parameter-type` is specified in the platform (`.ope`) file.

9.1.4 Local and Remote Subcomponents

As explained in "Local Objects and Remote Objects", each VR-Forces sim. engine simulates objects itself (local objects), and maintains state information about remote objects. All of these objects are instantiated as *DtVrfObjects*. The *DtVrfObject* class has a "has a" relationship with the classes:

- *DtSimObjectStateRepository*
- *DtSimObjectNetInterface*
- *DtSimTaskManager*
- *DtSimComponentManager*
- *DtVrfObjectPlanManager*

These classes are referred to as subcomponents of the object. Table 15 maps the subcomponent classes to the parameters that specify them.

Table 15. Subcomponent classes and parameters

Class	Parameter
<i>DtSimObjectStateRepository</i>	state-repository
<i>DtSimObjectNetInterface</i>	net-interface
<i>DtOrganizationManager</i>	organization-manager
<i>DtSimTaskManager</i>	task-manager
<i>DtSimComponentManager</i>	component-manager
<i>DtVrfObjectPlanManager</i>	plan-manager

VR-Forces creates different subcomponents for an object depending on whether it is a local object or a remote object. The subcomponents are specified in the **local-objects** and **remote-objects** parameter blocks for that object. Table 16 lists the permissible entries for each parameter listed in Table 15.

Table 16. Permissible subcomponent parameters

Parameter	Permissible values
state-repository	• ballistic-missile-entity-state-repository • bomb-entity-state-repository • cultural-feature-state-repository • fixed-wing-entity-state-repository • ground-entity-state-repository • human-state-repository • missile-entity-state-repository • rotary-wing-entity-state-repository • surface-entity-state-repository • subsurface-entity-state-repository • vrf-aggregate-state-repository • vrf-moving-object-state-repository • vrf-object-base-state-repository • vrf-object-state-repository • vrf-overlay-object-state-repository • vrf-tactical-smoke-state-repository

Table 16. Permissible subcomponent parameters (continued)

Parameter	Permissible values
net-interface	• aggregate-local-entity-net-interface • engineering-object-local-net-interface • individual-local-entity-net-interface-with-properties • local-control-object-net-interface • local-control-object-net-interface-with-properties • local-control-object-net-interface • local-environmental-net-interface • local-tactical-smoke-net-interface • pseudo-aggregate-local-entity-net-interface-with-properties • engineering-object-remote-net-interface • vrf-aggregate-remote-entity-net-interface-with-properties • vrf-individual-remote-entity-net-interface • vrf-individual-remote-entity-net-interface-with-properties • vrf-remote-environmental-net-interface • vrf-remote-environmental-net-interface-with-properties • vrf-remote-tactical-smoke-net-interface
task-manager	• local-task-manager
component-manager	• component-manager
plan-manager	• vrfobject-plan-manager

Every object must have a state repository specified. Objects that should be published need a net interface. Other subcomponents are optional. However, for most simulation objects, you will want to specify a task manager, plan manager, and component manager so that you can control the simulation object and give it tasks.

The local and remote objects parameters are specified in the platform (.ope) files. Therefore they apply to all simulation objects that reference a particular platform. Use the existing entries in the object parameter database as guidance if you create a new platform or change an existing platform. A typical entry for local and remote objects is:

```
(local-objects
 (state-repository "ground-entity-state-repository")
 (state-repository-min-tick-period -1.000000)
 (state-repository-min-tick-period-variance -1.000000)
 (net-interface "individual-local-entity-net-interface-with
 properties"))
```

```

(net-interface-min-tick-period -1.000000)
(net-interface-min-tick-period-variance -1.000000)
(task-manager "local-task-manager")
(component-manager "component-manager")
(plan-manager "vrfoobject-plan-manager")
(nav-interface "active-bot")
)
(remote-objects
  (state-repository "vrf-moving-object-state-repository")
  (state-repository-min-tick-period -1.000000)
  (state-repository-min-tick-period-variance -1.000000)
  (net-interface "vrf-individual-remote-entity-net-interface-with
properties")
  (net-interface-min-tick-period -1.000000)
  (net-interface-min-tick-period-variance -1.000000)
  (task-manager "")
  (component-manager "")
  (plan-manager "")
  (nav-interface "dynamic-obstacle")
)

```

The State Repository Hierarchy

Each VR-Forces object must have a state repository to maintain essential state information about the object, such as its current location, orientation, appearance, and so on. Therefore, each object must specify a state repository and the state repository must match the **parameter-type** for the object. (The state repository and **parameter-type** are specified in the platform file.)

Table 17 lists parameter types and the state repository type required.

Table 17. State repository and parameter types

State Repository Type	Parameter Type
ballistic-missile-entity-state-repository	ballistic-missile-param
bomb-entity-state-repository	bomb-param
cultural-feature-state-repository	cultural-feature-param
fixed-wing-entity-state-repository	fixed-wing-entity-param
ground-entity-state-repository	ground-vehicle-param
human-state-repository	human-param
missile-entity-state-repository	missile-param
rotary-wing-entity-state-repository	rotary-wing-entity-param
surface-entity-state-repository	surface-entity-param
subsurface-entity-state-repository	subsurface-entity-param
vrf-aggregate-state-repository	aggregate-object-param
vrf-moving-object-state-repository	moving-object-param

Table 17. State repository and parameter types (continued)

State Repository Type	Parameter Type
vrf-overlay-object-state-repository	vrf-object-param
vrf-object-state-repository	
vrf-tactical-smoke-state-repository	tactical-smoke-param

The Net Interface Hierarchy

VR-Forces manages network relationships of local and remote simulation objects through the network interface classes. Choose the set of local and remote network interface types that most closely represent the type of object being simulated. For example, for an individual vehicle such as a tank, choose individual-local-entity-net-interface for the local network interface and vrf-individual-remote-entity for the remote network interface. See "Object Parameters" for tables that list the network interfaces used for the simulation objects provided with VR-Forces.

9.2. Component Descriptors and Parameters

The parameters for a simulation object include a list of all the sensor, controller, and actuator components it uses. (For details, see "VR-Forces Uses a Component Architecture".) These components might be in the OPE file or in a system definition file. If there are no components of a certain type (for example, no sensors), you can omit the list. (In other words, you do not need a placeholder list for unused components.)

A component descriptor is a text block in a platform or system definition file that describes a component (see Example 1). A component descriptor must include the type of the descriptor and the type of the component that it is describing. A component descriptor can also contain parameters for the component. "9.2.1 Common Elements of Component Descriptors" below describes the construction of a component descriptor.

Components can receive data from, and convey data to, other components through ports. These ports are specified in platform files and system definition files.

A component can create a port group, which groups a set of related ports together. For example, the automotive-control port group groups the throttle, steering, and brake ports. Use of port groups simplifies specification of connections because you just have to specify a port group instead of many individual ports. For a detailed discussion of ports and port groups, please see API documentation.

The port group also provides a mechanism for ensuring that only one data provider attempts to put values on those ports each simulation frame. For more information, see "9.2.2 Configuring the Priority of Components" on page 178.

9.2.1 Common Elements of Component Descriptors

The common elements of component descriptors are:

- **Name** – The first element in a component descriptor specifies the component's name. This name can be whatever you want, but it must be unique within the simulation object or the system the component is defined in. It is used in the

connections section of the simulation object's parameters. Other components may use this name to find the specified component. In Example 1, the name of the first component is move-along (line 2).

- **component-descriptor-type** – The component descriptor factory creates the type of component descriptor specified by this type. This lets the system know what types of additional data may be present. In Example 1, the component descriptor type for the move-along component is ground-move-to-descriptor (line 3).
- **component-type** – The component factory creates the type of component specified in this entry. In Example 1, the component type is ground-auto-move-along-controller (line 4).

i The **component-descriptor-type** and **component-type** values are specified in code. You cannot make them up. See *compTypes.h* and *compDescTypes.h*.

- **connections (optional)** – The connections (Example 1, line 44) entry specifies how the components connect to one another (through ports, port groups, or both). Components attach to each other by connecting output and input ports to one another, or output port groups to input port groups. Since a component can have more than one port, port group, or both, the name of the port or port group to use is also specified. If a simulation object does not have any components that need to connect to one another, then you can omit this entry. Items in the connections entry can be in one of the following forms:

- Specify a port-to-port connection:

```
connect component_name:output_port component_name:input_port
```

- Specify a port group-to-port group connection:

```
connect component_name:output_port_group
      component_name:input_port_group
```

The connections in Example 1 use this format.

- Specify a port-to-port connection, in which one or both of the ports exists inside of a group:

The input port belongs to an input port group:

```
connect component_name:output_port
      component_name:input_port_group:input_port
```

The output port belongs to an output port group:

```
connect component_name:output_port_group:output_port
      component_name:input_port
```

Both input and output ports belong to groups:

```
connect component_name:output_port_group:output_port
      component_name:input_port_group:input_port
```

i The port and port group names are defined in code for each component. Please see class documentation for the names of port and port groups used by a given component.

Example 1 shows component descriptors for the Move Along and Patrol Between Points controllers and the Powertrain actuator for a ground vehicle (from the *ground-tracked.sysdef* file.) The Powertrain actuator simulates movement of a ground vehicle based on the throttle, brake, and steering inputs it receives from the automotive controllers. The example also shows how the output port groups for the controllers are connected to the input port group of the powertrain.

Example 1

```

1  (controllers
2      (move-along
3          (component-descriptor-type "ground-move-to-descriptor")
4          (component-type "ground-auto-move-along-controller")
5          (min-tick-period -1.000000)
6          (min-tick-period-variance -1.000000)
7          (tick-period-uses-real-time False)
8          (process-state-repository-name "")
9          (process-state-repository-type "")
10         (is-enabled True)
11         (near-distance 25.000000)
12         (at-distance 1.000000)
13         (approach-speed 2.000000)
14     )
15     (patrol-between-points
16         (component-descriptor-type "ground-move-to-descriptor")
17         (component-type "ground-auto-patrol-between-controller")
18         (min-tick-period -1.000000)
19         (min-tick-period-variance -1.000000)
20         (tick-period-uses-real-time False)
21         (process-state-repository-name "")
22         (process-state-repository-type "")
23         (is-enabled True)
24         (near-distance 25.000000)
25         (at-distance 1.000000)
26         (approach-speed 2.000000)
27     )
28 ) ;; end controllers
29 (actuators
30     (powertrain
31         (component-descriptor-type "automotive-component-
32 descriptor")
33         (component-type "automotive-actuator")
34         (min-tick-period -1.000000)
35         (min-tick-period-variance -1.000000)
36         (tick-period-uses-real-time False)
37         (process-state-repository-name "")
38         (process-state-repository-type "")
39         (is-enabled True)
40         (art-part-list )
41         (check-terrain-collisions False)

```

```
41         (use-high-fidelity-ground-clamping True)
42     )
43 ) ;; end actuators
44 (connections
45     (connect move-along:automotive-control powertrain:automotive-
46       control)
47     (connect patrol-between-points:automotive-control
48       powertrain:automotive-control)
49 ) ;; end connections
```

The connection in line 47 would read: the move-along component sends output to its automotive-control output port group. The powertrain receives the move-along controllers output through its automotive-control input port group.

9.2.2 Configuring the Priority of Components

Each simulation frame, VR-Forces ticks each simulation object's components in the order in which they are listed in the platform and system definition files. If a component believes it should be providing data, it takes control of the receiving component's ports or port group. Once a component acquires the ports, no other component can send data to that component during that frame. Therefore the order in which components are listed provides a natural priority scheme.

For example, among the ground movement controllers, the collision avoidance controller is placed on the controller list before any of the other movement controllers. If this controller needs to provide input to the ground automotive actuator to avoid a collision, it takes control of the automotive-control port group on the next tick and overrides the controller that previously owned the port group (such as Move To or Move Along).

9.3. Platform Files

Platform files (extension *.ope*) are the foundational configuration files for all simulated objects. They provide the basic description of each type of simulation object that you can create, such as a ground vehicle, surface vessel, or human. They also define types of tactical graphics. All platform files are part of an SMS. A platform file includes all of the components that every simulation object of that type needs to have, such as collision avoidance, radios, script controllers, and so on.

A platform file does not include systems that vary among specific examples of the platform, such as weapon systems, movement systems, and damage systems.

A platform file uses variables for the parameters that need to be different for specific examples of a platform. For example, all ground vehicles have a maximum speed, but tanks and civilian vehicles have a different value for this parameter. The values for these variables are set in the Simulation Object Editor.

You can create and edit platform files using a text editor. Platform files use MTL syntax. Most VR-Forces users will not need to create new platform files. You may need to add components or change parameters to be variables.

9.4. System Definition Files

The Simulation Object Editor allows VR-Forces users to easily add component systems, such as weapons, sensors, and movement, to simulation objects without needing to understand the components, configuration files, and connections required to make these systems work. These systems are defined in system definition files (*.sysdef*). System definition files are referenced in a simulation object's *.entity* file. They are read by the SOE and can be edited in it.

System definition files have two major sections – data and metadata. The data is a list of the components and connections required to implement the system for a simulation object.

The metadata explains to the Simulation Object Editor how to present the system to the user and how to integrate it into the entity model.

Weapon systems and damage systems have two system definition files, a template file (*.template_sysdef*), which specifies components shared by similar types of weapons or damage systems, and a weapon or platform-specific system definition file (*.sysdef*).

9.4.1 Creating a System Definition File

If you want to create a new system definition file, MAK recommends that you copy an existing file and use it as a starting point for adding the components that you want to configure into a system. You can use the SOE to do this or a text editor.

The format of a system definition file is:

```
(system-name
  (controllers
  )
  (actuators
  )
  (connections
  )
  (resources
  )
  (meta-data
    (system-name "Spot Report Generator")
    (system-description "Allows the entity to send spot reports
      through its radio when its sensors detect other entities
      in the scenario.")
    (allowed-state-repository-types "all")
    (system-categories "other")
    (meta-data-entry-list
    )
    (parameter-data-list
    )
  )
)
```

As you can see, system definition files use MTL syntax. (For information about MTL syntax, see "MAK Technologies Lisp (MTL) Files".) The system-name is an arbitrary name that uniquely identifies the system. The component sections list the various controllers, actuators, sensors, and resources that are needed to build the system. The connections section lists the connections among the components.

 You can call the system whatever you want, but the name must not start with a number.

9.4.2 Configuring System Connections

Within each system definition file, you list the connections among the components in the file. (For general information about connections, see "9.2.1 Common Elements of Component Descriptors" on page 175.) For systems that are self-contained, for example movement systems, there is no need to connect to components in other systems. However, some systems need to connect to components outside their system. For example, a visual sensor needs a list of object types to detect. This section explains how to make these out-of-system connections.

To understand how system definition files and *.entity* files are integrated, we will look at:

- A **componentSystem** element in a *.entity* file.
- The related system definition file.
- The connection section in the system definition file.

The M1A2, which uses the Ground_Vehicle platform, is configured with a visual sensor (in *M1A2_Abrams_MBT.entity*):

```
<simObject objectType="1:1:1:225:1:1:3:0" matchType="1:1:1:225:1:1:3:-
1" platform="platforms\Ground_Vehicle.ope">
...
<componentSystem systemName="sensor" platform="@(\system-
dir)/sensors/visual-sensor.sysdef">
  <bodyPosition paramName="sensor-position" right="0.57" forward="-
1.14" up="2.1"/>
  <real paramName="max-range">4000</real>
</componentSystem>
```

The sensor component references the system definition file *visual-sensor.sysdef*. The parameters specified provide values for the variables in the system definition file. (For more information, see "9.5. Adding Variable Bindings to Platform and System Files" on page 183.)

The system definition file has the following connections:

```
(visual-sensor-system
...
(connections
(connect system:object-types-to-detect visual-sensor:object-
types-to-detect)
```

```

        (connect visual-sensor:detected-objects system:detected-
objects)
        (connect system:sensor-offset visual-sensor:sensor-offset)
    )
    (resources )
    (meta-data
        (system-name "Visual Sensor")
        (system-description "Allows an entity to detect other objects
through visible light.")
        (allowed-state-repository-types "all")
        (system-categories "sensor")
    ...
    )

```

The system definition file defines connections for its components. The first connection in the list (line 4) means that some external component (**system**) is providing input from its **object-types-to-detect** port to the visual sensor's **object-types-to-detect** port. The second connection specifies that the visual sensor's **detected-objects** port sends data to some external component's **detected-objects** port. When VR-Forces creates a simulation object, it needs to substitute an actual component for the generic "system" component specified in the system definition file. It does this by referring to an auto-connection block in the OPE file

The platform file has a set of auto-connection blocks like this (from *Ground_Vehicle.ope*):

```

(auto-connection
    (source-controller "target-selection")
    (source-port-name "object-types-to-detect")
    (destination-category "sensor")
    (destination-port-name "object-types-to-detect")
)

```

The auto-connection block in the OPE file is a template that VR-Forces uses to match systems of a specified category to the appropriate controllers. In this example, the **target-selection** controller sends data through its **object-types-to-detect** port to the **object-types-to-detect** port of any system that is in the sensor category. Since the visual sensor is in the sensor category, it substitutes the **target-selection** controller for the generic system in building its connections.

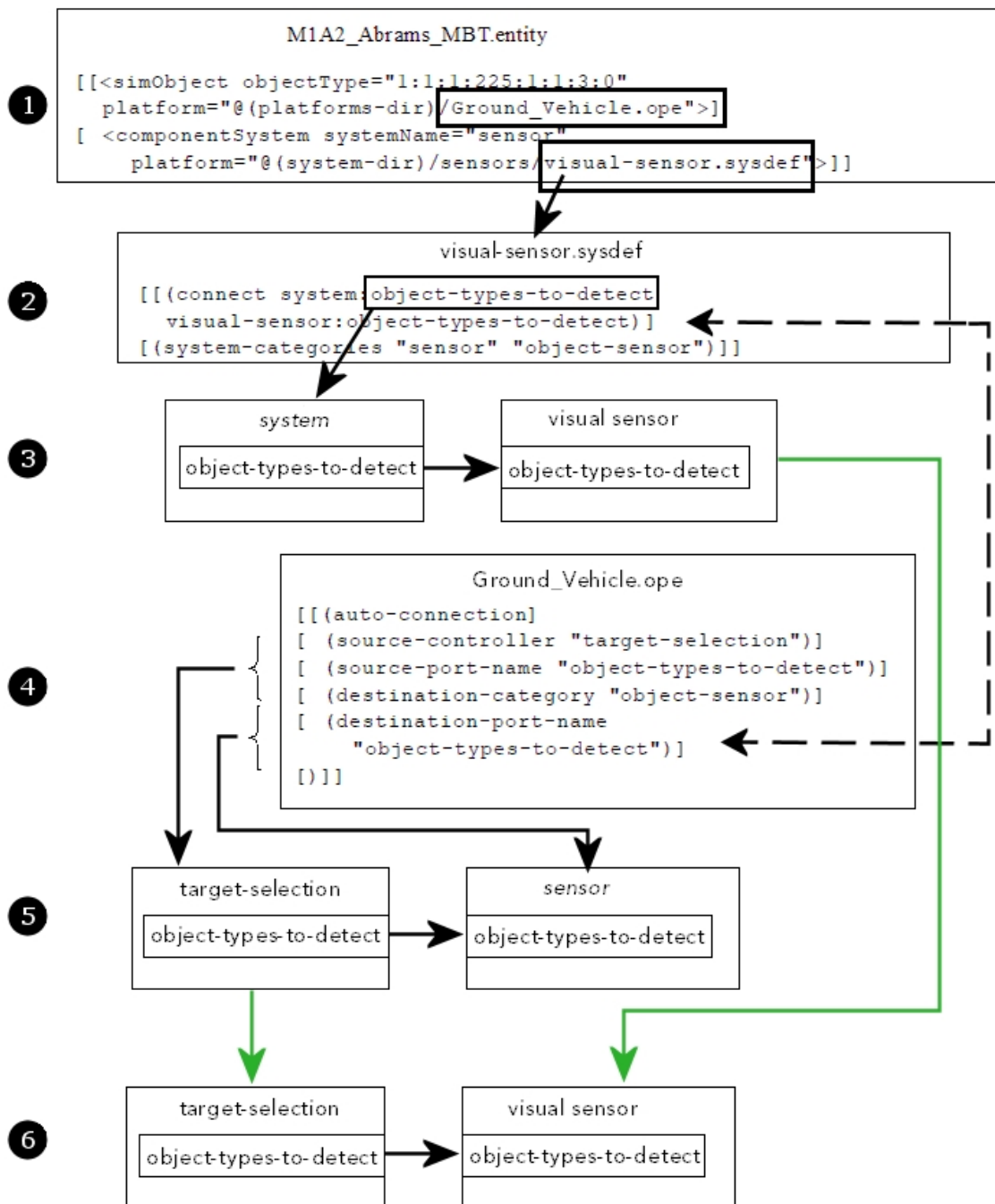
Figure 90 illustrates the relationships of the OPE file, *.entity* file, and system definition files referenced in this section and how connections are made.

The M1A2 entity, which uses the Ground_Vehicle platform, has a visual sensor (callout 1). The visual sensor has several connections. The one illustrated in the figure (2) takes input from an external source's **object-types-to-detect** port and sends it to the visual sensor's **object-types-to-detect** port (3). In this connection, the external source is represented by **system**. Before the sensor can actually work, this unspecified system must be determined.

When the entity is created, VR-Forces checks all the auto-connections in the OPE file (4). In this case, there is an auto-connection from the target-select controller's **object-types-to-detect** port to the **object-types-to-detect** port of a component whose destination category is **sensor** (5). To make this connection, the unspecified sensor must be determined.

VR-Forces matches the **destination-category** value "sensor" in the OPE file to the **system-categories** value "sensor" in the system definition file and the connections can now be completed (6).

Figure 90. External component connection



Systems that use templates must pass information between template files and individual system files. The "all" connection supports this exchange of information, as in this example:

(connections

```
(connect system:all main-14_5mm-gun:all)
(connect main-14_5mm-gun:all system:all)
)
```

 Most VR-Forces users add new systems by copying existing ones and changing their parameters. If you add new systems in this way, you do not need to worry about adding connections because the existing auto-connection system will handle them automatically. If developers add new types of components that require intersystem connections, they must add new auto-connection blocks to the appropriate OPE files for them to connect.

9.5. Adding Variable Bindings to Platform and System Files

Platform files and most system definition files are used by multiple simulation objects. However, the values of some parameters may need to vary for the different simulation objects that use them. This is handled by using variables in the platform and system definition files and specifying the values for these variables in the *.entity* files. VR-Forces uses variable bindings for the parameters that you are most likely to want to change from simulation object to simulation object. If there are other parameters that you want to vary, you can add your own variable bindings.

When you add a variable binding, you must:

1. Change the hard-coded value to a variable in the platform or system definition file.
2. Add the variable to the *.entity* file.
3. Update the Simulation Object Editor so that you can edit the value for the new variable.

This example shows a variable binding for fuel in a movement system. The variable is specified in the system definition file:

```
(resources
  (fuel
    (resource-type "real-resource")
    (amount $fuel-amount)
    (full-amount $fuel-amount)
  )
)
```

The *.entity* file for a simulation object that uses this movement system specifies a value for the variable:

```
<componentSystem systemName="movement"
  platform="@ (system-dir)/movement/ground-tracked.sysdef">
  <real paramName="fuel-amount">465</real>
</componentSystem>
```

As you can see, the parameter defined as **fuel-amount** in the *.entity* file is referenced as `$fuel-amount` in the system definition file.

9.5.1 Adding a Variable Binding

The procedure for adding a variable binding is the same for platform files and system definition files.

► **To add a variable binding to a platform file or system definition file:**

1. In the system definition file, substitute a variable name, in the format *\$variable-name* for the hard-coded value that you want to replace. For example, the *Ground_Vehicle.ope* file has a value **is-organized**:

```
(groundParams
  ...
  (is-organized True)
  ...
)
```

To make this a variable, replace the value with a variable name:

```
(groundParams
  ...
  (is-organized $entity-is-organized)
  ...
)
```

You can do this by hand in the OPE file.

2. Optionally, specify a default value for the variable. The default value is used if no value is specified for an entity type that uses this OPE file. The syntax for specifying a default value is:

```
(default "value")
```

In this example, to set the default value to True:

```
(groundParams
  ...
  (is-organized $entity-is-organized (default "True"))
  ...
)
```

3. In each *.entity* file that uses this platform file or system definition file, add an element to specify a value for the variable. The attribute must specify the type of the value, the name of the variable, and the value for the variable, for example.

```
<simObject objectType="1:1:1:225:1:1:3:0"
  platform="platforms\Ground_vehicle.ope">
  <bool paramName="entity-is-organized">True</bool>
  ...
</simObject>
```

Although you can add these entries by hand, it is highly recommended that you do so in the Simulation Object Editor by applying platform updates. When you apply platform updates, all changes to platform and system definition files are applied to all simulation objects that use them. For details, see "2.9. Applying New and Updated Variables" on page 24.

 If you do not add an attribute to an *.entity* file that references the modified platform or system definition file, a default value will be used, which may result in unexpected behavior.

9.5.2 Making Variables Editable in the Simulation Object Editor

When you add new variable bindings, you can make them editable in the Simulation Object Editor (SOE). Variables in platform files are edited in the General, Physical, and Movement tabs. They include variables such as speed, mass, and bounding volume. Variables in system definition files are edited in the Properties dialog boxes for the individual systems. The procedures for adding these two types of variables to the Simulation Object Editor are different.

9.5.3 Adding Platform Variable Bindings to the SOE

Variable bindings for platform files are edited in the General, Physical, and Movement tabs. The layout of these tabs is controlled by *.ui* files. The Simulation Object Editor has a *.ui* file for each platform. The files are in *./data/simulationModelSets/<model_set>/gui/ui*. You can edit *.ui* files in a text editor or in Qt Designer. Qt Designer is included with VR-Forces (*./bin64/designer.exe*).

 You do not need a developer's license to use Qt Designer.

In "9.5. Adding Variable Bindings to Platform and System Files" on page 183, we added a new variable binding to the *Ground_Platform.ope* file. The corresponding entry in *ground-vehicle-param.ui* file would be:

```
<item>
  <widget class="QCheckBox" name="entity_is_organized">
    <property name="text">
      <string>Is Organized</string>
    </property>
  </widget>
</item>
```

Notice that the variable name *entity-is-organized* in the OPE file is expressed as *entity_is_organized* in the *.ui* file. You must use underscores in the *.ui* file for the variable name.

If you are adding an attribute to the Simulation Object Editor and editing the *.ui* file by hand, the easiest approach is to copy an existing item, paste it where you want, and edit it for the new attribute. Many of these attributes have the **display-units** and **source-units** properties. For attributes that are expressed in units, such as meters or kilograms, these properties specify the unit. The unit value is expressed in the attribute label, for example (m) or (kg). If your new attribute is not expressed in units, leave these fields blank.

UI File Details

Whether you edit a *.ui* file by hand or using Qt Designer, you need to understand how data entry elements are defined. This varies depending on the type of widget used, such as text box, check box, option button, or list.

It is beyond the scope of this manual to explain Qt Designer and GUI layout. However, at the most basic level, the controls in the GUI are specified in item elements. Each item has a widget that has a **class** and an **objectName**. (The XML attribute is **name**. In Qt Designer it is **objectName**.)

- The **objectName** for an item is the platform variable name with dashes changed to underscores. For example, the **objectName** for the short-name variable is short_name.
- If an item has a label, the **objectName** must be <variable_name>_label. For example, the **objectName** for the Short Name label is short_name_label.

Each data entry widget creates a data entry field that is appropriate for the variable type (integer, real, string). Other elements (systems, vector entry) create more complex controls.

Depending on the type of information displayed, you can assign additional properties to each data entry field. These properties can be:

- **element_type**. String property that can be assigned a particular reader/writer (variable) type (for example, DtRwReal) or any other internal type that has been registered for element creations. These types are:
 - single_system_selector
 - multi_system_selector
 - sensor_signatures
 - aggregate_composition
 - aggregate_formation_editor
 - push_button
 - scripted-movement-system-selector.
- **enable_if_checked**. String property that defines a check box element. If the check box is selected, the variable is enabled. If clear, it is disabled.
- **disable_if_checked**. String property that will define a check box element. If the check box is selected, the variable is disabled. If clear, it is enabled.
- **disable_if_multiple_selection**. Boolean property. If selected, when the user selects multiple simulation objects, this item cannot be edited.
- **hide_by_default**. Boolean element. If selected, this item is hidden by default and is visible only if the Advanced button is clicked.
- **signal_slot**. String element. If defined for a push button, a particular slot is called when that signal is sent. The format is:

```
<signal>;<referencedControl>;<slot>released();bounding_volume;
```

onCalculateSupportPointsChanged()

The referenced control is the name of an existing control in the *.ui* file.

9.5.4 Adding System Definition File Variables to the SOE

When you add a system to a simulation object in the Simulation Object Editor, you can edit its properties. The Properties dialog box is not configured in a *.ui* file. It is generated based on the metadata section of the system definition file. Suppose that you want to change the rounds per minute parameter in *120mm-gun.sysdef* to a variable binding. You would create the variable binding as described in "9.5. Adding Variable Bindings to Platform and System Files" on page 183 and also add details to the metadata as shown in this example:

```
(weapon-120mm-gun
...
(rounds-per-minute $rounds-fired-per-minute)
...
(meta-data
  (parameter-data-list
    (bool-parameter-data
      (parameter-name "rounds-fired-per-minute")
      (variable-type "DtRWInt")
      (display-label "Rounds Fired Per Minute")
      (display-units "")
      (source-units "")
      (default-value True)
    )
  )
...)
```

9.6. State Properties and Parameter Properties

All simulation objects and tactical graphics have properties, such as ordered speed, mass, and force, that are stored in their state repository. You can add the following types of properties to these objects without using the VR-Forces Toolkit:

- State properties. A property that can store state information locally or can be published on the network. You can access the property using Lua scripts or the C++ API and you can change the value.
- Parameter properties. A state property whose value is static and cannot be changed at runtime.

The VR-Forces GUI and simulation components do not know anything about these state properties unless you write code to integrate them.

9.6.1 Adding State Properties and Parameter Properties

You define state and parameter properties for simulation objects that you want to have them. State properties are defined in the *state-data* section of the OPE files and parameter properties in the *parameter-data* section. You can also add state properties to an entity based on what sim components are added to the entity.

When you define a sim component in a sysdef, you can add a `state-data` or `parameter-data` section to the sim component description. When the entity creates that sim component, it automatically adds any defined state properties to the list of properties defined in its OPE.

The basic syntax for defining a state property is:

```
(type property_name default [publish record_type])
```

where:

- *type* is a data type. See Table 18 for a list of supported data types.
- *property_name* is the name of the property, such as `NumberOfWheels`.
- *default* is a default value.
- *publish* is an optional parameter that specifies that the state property should be published.
- *record_type* is an optional parameter to the *publish* flag. For details, please see "Publishing User-Created State Properties" in the *VR-Forces Developers Guide*.

This example specifies two state properties:

```
(state-data
  (DtRwInt TargetsMissed 0)
  (DtRwInt NumberOfWheels 4 publish)
)
```

Both properties are of type `DtRwInt`. The `TargetsMissed` property is initialized with the value 0. It is not published. The `NumberOfWheels` property has a default value of 4. It is published to the network.

You can bind state properties to variables. This allows you to specify different values for different simulation objects. For a complete discussion of binding properties to variables, see "9.5. Adding Variable Bindings to Platform and System Files" on page 183. The format for specifying a state property and binding it to a variable is:

```
(type property_name variable [(default default_value) publish record_type])
```

where:

- *type* is a data type. See Table 18 for a list of supported data types.
- *property_name* is the name of the property, such as `NumberOfWheels`.
- *variable* is a variable name that you use to add the property to platform and system files.
- *(default default_value)* specifies a default value for a variable. (Optional.)
- *publish* specifies that the state property should be published. (Optional.)
- *record_type* is a parameter to the *publish* flag. For details, please see "Publishing User-Created State Properties" in *VR-Forces Developers Guide*. (Optional.)

This example adds a state property with a variable binding:

```
(state-data
  (DtRwInt NumberOfWheels $number-of-wheels (default 4) publish)
)
```

You specify parameter properties in the `parameter-data` section of the OPE file. The syntax is the same.

Table 18 identifies the data types supported for state properties. The Data Type column indicates the name of the data type used in the OPE file definitions. (The data types used by C++ are slightly different.) The Description column describes the format of this type. The HLA Representation/Encoding column indicates what type should be used in an HLA 1516 FOM to match the specification in the OPE. For information about HLA and state properties, please see "Publishing User-Created State Properties" in *VR-Forces Developers Guide*.

Table 18. Data types supported by state and parameter properties

Data Type	Description	HLA Representation/Encoding
DtRwInt, DtRwInt32	A 32-bit integer	RPRunsignedInteger32BE, HLAinteger32BE
DtRwInt16	A 16-bit integer	RPRunsignedInteger16BE, HLAinteger16BE
DtRwInt8	An 8-bit integer	RPRunsignedInteger8BE, HLAoctet
DtRwReal, DtRwReal64	A 64-bit float	HLAfloat64BE
DtRwReal32	A 32-bit float	HLAfloat32BE
DtRwBoolean	A Boolean	RPRboolean
DtRwString	An ASCII string	RPRnullTerminatedArray of HLAASCIIchar
C++: DtRwPropertiesFixedArray OPE: DtRwFixedArray	A fixed sized list of items	HLAfixedArray
C++: DtRwPropertySizedString OPE: DtRwSizedString	An ASCII string (alternate encoding for DtRwString)	HLAASCIIstring
C++: DtRwPropertiesStructure OPE: DtRwStructure	A fixed structure with multiple named fields	HLAfixedRecord with matching fields.
C++: DtRwPropertiesList OPE: DtRwList	A variable sized list of items	HLAvariableArray
C++: DtRwPropertiesMap OPE: DtRwMap	A variable sized map of items	HLAvariableArray of HLAfixedRecords, where the first field in each record is the key.

9.6.2 Getting and Setting State Properties Using Lua Scripts

You can set and get the values for state properties using scripted tasks and sets or through C++ code. For details about accessing them using C++, please see the *VR-Forces Developers Guide*. This section describes how to access state properties using Lua.

To get the value of a state property from Lua, use the `getStateProperty()` function of a `SimObject`. Use `getParameterProperty()` to retrieve parameter properties.

```
local hits = this:getStateProperty("TargetsHit")
```

To set the value of a state property from Lua, use the `setStateProperty()` function.

```
this:setStateProperty("TargetsHit", 2)
```

The `setStateProperty()` function has an optional third argument that specifies that the property should be created if it does not already exist. If not set, this argument defaults to false. When a state property is added this way, VR-Forces must try to determine what the type of the property is, and it might not make the choice you intended, especially for properties with complex types. Therefore, if you are defining a complex property we recommend that you add it to an OPE file. Properties added using this argument cannot be published to the network.

```
this:setStateProperty("SomeNewProperty", 1.2, true)
```

9.6.3 Displaying State Property Values in the Information Dialog Box

You can add user-created state properties to the State Data page in the Information dialog box.

► **To add state properties to the State Data page:**

1. Open `./appData/settings/vrfGui/default_guiConfiguration.gui` in a text editor.
2. Find the `DtEntityInformationPage` `pageitem` element.
3. Add an `informationitem` element for each state property you want displayed, in the format:

```
<informationitem  
informationitemname="DtStatePropertyPageItem:MyStateProp:Display  
Name"/>
```

where:

- *MyStateProp* is the name of the state property as specified in the OPE file.
- *Display Name* is the text that you want to display for the property in the Information dialog box.

9.6.4 An Example of Adding State Properties

This section is an example of how you can add state properties, configure them in the Simulation Object Editor, set them with a scripted set, and use them in a scripted task.

For this example, let's assume a scenario in which ground vehicles need to travel from one location to another. There are several routes they can follow. One route can only be traveled by smaller wheeled vehicles. A second can be traveled by large wheeled vehicles, defined as any vehicles with more than four wheels. A third can be traveled by tracked vehicles.

To make this all work, we need to know if a vehicle has wheels, and if so, how many. We will start by adding state properties to hold this information.

Add the State Properties

We want to add two state properties. `haswheels` indicates that a vehicle has wheels. `numwheels` holds the number of wheels. We will add these properties to `./data/simulationModelSets/sms_name/vrfSim/platforms/Ground_Vehicle.ope` as variables so that we can add them to the Simulation Object Editor. `numwheels` has a default value of 4.

```
(state-data
  (DtrwBoolean haswheels $has-wheels publish)
  (DtrwInt numwheels $number-of-wheels (default "4") publish)
  ...
)
```

Add the State Properties to the Simulation Object Editor

You do not have to make state properties settable in the Simulation Object Editor, but if you want to use them for a lot of simulation objects, it is convenient to be able to set them when you add new simulation objects or update existing ones.

Ground vehicle variables are configured in `./data/simulationModelSets/sms_name/gui/ui/ground-vehicle-param.ui`. Each variable is configured as an `item` element. We want to add the state properties so that we set `haswheels` with a check box. If the check box is enabled, the `numwheels` field becomes enabled and we can set that.

If you take a look at the variables available to a ground vehicle in the Simulation Object Editor, you will see that many of them have check boxes and an associated value. You will also see that the Configure button is disabled unless Can Be Embarked On is selected. Using the existing item elements as a guide, we add the following code after Maximum Pivot Speed, so that it is listed after Maximum Pivot Speed widgets:

```
<item>
  <layout class="QHBoxLayout" name="horizontalLayout_66">
    <item>
      <widget class="QCheckBox" name="has_wheels">
        <property name="text">
          <string>Has wheels</string>
        </property>
        <property name="element_type" stdset="0">
          <string notr="true">DtrwBoolean</string>
        </property>
      </widget>
    </item>
```

```

    <item>
      <widget class="QLineEdit" name="number_of_wheels">
        <property name="enable_if_checked" stdset="0">
          <string notr="true">has_wheels</string>
        </property>
        <property name="source-units" stdset="0">
          <string notr="true"></string>
        </property>
        <property name="display-units" stdset="0">
          <string notr="true"></string>
        </property>
      </widget>
    </item>
  </layout>
</item>

```

We set the name for the layout element to a unique name (horizontalLayout_66).

The QCheckBox widget has the name `has_wheels`. This is the variable for the `haswheels` property (`$has-wheels`), with the hyphen replaced with an underscore. (This convention is described in "9.5.2 Making Variables Editable in the Simulation Object Editor" on page 185 and "9.5.3 Adding Platform Variable Bindings to the SOE" on page 185.) The label for the check box is Has Wheels and it is of type `DtRwBoolean`, which matches the data type we set in the OPE file.

The QLineEdit widget (the data entry field) has the name `number_of_wheels` to reference the `$number-of-wheels` variable. The **enable_if_checked** property is set to `has_wheels`. This means if `has_wheels` is checked, `number_of_wheels` is enabled. We left the other properties blank since we do not express the number of wheels in any unit of measure.

i You can also edit *ground-vehicle-param.ui* in Qt Designer, which is included with VR-Forces (*./bin64/designer.exe*). When you add a widget for a variable, the `objectName` property should be set to the variable name, for example, `number_of_wheels`.

Update the Platform in the Simulation Object Editor

Now we need to update the Simulation Object Editor to use the updated *ground-vehicle-param.ui* file.

1. Start the Simulation Object Editor.
2. Load an entity-level SMS file that you can edit. If you do not have one, see "2.7. Creating New Simulation Model Sets" on page 22 for instructions.
3. Choose **File > Regenerate SMS Files**. An information prompt opens.
4. Click OK. The Simulation Object Editor is updated so that all ground vehicles now have the Has Wheels check box and the number of wheels data entry field (Figure 91).

Figure 91. Has Wheels check box and data entry field

The screenshot shows the 'Movement Dynamics' configuration window. The 'Movement Dynamics' dropdown is set to 'Wheels (off road)'. The 'Has Wheels' checkbox is checked, and the 'Number of wheels' field is set to 8. Other parameters like Mass, Maximum Speed, and Acceleration are also visible.

Parameter	Value
Mass(kilograms):	29440.00
Maximum Speed(km/h):	87.84
Default Speed(km/h):	36.00
Maximum Reverse Speed(km/h):	36.00
Maximum Slope(%):	50.00
Turning Radius(m):	4.00
Maximum Acceleration(m/s^2):	4.00
Maximum Deceleration(m/s^2):	4.00
Maximum Lateral Acceleration:	4.00
Can Pivot	☐
Maximum Pivot Speed(deg/s):	0.00
Has Wheels	☒
Number of wheels	8
Maximum Disaggregation Distance	
Can Embark	☒
Can Be Moved Onto For Embark	☒
Can Be Embarked Upon	☒ Configure...
Protect Embarked Entities from Collateral Damage	☒
Enable Navigation	☒
Use Detailed Geometry in Simulation Engine	☐
Use Object Visuals	☐
Model Path	\$\{SHARED_DATA_DIR\}\ModelData\ Vehicles\Wheeled\M977HEMTT\M Select...
NavigationData:	None Generate Clear

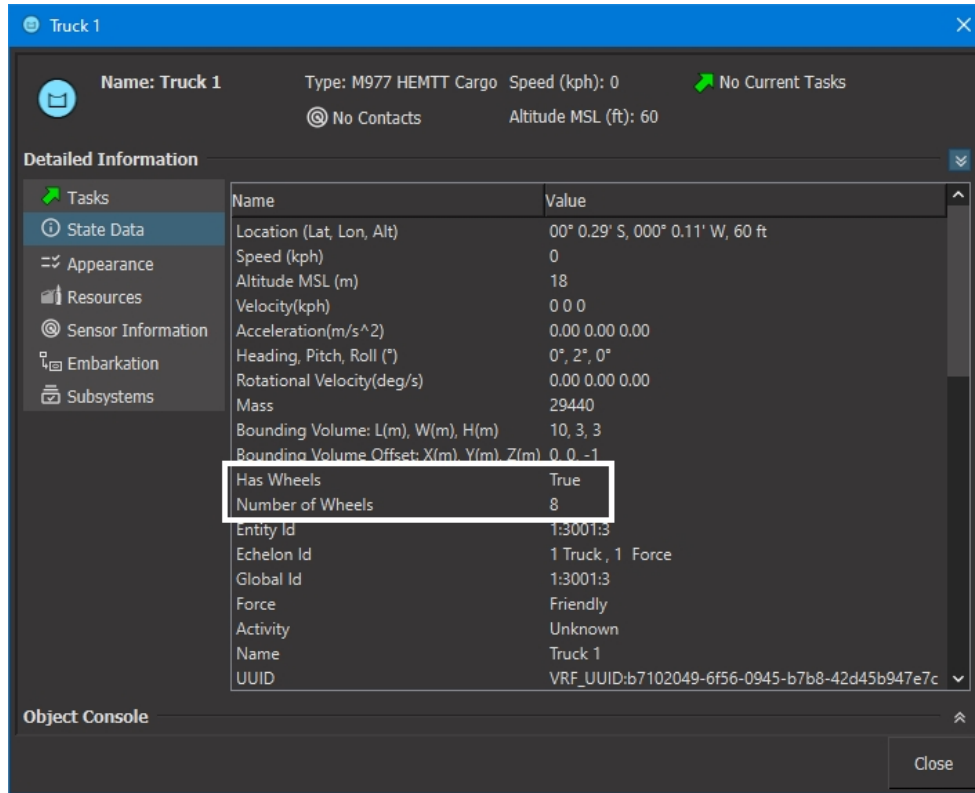
5. You can now configure vehicles to have wheels and set the number of wheels.

Add the State Properties to the Information Dialog Box

If you want to see the values for the state properties in the Information dialog box, add the following lines to the `DtEntityInformationPage` `pageItem` element in `./appData/settings/vrfGui/default_guiConfiguration.gui`:

```
<informationitem
informationitemname="DtStatePropertyPageItem:haswheels:Has wheels"/>
<informationitem
informationitemname="DtStatePropertyPageItem:numwheels:Number of
wheels"/>
```

Figure 92 shows the Information dialog box for an entity that is set to have wheels.

Figure 92. Information dialog box with user state properties added

Write a Scripted Set to Set the Number of Wheels

You might not want to configure every ground vehicle in the Simulation Object Editor with its wheel status. Maybe you just want to be able to set this in the GUI. To do that you can write a scripted set data request.

Figure 93 shows the setup for the Set Wheels scripted set. It has two parameters that match the state properties we added. It is valid for land entities and is on the Set menu.

Figure 93. Set Wheels metadata

Script ID: <auto generate>

Script Type: Set Language: Lua

Name: Set Wheels

Extended Name:

Short Description: Specify whether vehicle has wheels and, if so, how

☐ Description:

Version: 1

Tool Tip:

Menu Icon:

Command Level: Tertiary

Magic Command: No

☐ Show On Top Level Right Click Menu

☒ Show Set on Set Menu

Menu Location: Top Level Menu

☐ Show Set on Set Toolbar

Toolbar Location: At End of Toolbar

Parameters

Parameter	Type	Visible
hasWheels	Boolean	Yes
numberOf...	Integer	Yes

☒ Make Available Based on Entity Type

Valid for Object Types

All

Platform Land

☐ Make Available Based on Behavior Set

☐ Make Available to Global Plans

Edit Script Preview Add Cancel

The Lua script for Set Wheels is as follows. It starts out by getting the current values for the two state properties and printing them to the object console. Then it takes in the input from the Set Wheels dialog box and sets the state properties to the new values. Finally, it gets the state properties and prints them to the console to confirm that it worked. We have left out the generic comments and the unused functions and commented the code.

```
require "vrfutil"
function init()
    vrf:setTickWhilePaused(true)
    -- get the current values for the state properties
    local haswheels = this:getStateProperty("haswheels")
    local numwheels = this:getStateProperty("numwheels")
    -- if the entity has wheels, print out that it has them and how many
    -- this tests that we are getting the state properties correctly
    if haswheels then
        print("This vehicle has wheels.")
        print("Number of wheels is: ", numwheels)
    else
```

```
-- if no wheels print this instead
    print("This vehicle does not have wheels.")
end
finished = 0
vrf:setTickPeriod(0.5)
end
function tick()
-- this happens after we complete the Set wheels dialog box and
click OK
-- get the haswheels value from the dbx and set the state property
haswheels = setParameters.hasWheels
this:setStateProperty("haswheels", haswheels)
-- get the number of wheels from the dbx and set the state property
wheels = setParameters.numberOfWheels
this:setStateProperty("numwheels", wheels)

-- test it again. get the property values and print out the results.
newhaswheels = this:getStateProperty("haswheels")
newnumwheels = this:getStateProperty("numwheels")

if newhaswheels then
    print("This vehicle has wheels.")
    print("Number of wheels is: ", newnumwheels)
else
    print("This vehicle does not have wheels.")
end
finished = 1

if finished == 1 then
    vrf:endSet()
end
end
```

Write a Task that Uses the State Properties

Now we will write a script that uses the state properties. Remember that the premise of the exercise was based on the need for different vehicles to travel on different routes to reach a destination. In the scenario, we create three routes – Tracked, Heavy, and Light, that lead to the destination (Base). Our task, Move to Base, gets the haswheels and numwheels state properties and based on what it finds gives a Move Along Route task to the entity using the appropriate route. There is no input to the task, so the New Scripted Task dialog box just sets up the name of the task and puts it on the menu (Figure 94).

Figure 94. Move to Base metadata

New Script

Script ID: <auto generate>

Script Type: Task Language: Lua

Name: Move to Base

Extended Name:

Short Description: Choose the route to take based on the number of w

☐ Description:

Version: 1

☐ Depth control

☐ Lifeform Posture

☒ Movement

☐ Radio Communication

☐ Weapon

☐ Can Be Executed More Than Once at a Time

Tool Tip:

Menu Icon:

Command Level: Tertiary

Magic Command: No

☐ Show On Top Level Right Click Menu

☒ Show Task on Task Menu

Menu Location: Top Level Menu

☐ Show Task on Task Toolbar

Toolbar Location: At End of Toolbar

☒ Make Available Based on Entity Type

Valid for Object Types

All

Platform Land

☐ Make Available Based on Behavior Set

☐ Make Available to Global Plans

Edit Script Preview Add Cancel

Here is the Lua script:

```
require "vrfutil"
function init()
    moveAlongRouteSubtaskID = -1

    -- get the properties and print to the console for test purposes
    haswheels = this:getStateProperty("haswheels")
    numwheels = this:getStateProperty("numwheels")
    if haswheels then
        print("This vehicle has wheels.")
        print("Number of wheels is: ", numwheels)
    else
        print("This vehicle does not have wheels.")
    end
    vrf:setTickPeriod(0.5)
end
```

```
function tick()
  if moveAlongRouteSubtaskID == -1 then
    -- see if the vehicle has wheels. If yes, give proper move along
    task
    -- based on the number of wheels
    if haswheels then
      if numwheels <= 4 then
        moveAlongRouteSubtaskID = vrf:startSubtask("move-
along",
          {route = "Light"} )
      end
      if numwheels > 4 then
        moveAlongRouteSubtaskID = vrf:startSubtask("move-
along",
          {route = "Heavy"} )
      end
    end
    -- if no wheels, assume tracked and send on tracked route.
    else
      moveAlongRouteSubtaskID = vrf:startSubtask("move-a-long",
        {route = "Tracked"} )
    end
  end
  if vrf:isSubtaskComplete(moveAlongRouteSubtaskID) then
    vrf:endTask(true)
  end
end
```

In your scenario, create a few ground vehicles. If necessary, specify the number of wheels they have. Give them the Move to Base task and they will pick the correct route to move along.

9.7. Configuring Units

All units have the **parameter-type** aggregate-object-param.

In entity-level scenarios, most functionality resides in entities and units have relatively few individual parameters. The principle function of the various elements is to specify controllers and formations, and to specify the **echelon-level** for the different levels of units.

9.8. Configuring Tactical Graphics

The object parameter database includes parameters for tactical graphics. Tactical graphics provided with VR-Forces have the **parameter-type** vrf-base-object-param. You should not change the parameters for the objects provided with VR-Forces. However, if you are using the VR-Forces toolkit, you could create other types of tactical graphics. You would have to include parameters for them in the object parameter database.

9.9. Pathname Interpretation in Simulation Model Set Files

The files that make up a simulation model set, such as *.entity* files and system definition files, usually contain references to other files in the SMS. These supporting files may also have references to other configuration files. VR-Forces supports the file relationships by specifying files using relative paths and by defining directory variables. This allows you to maintain custom simulation model sets outside of the VR-Forces file hierarchy and not have to move them every time you install a new version of VR-Forces.

Examples of directory variables are in the SMS file and OPD file. These lines from the *EntityLevel.sms* file show a binding (model-set-directory) and its use:

```
(simulation-model-set
  (variable-bindings
    (DtRwString
      (model-set-directory "EntityLevel")
    )
    (DtRwString
      (opd-dir "$(model-set-directory)")
    )
    ...
  )
  (opd-file "$(model-set-directory)\vrfsim.opd")
  (physical-world "$(model-set-directory)\physicalworldParams.mtl")
  ...
)
```

Similarly, the *vrfsim.opd* file has bindings, which are then used in *.entity* files, as shown.

From *vrfsim.opd*:

```
(variable-bindings
  (DtRwString
    ...
    (system-dir "$(opd-dir)/vrfsim/systems"
    )
    (DtRwString
      (ammoselect-dir "$(opd-dir)/vrfsim/ammoselect"
    )
    ...
  )
)
```

From *M1A2_Tank.entity* (**system-dir** variable):

```
<componentSystem systemName="damage"
  platform="@$(system-dir)/damage/ground-heavy-armor.sysdef"/>
```




10. Configuring Sensors

This chapter explains how to configure sensors and sensor signatures.

10.1. Configuring Sensors	202
10.1.1 Adding a New Sensor to a Simulation Object	202
10.1.2 Editing a Sensor System Definition File	203
10.1.3 Specifying a Sensor Signature	204
10.1.4 Specifying Sensor Detection Probabilities	208
10.1.5 Adding Detection Types (Why Doesn't My Sensor Sense Anything?) ..	209
10.1.6 Adding a New Sensor Domain	210
10.1.7 Illumination and Time of Day	212
10.1.8 Configuring the Effects of Rain on Radar	213
10.2. Detection Tables	213

10.1. Configuring Sensors

VR-Forces includes several sensor systems and you can add them to simulation objects using the Simulation Object Editor. This section provides additional details about how sensors are configured. A complete sensor configuration requires:

- One or more **componentSystem** elements in the *.entity* file for sensor systems.
- A list of sensor signature values in the *.entity* file for the sensor domains that can detect the entity type.
- A signature propagator entry in the *./data/simulationModelSets/<model_set>/physicalWorldParams.mtl* file (one per sensor domain).

VR-Forces includes sensors such as radar, infrared, visual, magnetic anomaly detector (MAD), and sonar. The simulation objects configured in the object parameter database are configured with appropriate sensors and sensor signatures. For details about sensor components and how they work, please see API documentation.

10.1.1 Adding a New Sensor to a Simulation Object

It is recommended that you use the Simulation Object Editor to add new sensors to a simulation object. If you want to do it manually, you can edit its *.entity* file.

All sensors provided with VR-Forces are signature-object-sensors. They are distinguished by the domain in which they operate (for example, visual, radar, infrared, MAD, or sonar).

► **To add a sensor to a simulation object by editing its *.entity* file:**

1. Copy an existing **componentSystem** element in the *.entity* file and paste it into the file.
2. Change the **systemName** to be unique, for example, sensor-2.
3. Change the name of the system definition file to point to the sensor you want to add.
4. Change the values for the variables that must be specified for the sensor.

Connecting a Sensor

If you add a sensor that has been configured to use the auto-connection feature (all sensors provided by MAK support auto-connections), its connections are handled automatically. (For details, see "9.4.2 Configuring System Connections" on page 180.)

If you add a new type of sensor, you must connect it to input through its input port and then send the data to its output port for the appropriate controller. The sensor gets input through its *object-types-to-detect* input port. It sends that data out through its *detected-objects* output port. The connections for the visual sensor, in *visual-sensor.sysdef* are:

```
(connect system:object-types-to-detect visual-sensor:object-types-to-detect)
```

```
(connect visual-sensor:detected-objects system:detected-objects)
```

```
(connect system:sensor-offset visual-sensor:sensor-offset)
```

If you were to create an acoustic sensor, its connections would look like this:

```
(connect system:object-types-to-detect acoustic-sensor:object-types-to-detect)
(connect acoustic-sensor:detected-objects system:detected-objects)
(connect system:sensor-offset acoustic-sensor:sensor-offset)
```

10.1.2 Editing a Sensor System Definition File

This example is from the visual sensor system definition file (*visual-sensor.sysdef*):

```
(visual-sensor
  (component-descriptor-type "signature-sensor-descriptor")
  (component-type "signature-sensor")
  (min-tick-period -1.000000)
  (min-tick-period-variance 0.100000)
  (periodic-tick-while-paused False)
  (process-state-repository-name "")
  (process-state-repository-type "")
  (is-enabled True)
  (is-enabled-for-clutter-entities False)
  ...
  (detect-only-hostile-forces False)
  (detect-own-force False)
  (detection-types )
  (detect-destroyed-objects False)
  ...
  (sensor-geometry
    (in-range
      (range $max-range)
    )
  )
  ...
  (sensor-offset $sensor-position)
  (sensor-domain "visual")
  (sensor-positional-error 0.000000)
  (sensor-sensitivity $sensor-sensitivity-start)
  (detection-probability
    (round-down 1)
  )
  (detection-level-determinator
    (determinator-type "signature-detection-level-determinator")
    (detection-level-to-set-hostility 3)
    (detection-level-to-set-radius 4)
    (combat-identification-level-table-file
      (filename "$(detection-dir)\std-visual-detection-table.csv")
    )
  )
  ...
)
(controllers )
(actuators )
```

```
(connections
  (connect system:object-types-to-detect
    visual-sensor:object-types-to-detect)
  (connect visual-sensor:detected-objects system:detected-objects)
  (connect system:sensor-offset visual-sensor:sensor-offset)
)
```

As you can see from viewing the file, most of the parameters that you would want to vary from entity to entity are variables whose values are set in the *.entity* file.

10.1.3 Specifying a Sensor Signature

A sensor signature indicates that a simulation object can be detected by other simulation objects that have a sensor for that signature. The value of the signature represents the maximum distance, at which the entity can be detected. However, this value is affected by modifiers and the calculations made by the propagator. (Technically, the signature does not assume a particular measurement unit. As implemented, distance is measured in kilometers.)

The modifiers specify multipliers for the base value, based on the current state of the object. For example, a simulation object might have a base value of 5.0 for its visual signature, and a modifier of 1.5 when moving. This means the signature of the entity is 7.5 when moving and 5.0 when not moving.

This is the sensor signature block for humans:

```
(sensor-signatures
  (visual
    (base-signature $visual-signature)
    (modifiers
      (file
        (external-file
          (filename "signatureRules/lifeform-visual-rules.mtl")
        )
      )
      (file
        (external-file
          (filename "signatureRules/visual-illumination-rules.mtl")
        )
      )
    )
  )
  (infrared
    (base-signature $infrared-signature)
    (modifiers )
  )
  (radar
    (base-signature $radar-signature)
    (modifiers )
  )
)
```

This sensor-signature block shows that a DI M4 can be detected by visual, infrared, and radar sensors. The base-signature values are variables. Their values come from the *.entity* file.

```
<sensorSignatures>
  <real paramName="visual-signature">1.2</real>
  <real paramName="infrared-signature">1.2</real>
  <real paramName="radar-signature">0.25</real>
</sensorSignatures>
```

The visual sensor has two modifiers. The file *lifeform-visual-rules.mtl* is:

```
(signature-modifiers
  (speed-less-than
    (speed 1.0)
    (multiplier 0.5)
  )
)
```

This indicates that if the entity's speed is less than 1 KMH, multiply the base signature by 0.5. A slowly moving entity is less likely to be seen.

For details about the visual illumination rules modifier, see "10.1.7 Illumination and Time of Day" on page 212.

If you do not want a simulation object to be detectable by a particular domain of sensor, remove the sensor signature from the *.entity* file. If you want it to be detectable by an additional sensor domain, add its sensor signature to the *.entity* file. You can edit the values of sensor signatures in the Simulation Object Editor. You cannot add sensor signatures in the Simulation Object Editor.

VR-Forces has a set of predefined sensor signatures in *./data/simulationModelSets/EntityLevel/extra/sensor-signatures.csv*. These values are used only in the Simulation Object Editor to quickly set standard sensor signatures for entities in the various sensor domains. The standard sensor signatures are also used if you edit the properties of a sensor system in the Simulation Object Editor.

Table 19 lists the sensor signatures that VR-Forces uses for general categories of simulation objects in *EntityLevel.sms*. These categories do not match the tags list in the Simulation Object Editor or those in the *sensor-signatures.csv* file. They represent a more granular approach to categorizing simulation objects for the purpose of applying a consistent set of sensor signatures.

Table 19. Entity-level sensor signatures

Entity Category (with example)	Visual	Radar	Infrared	Sonar Active	Sonar Passive	MAD
Military Fixed Wing - stealth technology	2.5	4	6	N/A	N/A	N/A
Military Fixed Wing - Large Jet (Cargo)	7	30	30	N/A	N/A	N/A

Table 19. Entity-level sensor signatures (continued)

Entity Category (with example)	Visual	Radar	Infrared	Sonar Active	Sonar Passive	MAD
Military Fixed Wing - Small Jet (Fighter)	4	6	20	N/A	N/A	N/A
Military Fixed Wing - Large Prop	5	25	15	N/A	N/A	N/A
Military Fixed Wing - Small Prop	4	20	15	N/A	N/A	N/A
Military Fixed Wing - Large UAV (Predator)	5	20	16	N/A	N/A	N/A
Military Fixed Wing - Small UAV	2	2	1.5	N/A	N/A	N/A
Military Fixed Wing - Micro UAV (Hand Launch)	1	0.5	0.5	N/A	N/A	N/A
Civilian Fixed Wing - Large (Passenger)	10	40	35	N/A	N/A	N/A
Civilian Fixed Wing - Medium (regional)	7	30	30	N/A	N/A	N/A
Civilian Fixed Wing - Small (Cesna)	5	20	12	N/A	N/A	N/A
Small Rotary Wing	4	20	20	N/A	N/A	0.3
Medium Rotary Wing	5	25	25	N/A	N/A	0.3
Large Rotary Wing	7	30	30	N/A	N/A	0.3
Military/Civilian Large Cargo Ship (tanker/cargo)	35	40	30	10	8	N/A
Military Aircraft Carrier	35	40	30	10	8	N/A
Military Cruiser	25	30	25	8	6	N/A
Military Destroyer	20	28	25	8	6	N/A
Military Frigate	18	25	25	8	6	N/A
Military Large Patrol (Corvette)	18	25	25	6	4	N/A
Military Small Patrol (covered)	10	25	25	3	2	N/A
Military Small (zodiac)	4	6	8	2	1.5	N/A
Civilian Large Commercial Passenger (Cruise)	45	50	50	10	8	N/A
Civilian Medium Commercial	25	40	35	8	5	N/A

Table 19. Entity-level sensor signatures (continued)

Entity Category (with example)	Visual	Radar	Infrared	Sonar Active	Sonar Passive	MAD
Civilian Small Commercial (tug)	15	30	35	4	3	N/A
Civilian Private Medium	15	25	25	4	3	N/A
Civilian Private Small (24->30ft day cruiser)	10	25	25	3	2	N/A
Civilian Private Micro (jet ski)	5	9	10	1	2	N/A
Civilian Sailboat (J-24)	15	20	10	2	0.5	N/A
Small Lifeboat (bright colors)	7	12	15	1	1	N/A
Large Lifeboat (bright colors)	10	15	15	1	1	N/A
Military Submarine - Large (Ohio)	2	3	2	1	0.01	N/A
Military Submarine - Medium (LA)	1.5	3	2	0.75	0.01	N/A
Military Submarine - Small	1.5	3	2	0.5	0.25	N/A
Military Submarine - Torpedo (non decoy)	0	0	0	1	2	N/A
Human (no camouflage)	1.2	0.25	1.2	N/A	N/A	N/A
Human (with camouflage)	0.5	0.25	0.75	N/A	N/A	N/A
Human in Water (with bright vest)	2	2	1.5	N/A	N/A	N/A
Human in Water (no vest)	0.75	2	1.5	N/A	N/A	N/A
Civilian Passenger Car	4	0.5	4	N/A	N/A	N/A
Civilian Truck/Bus	6	1	6	N/A	N/A	N/A
Military Artillery - large	3	0.75	4	N/A	N/A	N/A
Military Artillery - small	1	0.5	2	N/A	N/A	N/A
Military Large vehicle (Logistics)	5	1	5	N/A	N/A	N/A
Military Medium vehicle (Tank)	3	0.5	4	N/A	N/A	N/A
Military Small vehicle (Jeep/Technical)	1	1	2	N/A	N/A	N/A
Military Micro vehicle (human sized or smaller)	0.5	0.25	1	N/A	N/A	N/A

10.1.4 Specifying Sensor Detection Probabilities

Sensor systems can have a detection probability table. If a sensor system does not have a detection probability table, it always detects targets (assuming they meet detection criteria such as being in line-of-sight, within the detection range, are not being jammed, and so on) and the combat identification level (CID) always advances as the apparent signature increases over time. By adding a detection probability table to a sensor, you can add variability to a sensor's ability to detect targets and update information about them.

 The systems provided with VR-Forces do not have detection probability tables.

A detection probability table associates apparent signatures with detection probabilities. (Apparent signature refers to the radar return strength, if the sensor is a radar, or to the apparent brightness of a target on the sensor in question. For more information about apparent signatures, see "10.2. Detection Tables" on page 213.) The detection probability is applied each detection period of the sensor. For radar, the detection period models the sensor sweep, such that the probability is applied individually for each sensor sweep.

The probability of detection affects detection:

- **Initial detection.** If an object is not on the current detection list, the probability of detection is applied after the apparent signature is calculated. If a random draw against the probability passes, the target passes to the CID detection table calculation to determine the CID level. If the random draw fails, then the object is treated as if it was not detected. It is processed again on the next sensor detection run.
- **Continuing detection.** If an object is on the detected list, a random draw is done against the detection probability for its current apparent signature. If this passes, the entity remains detected as normal. Its location and CID level are updated if it is appropriate to do so. If this check fails, the entity remains on the detection list, however the data in the contact information is not updated with any new data, including location, CID level, and so on. On the next sensor detection run, this target is processed again as part of the continuing detection section of the sensor.

The following example shows how you could add a sensor detection probability table to the active-radar-sensor system

(*./data/simulationModelSets/EntityLevel/vrfSim/systems/sensors/active-radar-sensor.sysdef*). (The system included with VR-Forces does not have a sensor detection probability table.)

```
(active-radar-sensor-system
  (systems )
  (sensors
    (radar-sensor
      (component-descriptor-type "radar-signature-sensor-descriptor")
      (component-type "radar-signature-sensor")
      ...
      (detection-period 2.000000)
      (detection-probability
        (signature 1.0
          (probability 0.2)
        )
      )
    )
  )
)
```

```

        (signature 1.5
         (probability 0.2)
        )
        (signature 2.0
         (probability 0.2)
        )
        (signature 2.5
         (probability 0.2)
        )
        (signature 3.0
         (probability 0.2)
        )
        (signature 3.5
         (probability 0.2)
        )
        (signature 4.0
         (probability 0.2)
        )
    )
    ...
    (detection-period-variance 0.5)
    ...
)
)
)

```

10.1.5 Adding Detection Types (Why Doesn't My Sensor Sense Anything?)

To save computation time in VR-Forces, sensors are configured to detect only the objects that the entity needs to know about. If no systems on the entity specify entity types of interest, or the output of the sensor is not connected to anything, the sensor does not attempt to detect simulation objects. In a typical combat entity, a weapon system specifies entity types of interest and a contact-fusion controller accepts sensor outputs and creates a contact list. If an entity does not have similar components, then the sensor will not function.

Most simulation objects included with VR-Forces have a contact-fusion controller, but objects based on platforms such as *Cultural_Feature.ope* typically do not have this controller. (The *Cultural_Feature_With_Radio.ope* file does have the controller.)

In lieu of a weapon system, you can specify object types for the sensor to detect directly in the sensor.

► **To add detection types to a simulation object:**

1. Select the simulation object.
2. For an entity-level simulation object, select the Systems tab.
For an aggregate-level object, select the Sensors tab.
3. Under Sensors, select the system that you want to adjust and then click Edit System Properties (🔧). The properties dialog box opens.

4. Above the Additional Detection Types list, click Add New Object Type (+). The Add New Object Type dialog box opens.
5. For each field of the simulation object type (Kind, Domain, Country, and so on), select an option from the list.

The options are drawn from the SISO Enumerations document. As you select an option, the enumeration at the bottom of the dialog box changes to reflect your choice. You do not need to make a selection for each field because VR-Forces can use wild carding to match simulation objects against the final enumeration.

6. If appropriate, add additional enumerations for each simulation object type that is valid.
7. Click OK. You return to the properties dialog box and the object type is added to the list.
8. Click OK.

 If you are experiencing an issue where an entity configured with a weapon fails to fire on a simulation object, see "11.2.1 Sensing Targets" on page 223.

10.1.6 Adding a New Sensor Domain

You can easily add new sensor domains to your simulation objects. Suppose that you want to add an acoustic sensor. You would:

1. Add a block to the signature-propagator-params block in *physicalWorldParams.mtl*.
2. Add a sensor **componentSystem** element to each entity that can detect targets acoustically.
3. Add a sensor signature to each simulation object that can be detected acoustically.

Configuring a Signature Propagator

The *physicalWorldParams.mtl* file specifies the list of sensors available and the propagator to use for each. This is a portion of that file:

```
(physical-world-params
...
(signature-propagator-params
  (radar
    (propagator-descriptor-type "standard-propagator-descriptor")
    (propagator-type "standard-propagator")
    (check-line-of-sight True)
  )
  (visual
    (propagator-descriptor-type "standard-propagator-descriptor")
    (propagator-type "visual-propagator")
```

```
        (check-line-of-sight True)
      ...
    )
  )
)
```

If you edit the file to add an acoustic sensor, it might look like:

```
(physical-world-params
  (signature-propagator-params
    (visual
      (propagator-descriptor-type "standard-propagator-descriptor")
      (propagator-type "visual-propagator")
      (check-line-of-sight True)
    )
    (acoustic
      (propagator-descriptor-type "standard-propagator-descriptor")
      (propagator-type "standard-propagator")
      (check-line-of-sight False)
    )
  )
  ...
)
```

You do not have to be able to see a simulation object to hear it, so you do not need to check line of sight.

Adding a Sensor Component

To add a new sensor component, follow the procedure in "10.1.1 Adding a New Sensor to a Simulation Object" on page 202. A new acoustic sensor might look like:

```
(acoustic-sensor
  (component-descriptor-type "signature-sensor-descriptor")
  (component-type "signature-sensor")
  (min-tick-period 2.000000)
  (min-tick-period-variance 0.100000)
  (tick-period-uses-real-time False)
  (process-state-repository-name "")
  (process-state-repository-type "")
  (detect-only-hostile-forces False)
  (detection-types )
  (detect-destroyed-objects False)
  (sensor-geometry
    (in-range
      (range $max-range)
    )
  )
  (sensor-domain "acoustic")
  (sensor-offset $sensor-position)
  (sensor-positional-error 0.000000)
  (detection-level-determinator
```

```
(determinator-type "signature-detection-level-determinator")
(detection-level-to-set-hostility 3)
(combat-identification-level-table-file
  "$(detection-dir)\std-acoustic-detection-table.csv")
)
)
)
(controllers
)
(actuators
)
(connections
  (connect system:object-types-to-detect
    acoustic-sensor:object-types-to-detect)
  (connect acoustic-sensor:detected-objects system:detected-objects)
)
)
```

 You can call the sensor whatever you want, but the name must not start with a number.

Adding a Sensor Signature

If you add a new sensor, you need to add a signature for it to each simulation object that can be detected by the sensor. If you add an acoustic sensor, a sensor signature might look like:

```
(sensor-signatures
  (visual
    (base-signature 4.000000)
    (modifiers )
  )
  (acoustic
    (base-signature 2.000000)
    (modifiers )
  )
  ...
)
```

You can probably see simulation objects at a greater distance than you can hear them, so the acoustic sensor has a lower base-signature value than the visual sensor.

10.1.7 Illumination and Time of Day

Visual sensors take into account the time of day. They do this through an illumination value that modifies the visual sensor signature. The modification value is configured in `./data/simulationModelSets/<model_set>/vrfSim/signatureRules/visual-illumination-rules.mtl`:

```
(signature-modifiers
 (environmental
```

```

        (modifier-coefficients 1.0 0.1)
    )
)

```

"10.1.3 Specifying a Sensor Signature" on page 204 shows how this file can be referenced in a sensor signature. Given this file, the modifier coefficients are applied to the illumination value:

$$\text{Multiplier} = 1.0 * \text{illumination} + 0.1$$

The illumination value is derived from the time of day based on parameters in *physicalWorldParams.mtl*:

```

...
(minimal-illumination-value 0.01)
(nighttime-illumination-value 0.02)
...

```

where:

- **minimal-illumination-value** indicates the minimal illumination value in the absence of sunlight and moonlight.
- **nighttime-illumination-value** indicates the nighttime illumination value if the moon is completely full.

10.1.8 Configuring the Effects of Rain on Radar

VR-Forces can take into account the effect of rain on radar sensors. It does so based on the amount of rainfall per hour, the range from the sensor to the target, and the **rain-to-attenuation-coefficient** parameter in the radar block of the *physicalWorldParams.mtl* file in an SMS:

```

(radar
  (propagator-descriptor-type "radar-propagator-descriptor")
  (propagator-type "radar-propagator")
  (check-line-of-sight True)
  (rain-to-attenuation-coefficient 0.05)
)

```

The higher the coefficient value, the more that rain reduces the effective range of the radar. Rain is measured in millimeters per hour, where the maximum rain setting in VR-Forces is equal to 20 mm/h.

10.2. Detection Tables

The detection tables determine the level of knowledge that a simulation object has of detected simulation objects. The detection level is used in the Detect Entity condition in plans.

Detection tables are comma-delimited files that are most easily read in a spreadsheet. As illustrated in Figure 95, each column represents a detection interval, in seconds, from 0 through 160. Each row represents an apparent signature from one through 4. (An apparent signature is a simulation object's absolute signature after being modified by factors such as terrain and distance. For details, see the API documentation.)

Figure 95. Detection table displayed in a spreadsheet

	A	B	C	D	E	F	G	H	I	J	K
1	CID Table-radar	Version 1.0		Detection Interval	(seconds) ->						
2			0	1	5	10	20	40	80	160	
3	Apparent	1	1	1	2	2	2	3	3	3	
4	Signature	1.5	1	1	2	2	3	3	3	3	
5		2	1	1	2	2	3	3	3	3	
6	V	2.5	1	2	2	3	3	3	4	4	
7		3	1	2	3	3	3	4	4	4	
8		4	2	3	3	3	4	4	4	4	

The detection level is determined by finding the intersection of the apparent signature and the detection interval. For example, if a simulation object with an apparent signature of 3 has been detected for 40 seconds, its detection certainty level is 4 (full knowledge).

The apparent signature is the ratio of the sensor signature given for the simulation object (in kilometers) to the actual range. So if the apparent signature = 1, the detection interval shows the increase in combat ID level over time when the target is at the range given by the sensor signature. If the apparent signature is 2, the detection interval shows the detection progression at 1/2 that range, and so on.

The numbers in the table represent the certainty levels:

- 1 – Detected.
- 2 – Classified.
- 3 – Identified.
- 4 – Full Knowledge.

For details about CID detection levels, see "Detecting Targets".



11. Configuring Weapon Systems

This chapter describes configuration files for weapon systems in entity-level scenarios.

11.1. Introduction to Weapon Systems	216
11.1.1 The Weapon System System Definition Template File	217
11.1.2 The Weapon System System Definition File	219
11.2. Configuring Direct Fire Weapons	223
11.2.1 Sensing Targets	223
11.2.2 Configuring Ammunition	224
11.2.3 Hit Probability Tables	226
11.2.4 Damage Probability Tables	227
11.3. Configuring Missile Systems	227
11.3.1 Missile Launcher	228
11.3.2 Missile Projectile	229
11.3.3 Laser-Guided Missile Launcher	229
11.3.4 Laser-Guided Missile Projectile	229
11.4. Configuring Indirect Fire Weapons	230
11.5. Configuring Fixed Weapons	231
11.6. Configuring Bomb Bay Weapon Systems	232
11.7. Configuring Weapon Systems in the SOE	232
11.7.1 Specifying Target Types	232
11.7.2 Configuring Hit Probability	233
11.7.3 Configuring the Ammunition to Use	233
11.7.4 Using the Target Configuration Assistant	235
11.7.5 Configuring the Target Priorities	236
11.7.6 Configuring the Target Region	237
11.7.7 Configuring Missile Launcher Systems in the SOE	237
11.7.8 Configuring Explosive Devices	238

11.1. Introduction to Weapon Systems

Simulation objects in entity-level scenarios have many weapon systems. They all fall into one of these types:

- Ballistic direct fire, such as rifles and the main gun on a tank.
- Self-propelled or guided munitions, such as missiles and rockets.
- Indirect fire, such as artillery, naval guns, mortars, and hand grenades.
- Fixed munition, such as mines and IEDs.
- Bomb bay. A weapon system for dropping bombs.

You can create and edit weapon systems by editing configuration files in a text editor or in the Simulation Object Editor. The preferred method is to use the SOE. It provides a point-and-click interface, handles all syntax issues, and includes error checking to ensure that all components required by a system are present.

Weapon systems are defined by a system definition template (*.template_sysdef*) file that includes component definitions (controllers, actuators, and sensors) and a system definition file (*.sysdef*) that contains system-specific parameter values and targeting information, such as ammunition, hit probabilities, and target priorities. Typically a weapon system template includes at least one controller that controls aiming and firing and an actuator that models the physical characteristics of the weapon. Some weapons create munitions when they fire, so the engagement process involves an additional simulation object with its own movement system, weapon (warhead) system, or both.

All objects with weapon systems can engage using a task such as Fire At Target, Fire For Effect, Throw Grenade, and so on.

Simulation objects often engage targets autonomously. Objects that engage autonomously require a sensor to detect targets. Weapon systems are configured to engage certain target types, and this information is passed to the sensor so that the appropriate enemy objects can be detected.

Automatic engagements are controlled by a target selection controller, which is a component separate from any one weapon system. This controller is specified in the platform (*.ope*) file. Weapons pass information to and from the target selection controller through a weapon interface. This is a process state repository used to keep track of targets for the weapon and the number of rounds to fire at that target. No port is used, no connections are necessary; it is created for you. The target selection controller keeps a list of all weapon systems' weapon-interfaces, one for each system.

Weapon systems generate the fire interactions that are published on the network for an engagement. Detonation actions may be generated by the weapon or by the munition it launches.

Once there is a detonation, damage is determined by the target. Damage may be configured in one of two ways: by specifying the power of the munition and the damage caused to a target by a detonation of a given power, or by specifying damage probabilities for each munition type against each target type. For more information, see "12.1.1 Calculating Damage from Munitions" on page 240.

① Throughout this chapter, we include snippets from system definition files and other configuration files, but usually without explaining what the various parameters mean. If you want to know more about the parameters, view the referenced systems in the Simulation Object Editor. It has tooltips for most parameters.

11.1.1 The Weapon System System Definition Template File

System definition template files (*.template_sysdef*) contain the sensors, controllers, and actuators that are common to a category of weapon or damage system. They contain variable bindings for attributes that are common to the system type and which are given specific values in the individual weapon system and damage system files. They also contain metadata that is used in the SOE.

The following code is a highly edited version of *handheld-launchers.template.sysdef*. It illustrates how components, resources, and metadata are configured in a template file. Notice that the **meta-data** block includes parameter data for the Gun Location and Gun Part Number parameters. These parameters are editable on a per-entity basis when a handheld launcher system is added to an entity (Figure 96).

Figure 96. Properties for AT4 handheld weapon

The number of rounds parameter is a variable binding in the template file and is given a default value in the system files that use this template, for example, *handheld-AT4.sysdef* (discussed in "11.1.2 The Weapon System System Definition File" on page 219).

```
(weapon-handheld-AT4
  (controllers
    (human-acquire
      ...
      (target-acquire-criteria $target-acquire-criteria)
```

```
)
(main-gun-control
  ...
  (load-ammo-time $load-ammo-time (default 5.00000))
  (unload-ammo-time $unload-ammo-time)
  (rounds-per-trigger-pull 1)
  (disabled-by-suppression True)
  ...
  (targeting-control-table $targeting-control-table)
  ...
)
(aim-at-point
  ...
)
(aim-weapon
  ...
)
)
(actuators
  (weapon-state
    ...
  )
  (main-gun
    ...
    (art-part-list
      (primary-gun
        (art-part-type $gun-art-part-type)
        (parent-part-type -1)
        (attach-point $attach-point)
        ...
      )
    )
    )
    (neutral-elevation 0.000000)
    (muzzle-speed $muzzle-speed (default 250.0000))
    ...
    (hit-probability-table $hit-probability-table)
    (range-name $range-name)
    ...
  )
)
)
(connections
  ...
)
(resources
  (M84mm
    (resource-type "integer-resource")
    (amount $num-rounds)
    (full-amount $num-rounds)
```

```

    )
  )
  (meta-data
    (system-name "Handheld Launcher")
    (system-description "Template for system will allow the
      simulation object to be given the ability to add a
      handled launcher to a human.")
    (allowed-state-repository-types "human-param")
    (system-categories "weapon")
    (parameter-data-list
      (art-part-parameter-data
        ...
        (display-label "Gun Part Number")
        ...
      )
    )
  )
  (meta-data-entry-list )
  ...
)
)

```

11.1.2 The Weapon System System Definition File

System definition files provide weapon-specific values for the variable bindings in the template file. They reference the template file that they use so that VR-Forces can use the correct components and make the necessary connections for the weapon system. Weapon-specific system definition files also include a targeting control table and a hit probabilities table.

 This section explains the layout of the weapon-specific system definition file for the benefit of advanced VR-Forces users. However, it is recommended that you edit weapon systems using the Simulation Object Editor.

The targeting control table lists each target type that this weapon can fire on and the ammunition types that it can use for that target. If it can use more than one ammunition type, it lists them from the highest to lowest priority. (For more information about targeting control tables, see "11.2.1 Sensing Targets" on page 223.)

The hit probability table specifies the probability that the weapon can hit a target as a range list or a coefficient list. For more information about hit probability tables, see "11.2.3 Hit Probability Tables" on page 226.

- ① By default the metadata in a template file is merged with the metadata in the weapon-specific system file for use in the SOE. If you do not want a system file to merge its metadata with the template file, add the **merge-with-sub-systems** variable to the metadata and set it to False, for example:

```
(meta-data
  (system-name "Light Armor damage model")
  ...
  (meta-data-entry-list )
  (ui-filename "")
  (merge-with-sub-systems False)
)
```

This example is an edited version of *handheld-AT4.sysdef*, which references *handheld-launchers.template.sysdef*. Notice how it provides values for the **load-ammo-time** and **unload-ammo-time** variables.

```
(weapon-handheld-AT4
  (systems
    (handheld-launcher
      (variable-bindings
        (DtRWReal
          (load-ammo-time 5.000000)
        )
        (DtRWReal
          (unload-ammo-time 0.000000)
        )
        (DtRWBoolean
          (velocity-affects-hit-chance True)
        )
        (DtRWReal
          (max-skill-level-effect 0.250000)
        )
        (DtTargetAcquireCriteria
          (target-acquire-criteria
            (entity-type 0 0 0 0 0 0 0)
            (time-to-acquire 0.000000)
            (check-maximum-speed-to-acquire True)
            (max-speed-of-target 10.000000)
            (check-elevation-angle-to-acquire False)
            (max-elevation-angle-to-acquire 1.500000)
            (check-elevation-to-acquire False)
            (max-elevation-to-acquire 500.000000)
          )
          ...
        )
      )
    )
    (DtTargetingControlTable
      (targeting-control-table
        (ammunition-list
          (M84mm
```

```
        (amount $num-rounds)
      )
    )
    (entity-target-priorities
      (priority-entry
        (target-type 1 1 -1 1 -1 -1 -1)
        (priorities "M84mm")
      )
      (priority-entry
        (target-type 1 1 -1 -1 -1 -1 -1)
        (priorities "M84mm")
      )
    )
    ...
  )
  (max-targets 1)
  (target-region
    (and
      (not
        (in-range
          (range 30.000000)
        )
      )
      (in-range
        (range 300.000000)
      )
    )
  )
  (fire-with-spot-report-only False)
  ...
)
)
(DtRWReal
  (muzzle-speed 285.000000)
)
(DtRangeProbabilityTable
  (hit-probability-table
    (entity-range
      (entity-type 1 1 -1 -1 -1 -1 -1)
      (range-determinant
        (step-function
          (range 100.000000
            (probability 0.90000)
          )
          (range 200.000000
            (probability 0.5000)
          )
          (range 500.000000
            (probability 0.30000)
          )
          (range 750.000000

```

```
        (probability 0.10000)
      )
    )
  )
  ...
)
)
)
(system-definition
  (filename "$(system-dir)\weapons\templates\handheld-
launchers.template_sysdef")
)
(is-enabled True)
(part-of-roll-up False)
(use-parent-system-resource-manager True)
)
)
(connections
  (connect system:all handheld-launcher:all)
  (connect handheld-launcher:all system:all)
)
(resources )
(meta-data
  (system-name "AT4")
  (system-description "Handheld AT4 Anti-Tank weapon. Targets
ground
vehicles and helicopters.")
  (allowed-state-repository-types "human-param")
  (system-categories "weapon" "aimable-weapon")
  (parameter-data-list
    (int-parameter-data
      (parameter-name "num-rounds")
      (variable-type "DtRwInt")
      (display-label "Number of Rounds")
      (display-units "")
      (source-units "")
      (list-items "")
      (list-display 0)
      (allow-edit True)
      (display-value True)
      (roll-up-rule "sum")
      (default-value 1)
    )
  )
  (meta-data-entry-list )
  ...
)
...
)
```

11.2. Configuring Direct Fire Weapons

Entities that have direct fire weapons can fire in response to an engagement task for firing their weapons, and they can also detect targets through their sensors and automatically fire at the targets. Entities have to be able to:

- Know what targets they can fire at.
- Detect the targets.
- Determine the ammunition to use.
- Acquire the target and fire at it.

If you want to create a completely new type of weapon system, you must write code for the sensors, controllers, and actuators that it requires. However, if you need a system that is similar to an existing system, you can often create the new system by copying and modifying the configuration of the existing system. Although you can do so by directly editing configuration files, the recommended way to add and edit weapon systems is to use the Simulation Object Editor. To do so, you must understand how to:

- Configure a weapon system.
- Specify the simulation object types that can use the system.
- Specify the simulation object types that can be damaged by the system.
- Specify the amount of damage a simulation object receives from a munition.

The files that manage these relationships are:

- The weapon system template. The template configures the components common to similar types of weapons, such as ballistic guns or missile launchers.
- The weapon system. Defines a specific system, such as a 30 mm gun, the simulation object types that it can target, ammunition types it can use, hit probabilities, and the simulation object types that can use it.
- The damage system. Specifies the types of munitions that a simulation object is vulnerable to and the damage file to use to determine the probability of damage.

A direct fire weapon:

1. Acquires the target.
2. Selects the ammunition for the target using the ammo select file.
3. Checks that it has that resource.
4. Determines if it hits the target.
5. Sends a fire interaction.
6. After a delay based on distance and muzzle speed, it sends a detonation interaction.

11.2.1 Sensing Targets

Simulation objects that have direct fire weapons will detect targets through their sensors and automatically fire at the targets. Sensor systems can specify what types of targets to detect (using the **detection-types** parameter). The weapon systems can also tell the

sensors what they want to target.

 You can also edit target types in the Targets tab. See "11.7.1 Specifying Target Types" on page 232.

Each weapon system has an associated weapon interface. The **targeting-control** parameter takes a list of desired target types and creates a target selection criteria object, which is then set in the weapon interface. This code shows the targeting control for the *120mm-gun.sysdef* file.

```
(DtTargetingControlTable
  (targeting-control-table
    (max-targets 1)
    (target-region
      (in-range
        (range 3000.000000)
      )
    )
  )
  (fire-with-spot-report-only False)
  (ammunition-list
    (M829A1-AP-120mm
      (amount $num-ap-rounds)
    )
    (M830-HEAT-120mm
      (amount $num-heat-rounds)
    )
  )
  (entity-target-priorities
    (priority-entry
      (target-type 1 1 -1 1 -1 -1 -1)
      (priorities "M829A1-AP-120mm" "M830-HEAT-120mm")
    )
    (priority-entry
      (target-type 1 1 -1 2 -1 -1 -1)
      (priorities "M829A1-AP-120mm" "M830-HEAT-120mm")
    )
    ...
  )
)
```

The object's weapon management controller takes the target entity type information from the weapon interfaces and provides it to the sensor systems.

11.2.2 Configuring Ammunition

The targeting control table ammunition-list block specifies the ammunition types that a weapon system can use. Each priority entry in the table specifies a target type (using the best match method) and a space-separated prioritized list of ammunition that should be

used for that type of target (highest priority listed first). (The best match method is described in "The Best Match Method" on page 169. For an example of a targeting control table, see "11.2.1 Sensing Targets" on page 223.)

The ammunition types that you can include in a weapon system's ammunition-list are specified in `./data/simulationModelSets/<sms>/ammunitionParams.mtl`. The Simulation Object Editor references this file when it displays the lists of ammunition that you can add to a weapon system. Each ammunition type entry in `ammunitionParams.mtl` specifies a set of parameters that define its characteristics and how it is represented in the Simulation Object Editor:

- **munition-type** – 7-digit object type enumeration. This is the object type that is used in the fire and detonation interactions created, and should NOT contain -1s.
- **tracer-type** – 7-digit object type enumeration. This parameter is optional. Use it if you want a weapon to support tracer rounds.
- **warhead** – VR-Link warhead enumeration value to use in fire and detonation interactions for this munition.
- **guidance-mode**. How the munition is guided, if applicable. Options are:
 - 0 - None.
 - 1 - Laser.
 - 2 - Optical Wire.
 - 3 - Radar.
 - 4 - Thermal.
- **gui-label**. The string used to identify the munition in the Simulation Object Editor.
- **targets**. Object type enumerations for the simulation objects that the weapon can target.
- **tags**. The tags used for filtering the munition in the Simulation Object Editor object lists.

This code shows two entries in `ammunitionParams.mtl`, a missile and a ballistic munition:

```
(spiral-missile
  (munition-type 2 2 222 1 8 0 0)
  (warhead 0)
  (guidance-mode 0)
  (gui-label "AT-6 Spiral Missile")
  (targets
    (target-type 1 1 -1 -1 -1 -1 -1)
    (target-type 1 3 -1 11 1 -1 -1)
  )
  (tags "3D Model" "Military" "Munition")
)
(AT-30mm
  (munition-type 2 2 222 2 2 0 0)
  (tracer-type 2 2 222 2 2 0 0)
  (warhead 0)
  (guidance-mode 0)
  (gui-label "AT-30mm")
)
```

```
(targets
  (target-type 1 1 -1 -1 -1 -1 -1)
)
(tags "mm gun")
)
```

11.2.3 Hit Probability Tables

A weapon system definition file includes a hit probability table. This table is used by the system's actuator to determine whether it hits a target that it fires upon. A hit probability table has one or more **entity-range** entries. The name of the **entity-range** entry is not used, and can be something descriptive. An **entity-range** entry has the form:

- **entity-type** – 7 digit entity type enumeration, which may use -1s as wildcards.
- **range-determinant** – a range determinant can be either a list of probability-range entries with the name "range-list", or a probability-coefficient list with the name "coefficients".
- **range-list** – a list of probability-range entries. A probability-range entry has a name, "range", a range value (in meters), and a list of probability entries. The entry applies to targets in the range from the last entry to the current one (or from 0.0 in the case of the first entry). Each probability entry has a name that the table user uses to find the probability value it is interested in and the value itself (a real number between 0.0 and 1.0.) The ballistic gun looks for the probability entry with the name "probability".



- For ranges that are greater than the last specified range, the probability is zero.
- The range for missiles is the range from the missile to the target, not from the entity that fired the missile.

- **coefficients** – a coefficient probability list is a list of coefficient probability entries. Each entry has a probability name and the coefficients of a standard polynomial that can be used with a given range to determine the specified probability. The order of the polynomial is determined by the number of coefficients provided. For example, an entry of:

```
(probability -5.00000E-008 -300000E-012 1.0000)
```

indicates that the formula to use for a given range r would be:

$$(-5.00000E-008 * r^2) + (-300000E-012 * r) + 1.0$$

Values calculated will be constrained to be between 0.0 and 1.0. Zero values are factored out, for example:

```
(probability -5.00000E-008 0.0000 1.0000)
```

indicates that the formula to use for a given range r would be:

$$(- 5.00000E-008 * r^2) + (0.0 * r) + 1.0$$

This is an example of the hit probability table for the handheld-AK47 system:

```
(DtRangeProbabilityTable
  (hit-probability-table
    (entity-range
      (entity-type 3 -1 -1 -1 -1 -1 -1)
      (range-determinant
        (range-list
          (range 10.000000
            (probability 1.00000)
          )
          (range 200.000000
            (probability 0.9000)
          )
          (range 550.000000
            (probability 0.70000)
          )
          (range 3600.000000
            (probability 0.01000)
          )
        )
      )
    )
  )
)
```

11.2.4 Damage Probability Tables

The damage actuator uses damage probability tables specified by its component descriptor to determine direct and indirect fire damage. Damage tables are in the damage systems in *./data/simulationModelSets/<sms>/vrfSim/systems/damage*. A damage probability table has a list of surface entries, which provide a sub-table for each surface upon which a round might detonate. The name of the surface entry is the surface name used in the damage model.

For the damage-by-power damage system, VR-Forces has damage probability tables for each power type. For more information, see "Configuring the Damage File for Damage-By-Power" on page 245.

11.3. Configuring Missile Systems

Missiles are launched by an object's missile launcher system, which creates a missile entity. The missile flies itself and detonates based on the use of a hit probability table or on the type of missile.

If a missile launcher system definition specifies a hit probability table, when the missile is created, VR-Forces checks to see if the object type of the target is in the table. If it is, a random number is drawn and compared to the range-based probability. If it is determined that the missile should hit the target, the missile begins its normal fly out. If it collides with the terrain or another entity before reaching the target, it detonates and does not hit

the target. However, if the missile does not collide with anything and hits or passes the target, it automatically counts as a hit. The missile detonates on the entity. The SA-21 missile launcher and Patriot missile launcher use hit probability tables.

If a missile launcher system definition has a hit probability table, but the target object type is not listed, or if the system does not specify a hit probability table, detonation is determined based on the type of missile:

- Direct flight. Missiles fly directly to a target. Accuracy is controlled by their maneuverability parameters.
- Laser guided. Missiles follow a laser designator to the target. If the laser is blocked or disrupted, the missile goes to the last known location of the target.
- Route following. Some missiles, such as cruise missiles, can follow a route to a target and detonate either at the target or at a proximity to the target.

 The range for missiles is the range from the missile to the target, not from the entity that fired the missile.

11.3.1 Missile Launcher

The missile launcher for direct flight missiles works as follows:

1. The target selection controller chooses the best target for the weapon system and passes it to the weapon through the weapon's weapon-interface.
2. The weapon system's controller determines the right resource for that target type.
3. The missile resource is returned and the system loads the missile to fire.
4. The entity tries to acquire the target. It sends the desired rotation angle and elevation to acquire the target to the rotating mount actuator and elevation mount actuator.
5. Actuators move according to their slew rate speeds and send how far they went in a tick back to the weapon launcher so it can recalculate the remaining amount. It also publishes the articulated part numbers along with amount of rotation/elevation to the network.
6. When the feedback reaches the max angle off boresight, it can fire on the target. The missile object is created.
7. Firing state is sent to the launcher actuator.
8. The launcher actuator sends a fire interaction.

Parameters in the missile launcher's system affect whether it fires. For example, the Maverick Missile Launcher system includes these parameters on the Attributes tab in the Simulation Object Editor:

- Load Ammo Time
- Unload Ammo Time
- Max Vehicle Speed to Fire

- Max Azimuth Angle Off Boresight
- Max Elevation Angle Off Boresight

11.3.2 Missile Projectile

When the missile is created, it flies as follows:

1. Using the missile's movement parameters, the missile controller (track-entity if the target is an entity, or target-point if the target is a location) determines the values to be sent to the missile-maneuver actuator.
2. The missile-maneuver actuator calculates the new position, velocity, acceleration, and orientation of the missile.
3. The missile-collision actuator then checks to see whether the missile will collide with the terrain or an entity. If so, it sends out a detonation interaction.

Missile parameters affect how it detonates. The following code is from

`./data/simulationModelSets/EntityLevel/vrfSim/systems/weapons/missile-warhead.sysdef:`

```
(actuators
  (warhead
    ...
    (detonate-on-ground-impact True)
    (detonate-on-water-impact True)
    (detonate-on-entity-impact True)
    ...
    (allow-direct-hit True)
  )
)
```

11.3.3 Laser-Guided Missile Launcher

The missile launcher for laser-guided missiles works as follows:

1. The target selection controller chooses the best target for the weapon system and passes it to the weapon through the weapon's weapon-interface.
2. The weapon system's launcher controller tells the lasing controller the targets.
3. The lasing controller looks for an actively lased target that matches the lasing code. If there are no lased targets, it checks to see whether it can autonomously lase. If it can, it starts to lase the target.
4. Once the launcher controller has a lased target, it tells the movement system to rotate to the target.
5. The launcher controller checks for when the entity has rotated to line up with target, within the fire cone defined for it. It creates the missile object and fire interaction.

11.3.4 Laser-Guided Missile Projectile

Once it is created, a laser-guided missile flies as follows:

1. Using the missile's movement parameters, the missile controller determines the values to be sent to the missile-maneuver actuator.
2. The missile-maneuver actuator calculates the new position, velocity, acceleration, and orientation of the missile.
3. The missile-collision actuator then checks whether the missile will collide with the terrain or an entity. If so, it sends out a detonation interaction.

11.4. Configuring Indirect Fire Weapons

Indirect fire weapon systems use the indirect-fire-weapon-controller. Indirect fire weapons fire in response to an indirect fire task, such as Fire-for-Effect or Throw Grenade. They do not fire on their own, like many direct fire weapons do.

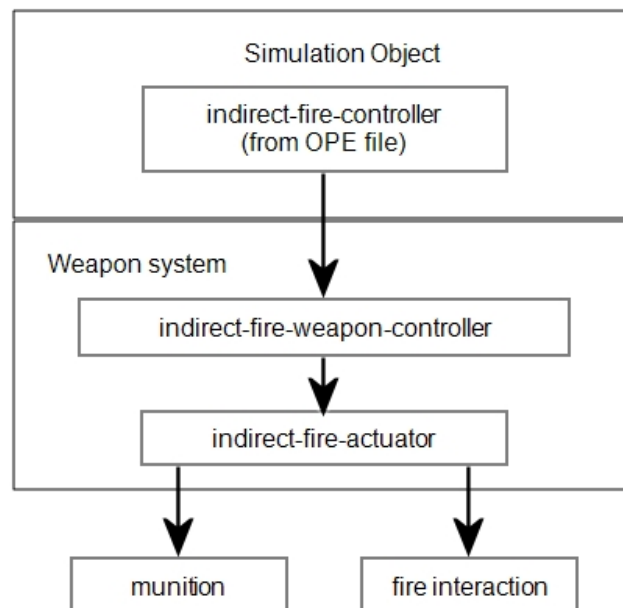
When a simulation object receives an indirect fire task, its indirect-fire-controller queues the task and checks for availability of an indirect fire weapon. It continues to check until a weapon becomes available, then it sends the fire task to the weapon through the indirect weapon interface. If a task requires multiple weapons, the indirect-fire-controller continues this process until the task is complete. (The indirect-fire-controller and indirect weapon interface are analogous to the target-select-controller and weapon interface described for direct fire weapons.)

When an indirect-fire-weapon-controller receives a task, it reads the fire mission from the indirect weapon interface. The indirect fire weapon controller tries to align the turret and load the munition. When the weapon is ready to fire, it passes the information to its indirect-fire-actuator.

The munition is specified as part of the task. The indirect-fire-actuator creates the desired munition and sends out the fire interaction.

Figure 97 illustrates this process.

Figure 97. Indirect fire



i Do not confuse the indirect-fire-controller and the indirect-fire-weapon-controller. The indirect-fire-controller is on the simulation object that has the weapon system (it is in the platform OPE file). The indirect-fire-weapon-controller is in the indirect fire weapon system. The controllers are illustrated in Figure 97.

Examples of indirect fire weapons include the M284-155 mm cannon (`./data/simulationModelSets/EntityLevel/vrfSim/systems/weapons/M284-155mm-cannon.sysdef`), the M252-81 mm mortar (`M252-81mm-mortar.sysdef`), and MK45 naval gun (`./data/simulationModelSets/EntityLevel/vrfSim/systems/weapons/MK45NavalGun.sysdef`). Also, although hand grenades would not normally be thought of as indirect fire weapons, the hand grenade tasks call the fire for effect tasks. Therefore, they use the same controllers and actuators as indirect fire weapons.

The munitions used by the indirect fire weapon systems are configured with explosive power in *detonationParams.mtl*. Therefore simulation objects that are susceptible to explosive power can be damaged by them. Their damage system definition files will point to the appropriate damage table files to assess damage from these munitions.

As with other weapon systems, the system definition files for indirect fire weapons have parameters that control when they can be used and how effective they are. You can set the values for these parameters in the Simulation Object Editor on the Attributes and Firing tabs. The Targets tab only has entries for ammunition, since there is no need for hit probability and target selection for indirect weapons.

11.5. Configuring Fixed Weapons

Fixed munitions, such as mines and IEDs, can be triggered:

- Immediately, by a detonate message.
- By proximity of a valid target. The weapon needs a sensor system or a proximity controller that can detect targets.
- Timed detonation, from when the system is armed.

Fixed munitions have a detonation controller. The detonation controller processes set detonation messages to determine how it is to be triggered and whether it is armed. It tells the warhead-detonation-actuator when to arm the weapon. For immediate and timed detonations, it tells the warhead actuator to detonate the device.

For proximity detonations, the detonation controller takes input from the external sensor or the proximity sensor. The warhead-actuator checks for armed and trigger messages from the detonation controller. It also checks for collisions with the terrain and simulation objects. If a collision occurs or it is triggered by the detonation controller, it sends a detonation interaction over the network.

VR-Forces has two types of fixed weapons: naval mines and IEDs.

IEDs are simulation objects that have an explosive device system (`./data/simulationModelSets/EntityLevel/vrfSim/systems/weapons/explosive-device.sysdef`). These objects need a sensor to detect targets. The Roadside IED object has a visual sensor. You can specify the ammunition, targets to detect, and the type of fuse on the Targets tab for the system in the Simulation Object Editor.

Naval mines use the

`./data/simulationModelSets/EntityLevel/vrfSim/systems/weapons/naval-mine-explosive-device.sysdef` system definition file. It has a proximity sensor built into the system definition, so it does not need an external sensor to detect targets. You can specify the ammunition, targets to detect, and the type of fuse on the Targets tab for the system in the Simulation Object Editor.

You can turn an entity into an IED by adding the explosive device system in the Simulation Object Editor. You can configure cultural features as IEDs by adding the Other Explosive Device system.

11.6. Configuring Bomb Bay Weapon Systems

The bomb-bay weapon system releases bombs in response to Release Bomb tasks. Bombs can drop directly onto targets or be laser guided. Once a bomb is released, it uses the `./data/simulationModelSets/EntityLevel/vrfSim/systems/movement/smart-bomb-dynamics.sysdef` movement system to control movement. It has a warhead actuator that is similar to the warhead actuator in missiles. Its ability to navigate toward and hit its target is controlled by parameters that you can set in the Simulation Object Editor. The Simulation Object Editor also allows you to specify the ammunition type for bomb bay systems. For more information about bombs, see "Release Bomb Tasks".

11.7. Configuring Weapon Systems in the SOE

Like other systems, weapon systems have a General tab for identification information and one or more tabs for physical parameters. Targeting is configured on the Targets tab. When you configure a direct fire weapon system, you must specify:

- The probability that the weapon will hit its target.
- The ammunition it should use.
- Target priorities.
- Target region.

The following sections explain each aspect of target configuration. However, sometimes it can be confusing to understand how to configure the minimum requirements for a target type. The SOE has two features that help you with this process — the Target Configuration Assistant and the Configuration Status window. For details, see "11.7.4 Using the Target Configuration Assistant" on page 235.

11.7.1 Specifying Target Types

Weapon systems have one or more target types that they can fire upon. You can add, edit, and remove target types. For each target type, you must specify Hit Data. You might also need to specify an ammunition type that is valid for the target.

► **To add a target type:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.

3. Click the Add button (+) next to the Target Type label. The Add New Target Type to Hit Table dialog box opens.
4. Specify an object enumeration for this target type.
5. Select a creation option:
 - Copy Currently Selected. Use the same type of hit table as the currently selected target.
 - Create Empty Range Table. Create a range list table for configuring how this weapon can fire on the new target.
 - Create Empty Coefficient Table. Create a coefficient table for configuring how this weapon can fire on the new target.
6. Click OK. The target is added to the Target Type list and a Hit Data table is created based on the selected option.
7. Check the Configuration Status window to see if there are any configuration issues as a result of adding this target type.

11.7.2 Configuring Hit Probability

You can configure hit probability using a range list or a coefficient list. You can configure a range list by editing the table or by editing the graph directly. You configure a coefficient list by editing the table.

► **To edit a hit probability table:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Target Type window, select the Target Type that you want to configure.
4. Edit the probabilities as described in "3.9.2 Editing Probability Tables" on page 68.

Changing the Hit Probability Based on Skill Level

VR-Forces users can change the skill level of entities with the Skill Level set data request. The skill level can be novice, experienced, or expert. The default skill level is expert. If the skill level of an entity is set to novice or experienced, the hit probability of the entity's weapon changes based on the **Maximum Skill Level Effect** parameter for the weapon system. For example, if the **Maximum Skill Level Effect** is set to .20 (20%), then the probability can shift as much as 20% for a novice, 10% for an experienced entity, and 0% for an expert.

For direct fire ballistic weapons, you can set **Maximum Skill Level Effect** on the Attributes tab for the weapon system. For indirect fire weapons (artillery and mortars), you set **Maximum Skill Level Effect** on the Firing tab.

For details about setting the skill level of an entity, see "Skill Level".

11.7.3 Configuring the Ammunition to Use

You can specify the types of ammunition that a weapon system can use. You also specify the default amount. Some weapon systems can use more than one type of ammunition.

► To add a type of ammunition for a weapon system:

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. Click the Add button (+) above the Ammunition window. The Choose Ammunition dialog box opens. It lists ammunition types that may be appropriate for this system. Ammunition types that are not supported for this weapon system have an alert icon next to their name. You can add them to the ammunition list, but they will not have any effect on targets unless you also configure them to work with the weapon system.
4. Select the ammunition type that you want to add.
5. Click OK. The ammunition is added to the list in the window.

Adding a New Ammunition Type from the Targets Page

You can add new ammunition types on the Munitions tab. However, you can also start the process from the Choose Ammunition dialog box that you open from the Targets tab.

► To add a new ammunition type:

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. Click the Add button (+) above the Ammunition window. The Choose Ammunition dialog box opens.
4. Click Add New from Existing. The Create New Model dialog box opens.
5. Enter a name for the new ammunition type.
6. On the Object Type line, click Edit. The Change Object Type dialog box opens.
7. Specify the object type for the new ammunition by selecting options from the lists or directly entering the object type.
8. Click OK. The new ammunition type is added to the list.
9. Select the Munitions tab.
10. Select the new ammunition type that you added.
11. Specify the parameters for this munition.

Removing an Ammunition Type**► To remove an ammunition type from a weapon system:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Ammunition window, select the row for the ammunition type that you want to remove.
4. Click the Remove button (✖) .

11.7.4 Using the Target Configuration Assistant

You can specify a target type, hit probabilities, and ammunition independently, as described in the previous sections, or you can specify them within one dialog box. This ensures that you have made the minimum configuration necessary for the weapon system to successfully fire on a target type.

► **To use the Target Configuration Assistant**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Hit Probabilities section, click the Target Configuration Assistant button. The Target Configuration Assistant opens (Figure 98).

Figure 98. Target Configuration Assistant

Target Configuration Assistant

1. Select a Target

Kind: 1 - Platform

Domain: All

Country: All

Category: All

Subcategory: All

Specific: All

Extra: All

Object Type: 1:-1:-1:-1:-1:-1 Select Specific Simulation Object

2. Enter Hit Probability

Maximum Range (m): 5000

Hit Probability (%): 30

3. Select Ammunition

- 9mm
- AKA-762mm
- API-14.5mm
- AT-30mm
- BM-14m-HEAT-125mm
- BM-9-AP-125mm
- Human Explosive Vest
- M15 AT Mine
- M16 AP Mine
- M16A2-556mm
- M21 AT Mine
- M2-12.7mm
- M433 40mm HEDP
- M433 40mm HEDP for use in auto weapons
- M72ADAM AP Mine

Add New from Existing OK Cancel

4. Select a target type. You can build an object type by selecting options from the lists, directly entering the object type, or selecting a specific simulation object as the target.
5. In the Maximum Range box, enter the maximum distance at which the weapon can hit the target. For range lists, the SOE automatically divides the range for the hit probability table into three segments.
6. In the Hit Probability box, enter the probability of hitting the target at the maximum range.

7. In the ammunition list, select the ammunition to use for this target type. The window lists all possible ammunition types. If an ammunition type is not configured to fire on the target type, it is preceded by an alert icon (as illustrated in Figure 98). You can still add this ammunition type, but you would need to configure it to fire on the target type or it will not work.
8. Click OK. The target type, hit probability, and ammunition are added to the page.
9. Check the Configuration Status window to see if there are any issues for this target type.
10. Optionally, specify weapon target priorities and the target region.

11.7.5 Configuring the Target Priorities

Weapons can only fire at objects that have been configured as possible targets. Within that set of possible targets, you can assign priorities. When the target controller identifies more than one possible target, it chooses its target based on the priorities.

You can specify the types of objects that a weapon can target and the priorities of those target types. For each target type, you can specify the preferred ammunition to use.

► **To add a new target priority:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. Click the Add button (+) above the Weapon Target Priorities window. The Add New Target Priority dialog box opens.
4. Specify the object type for the target you want to add.
5. Click OK. The target is added to the bottom of the list.

Changing a Target Priority's Object Type

You can change the object type of a target priority.

► **To change the object type of a target priority**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Weapon Target Priorities window, select the target priority you want to change.
4. Click the Edit button (✎) above the Weapon Target Priorities window. The Select Target Type dialog box opens.
5. Change the object type.
6. Click OK.

Changing the Priority of a Weapon Target

Weapon targets are listed in the Weapon Target Priorities window in order of priority, from highest to lowest. You can change the priority of a weapon target by changing the order in which it is listed.

► **To change the priority of a weapon target:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Weapon Target Priorities window, select the target whose priority you want to change.
4. Click the Up Arrow above the window to raise the target's priority. Click the Down Arrow to lower its priority.

11.7.6 Configuring the Target Region

You can configure a target region for a weapon system. (Many weapon systems do not have a value for this parameter.) If you specify a target region, the weapon system will not fire at a target unless it is within or without the region. For example, a visual sensor might detect a target at 500 meters, but you do not want the weapon to fire until the target is within 100 meters.

The target region can be specified as a range, a frustum, or both. You can specify a minimum range, maximum range, or both.

► **To specify the target region for a weapon system:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. To specify the target region as a range:
 - a. Select the Minimum Range check box.
 - b. Enter a value for the minimum range.
 - c. Select the Maximum Range check box.
 - d. Enter a value for the maximum range.
4. To specify the target region as a frustum:
 - a. Select the Use Frustum check box.
 - b. Specify the maximum and minimum values for the azimuth and elevation.
 - c. Optionally, select the Outside Frustum check box, to specify the that the target region is outside of the frustum.

11.7.7 Configuring Missile Launcher Systems in the SOE

VR-Forces has several types of missile launcher systems. The types of missiles that they use determines the parameters that must be set on the Targets tab.

- Laser-guided missile systems configure ammunition (as described in "11.7.3 Configuring the Ammunition to Use" on page 233) and weapon target priorities (as described in "11.7.5 Configuring the Target Priorities" on the previous page).
- Cruise missile launchers configure ammunition.

- Other missile launcher systems can use hit probability tables or they can calculate hits based on the type of missile, as explained in "11.3. Configuring Missile Systems" on page 227. When you configure a missile system, you specify how to calculate hit probability by choosing one of these options:
 - Simulate Missile Guidance. Do not use a hit probability table. Calculate based on missile type.
 - Use Probability Table. Configure the launcher like a direct fire weapon system. Specify target type, hit data, ammunition, weapon target priorities, and target region.

11.7.8 Configuring Explosive Devices

Explosive devices, such as IEDs and naval mines, must specify the ammunition type, detonations parameters, and objects to detect.

► **To configure an explosive device weapon system:**

1. Select the weapon system you want to edit.
2. Select the Targets tab.
3. In the Detonation Proximity Distance box, specify the distance to use for proximity detonation.
4. Select any of the check boxes that apply for types of impact this device will support.
5. Specify the ammunition, as described in "11.7.3 Configuring the Ammunition to Use" on page 233.
6. Specify the object types that this device can detect for proximity detonation.



12. Configuring Damage Systems

This chapter describes configuration files for damage systems in entity-level scenarios.

12.1. Introduction to Damage Systems	240
12.1.1 Calculating Damage from Munitions	240
12.2. Configuring Damage-by-Power	240
12.2.1 Configuring the Power of a Detonation	241
12.2.2 Configuring Detonation Power in the SOE	244
12.2.3 The Damage File for Damage-By-Power	245
12.2.4 Adding a New Power Type	247
12.3. Configuring Damage in the SOE	249
12.3.1 Configuring Damage Power Entries	249
12.3.2 Configuring Impact Surfaces	249
12.3.3 Configuring the Angle of Incidence	251
12.3.4 Specifying Effect Data	251
12.4. Viewing Inter-Object Damage Probability Results	252
12.5. Configuring Damage for Dynamic Terrain	254
12.5.1 Standard Terrain Damage	254
12.5.2 High Fidelity Terrain Damage	254
12.6. Configuring Detonation Results	256

12.1. Introduction to Damage Systems

Like weapon systems, damage systems are based on high-level `template_system` files and lower level system definition files that reference the templates. The template files specify the damage actuators and metadata. The system definition files specify the damage to entity surfaces by the different power types that can damage the entity.

You can create and edit damage systems by editing configuration files in a text editor or in the Simulation Object Editor. The preferred method is to use the SOE. It provides a point-and-click interface, handles all syntax issues, and includes error checking to ensure that all components required by a system are present.

Like other systems, damage systems have a General tab and Attributes tab for identification parameters and other default parameters. The damage done to the entity is configured on the Damage tab.

12.1.1 Calculating Damage from Munitions

VR-Forces calculates damage using damage-by-power.

Each munition is configured for its power in the explosive, kinetic, fragmentation, and armor-piercing domains as appropriate. (You can add new domains without writing code.) Damage systems (`.sysdef` files) contain entries for the types of munition power that can affect simulation objects that use the system. They configure the amount of damage that occurs when different levels of power are applied. The damage-by-power model is used for all entity-level simulation objects in VR-Forces 4.5 and later and is the preferred method for calculating damage.

12.2. Configuring Damage-by-Power

In the damage-by-power method of calculating damage, each munition has a `detonation-power-entry` block in `./data/simulationModelSets/base/detonationParams.mtl`. The following is the entry for the 7.62 mm bullet:

```
(detonation-power-entry
  (munition
    ;; 7.62 mm
    (munition-type 2 8 -1 2 2 -1 -1)
    (warhead -1)
    (guidance-mode 0)
  )
  (power-list
    (kinetic
      (direct 2.24) ;; kilojoules
    )
  )
)
```

The `munition` block identifies the munition type using the DIS/RPR FOM object enumeration. (The default configuration does not use the **warhead** or **guidance-mode** parameters.)

The `power-list` block is a list of domains and the configuration for a determinant that produces a power value based on an input range. In this example, the 7.62 mm shell has power in the kinetic domain.

The default configurations support these domains:

- Explosive. Munitions that affect simulation objects within some range of the detonation. Explosive munitions represent power in kiloPascals (kPa) of incident pressure.
- Kinetic. Munitions that directly strike a simulation object. Kinetic munitions represent power in kilojoules (kJ) of energy.
- Armor-piercing. Munitions that strike an armored object and penetrate it. Armor piercing munitions represent power in millimeters of penetration.
- Fragmentation. Explosives that cause damage by fragmenting, such as grenades.

 You can add other domains without writing code. For details, see "12.2.4 Adding a New Power Type" on page 247 .

The power values used in the default configuration are based on publicly available information about the power of the various types of munitions and the values used try to be consistent in their relative power.

12.2.1 Configuring the Power of a Detonation

You can configure detonation power by editing *detonationParams.mtl* or in the Simulation Object Editor. This section explains how to edit the file directly and what the parameters mean. For details about editing detonation power in the SOE, see "12.2.2 Configuring Detonation Power in the SOE" on page 244. To determine the power of a detonation at a specific location a power determinant must be defined. The determinant maps a range value to a power value. Range can have different possible meanings, depending on how the **range-from** parameter is defined for the determinant. If **range-from** is undefined, **impact-point** is assumed unless the determinant does not support this setting.

- If **range-from** is **impact-point**, the determinant determines power based on the range from the impact point of the detonation to some other location in the vicinity. This is used for the explosive and fragmentation domains, but could be used for any power that has some area of effect (for example, perhaps shrapnel or incendiary damage). This is equivalent to collateral damage as defined in the damage-by-ammo-type damage system.

Example (from *detonationParams.mtl*):

```
(detonation-power-entry
  (munition
    ;; 85 mm HEAT
    (munition-type 2 2 222 2 8 1 -1)
    (warhead -1)
    (guidance-mode 0)
  )
  (power-list
```

```
(explosive
  (range-from "impact-point")
  (range-list
    (range 3.5
      (power 70.0)
    )
    (range 4.6
      (power 40.0)
    )
    (range 5.7
      (power 27.5)
    )
    ...
    (range 35.0
      (power 2.4)
    )
  )
)
(kinetic
  (range-from "shooter-to-impact")
  (range-coefficients -2.6600E-004 0.0000 5319.0)
)
(armor-piercing
  (range-from "shooter-to-impact")
  (range-coefficients -9.0000E-006 0.0000 180.0)
)
)
```

- If **range-from** is shooter-to-impact (as in the previous example), the determinant determines power based on the range from the entity that fired the round to the point of impact. Any points outside of the point of impact are not affected by the detonation. This is used for the kinetic and armor-piercing domains, but could be used for any power that is directed at a particular target and has no area of effect. This is equivalent to direct damage as defined in the damage-by-ammo-type damage system.
- If **range-from** is unused, the determinant determines power without using the input range. This is used by the direct determinant, which ignores the **range-from** setting completely. (For more information, see "Determinant Types" below.)
- If **range-from** is shooter-and-impact-point, the determinant determines power based on both the range from the shooter and from the impact point. This option is not implemented by VR-Forces.

Determinant Types

VR-Forces supports these types of determinant.

- The **range-coefficients** determinant calculates the power based on a series of coefficients. Configuring coefficients works exactly the same way as the hit probability coefficients, except that the output value can be any number 0 or greater, not just a probability for 0.0 to 1.0. For example:

```
(kinetic
  (range-from "shooter-to-impact")
  (range-coefficients -1.3000E-003  0.0000  26005.0)
)
```

- The **range-list** determinant determines the power based on a step function. This is configured exactly the same way as the hit probability **range-list**, except that the output value can be any number 0 or greater, not just a probability for 0.0 to 1.0. For example:

```
(explosive
  (range-from "impact-point")
  (range-list
    (range 4.5
      (power 70.0)
    )
    (range 6.0
      (power 40.0)
    )
    ...
    (range 47.0
      (power 2.4)
    )
  )
)
```

- The **direct** determinant always produces the same power at the point of impact, regardless of the range from the shooter. It produces no power at any range from the point of impact. As a result, the **range-from** setting is ignored, because the range input is unused. For example:

```
(kinetic
  (direct 272.873) ;; kilojoules
)
```

- The **equivalent-tnt-mass** determinant allows you to define an amount of TNT, in kilograms, that produces an equivalent explosion. VR-Forces calculates power at a specified range in kPa. This determinant is used only for the explosive damage category. This example shows its use for M107 HE munition:

```
(detonation-power-entry
  (munition
    ;; M107 HE (155 mm)
    (munition-type 2 9 225 2 14 2 -1)
    (warhead -1)
    (guidance-mode 0)
  )
  (power-list
```

```
(explosive
  (range-from "impact-point")
  (equivalent-tnt-mass 5.74)
)
)
```

12.2.2 Configuring Detonation Power in the SOE

You configure the detonation power of each munition type on its Detonation tab. When you add or edit the detonation power for a munition, VR-Forces creates a copy of the *detonationParams.mtl* in the SMS. It is merged with the version in *base.sms* when a scenario is loaded.

► **To configure the detonation power of a munition:**

1. Select the Munitions tab of the Object List.
2. Select the munition that you want to configure.
3. Select the Detonation tab.
4. In the Power List, select the power type that you want to configure.
5. If the munition uses a range-list determinant or a coefficient-list determinant, edit the list by editing the table or by directly editing the graph, as described in "3.9. Editing Systems in the Simulation Object Editor" on page 66.

If the munition uses the direct determinant, the range is ignored. Specify the direct power in the Direct text box.

If the munition uses the equivalent TNT mass determinant, specify the explosive power in the Equivalent TNT text box.

Adding a Damage Power Type and Determinant

You can add and remove power types from the power list. When you add a power, you specify a determinant (range list, range coefficient, direct, equivalent TNT mass).

► **To add a power and determinant:**

1. Select the Munitions tab of the Object List.
2. Select the munition that you want to configure.
3. Select the Detonation tab.
4. On the Power List line, click the Add button (+). The Power and Determinant dialog box opens.
5. In the Categories window, select a power type. The options for this type of power are listed.
6. Select the determinant you want to use.
7. Specify the data for the determinant.
8. Click OK. The power is added to the list.

Changing the Determinant for a Power Type

You can change the determinant for a power type. If you change the determinant the values for the old determinant are removed and you must specify the correct data for the new determinant type.

► **To change the determinant for a detonation power type:**

1. Select the Munitions tab of the Object List.
2. Select the munition that you want to configure.
3. Select the Detonation tab.
4. Select the power you want to edit.
5. On the Power List line, click the Edit button (✎). The Choose Determinant dialog box opens.
6. Select the determinant you want to use.
7. Specify the data for the new determinant.
8. Click OK. The graph and data on the tab change.

12.2.3 The Damage File for Damage-By-Power

Entity damage is configured through the system definition files and the damage files they reference. A system definition file configures power in the **damage-by-munition-power** block.

The system definition files provided with VR-Forces have a **damage-model** block. However none of the files provided use this block. It is available for customers who want to configure munitions using the damage-by-ammo-type method. It also enables backwards compatibility with old SMSs.

Configuring Damage-by-Power in a System Definition File

A system definition file can define any number of **damage-by-power-entry** items, each for a different power-type. This is the damage domain of the power. To add a new type of power, add new settings in *detonationParams.mtl* and in the system definition files for simulation objects that need to be damaged by this effect.

Configuring the Damage File for Damage-By-Power

Damage system files configure damage based on the side of the bounding volume being affected. The damage actuator delivered with VR-Forces uses surfaces of front, rear, left-side, right-side, bottom, and top (other components could model the entity's geometry differently, and therefore use different surface strings). A surface entry has a list of angle-of-incidence entries. An angle-of-incidence entry has an angle (in radians).

The **determinant** block specifies the damage. The determinant can be a **step-function** or a list of coefficients. Each determinant can specify probabilities for catastrophic-kill, mobility-kill, firepower-kill or some combination of them.

The *ground-heavy-armor.sysdef* file has a parameter called **round-down** that indicates what probabilities should be used when the input power value is between two steps. For power

we want to round down to the next lower power (and lesser probability). Therefore **round-down** is set to 1 (true) for all damage files. If this parameter is not specified, the step function rounds up.

This code, which uses the step-function determinant, is from *./data/simulationModelSets/EntityLevel/vrfSim/systems/damage/ground-heavy-armor.sysdef*:

```
(damage-by-munition-power
...
  (damage-by-power-entry
    (key "explosive")
    (value
      (front
        (angle-of-incidence
          (angle 1.570800)
          (determinant
            (step-function (round-down 1)
              (power 7.500000
                (catastrophic-kill 0.100000)
                (firepower-kill 0.050000)
                (mobility-kill 0.050000)
              )
              (power 12.500000
                (catastrophic-kill 0.500000)
                (firepower-kill 0.250000)
                (mobility-kill 0.250000)
              )
              (power 20.000000
                (catastrophic-kill 0.600000)
                (firepower-kill 0.300000)
                (mobility-kill 0.300000)
              )
              (power 45.000000
                (catastrophic-kill 0.750000)
                (firepower-kill 0.370000)
                (mobility-kill 0.370000)
              )
              (power 70.000000
                (catastrophic-kill 0.990000)
                (firepower-kill 0.490000)
                (mobility-kill 0.490000)
              )
            )
          )
        )
      )
    )
  )
...
)
```

```
)
)
)
```

This code, which uses the **coefficients** determinant, is from

./data/simulationModelSets/EntityLevel/vrfSim/systems/damage/missile-default.sysdef:

```
(string-DtDamageProbabilityTable-map
  (damage-by-munition-power
    (damage-by-power-entry
      (key "explosive")
      (value
        (front
          (angle-of-incidence
            (angle 1.570800)
            (determinant
              (coefficients
                (catastrophic-kill 0.020000 -0.400000)
                (firepower-kill 0.022000 -0.440000)
                (mobility-kill 0.025000 -0.500000)
              )
            )
          )
        )
      )
    )
  )
  ...
)
```

12.2.4 Adding a New Power Type

To add a new type of power, you add it to the relevant munitions in *detonationParams.mtl*, add it to *./appData/settings/simulationObjectEditor/powerTypes.csv*, and then add it to the appropriate damage system. For example, suppose you want to model the effects of the heat generated by a munition. You could add thermal power. For the sake of this example, suppose that an 85 mm HEAT round has thermal power. To add thermal power, you must:

1. Add the new power to the **detonation-power-entry**:

```
(detonation-power-entry
  (munition
    ;; 85 mm HEAT
    (munition-type 2 2 222 2 8 1 -1)
    (warhead -1)
    (guidance-mode 0)
  )
  (power-list
    (explosive
      (range-from "impact-point")
      (range-list
        (range 3.5
          (power 70.0)
        )
      )
    )
  )
)
```

```
        )
        ...
    )
    )
    (kinetic
      (range-from "shooter-to-impact")
      (range-coefficients -2.6600E-004 0.0000 5319.0)
    )
    (armor-piercing
      (range-from "shooter-to-impact")
      (range-coefficients -9.0000E-006 0.0000 180.0)
    )
    (thermal
      (range-from "impact-point")
      (range-list
        (range 3.5
          (power 70.0)
        )
      )
      ...
    )
  )
)
```

You would have to determine the unit of heat that you wanted to use to measure its effect and come up with realistic power levels at different distances.

2. In the damage systems for the simulation objects that can be affected by thermal power, add a **damage-by-power** entry. Suppose that you want lifeforms to be damaged by thermal power. You would update the *lifeform-default.sysdef* file:

```
(damage-by-power-entry
  (key "thermal")
  (value
    (front-side
      ...
    )
    (left-side
      ...
    )
    (right-side
      ...
    )
    ...
  )
  ...
)
```

3. Add the power type to *./appData/settings/simulationObjectEditor/powerTypes.csv*. The fields for each entry in the file are:

- power. The power name.
- withoutaddition. The text that will be placed on the X and Y axis of graphs.
- withaddition. If there is additional information the Simulation Object Editor gives, the %1 will be replaced with that on the X and Y axis.
- withoutadditionx. If wanted, a different title for the X axis.
- withadditionx. If wanted a different replaceable title for the X axis.
- humanonly. true if this applies only to human damage.
- allowaddition. true if this can be used for addition, false if for title only.
- defaulttrangetype. The default range type (shooter-to-impact or impact-point). Please see *detonationParams.mtl* for more information.
- supportedranges. Comma list of supported range calculation values. Please see *detonationParams.mtl* for more information.

12.3. Configuring Damage in the SOE

The Simulation Object Editor provides an easy GUI for specifying damage for each of the power types that apply to an entity. You can configure damage for each surface on an entity and for angles of incidence.

 Most of the damage systems provided with VR-Forces do not configure damage for different angles of incidence.

12.3.1 Configuring Damage Power Entries

The Damage Power Entries window lists the types of damage power that VR-Forces supports. A damage system does not necessarily support all types of damage power. If a damage power type is not supported, it is labeled [Unused]. It does not have entries in the Effect Data table. You can disable use of a damage power type and enable unused damage power types.

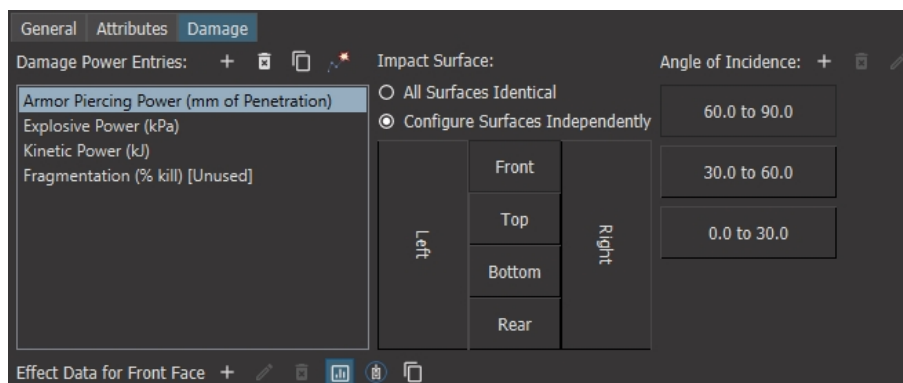
- To disable a damage power type, click the Remove button ().
- To enable an unused damage power type, add an entry to its Effect Data table.

12.3.2 Configuring Impact Surfaces

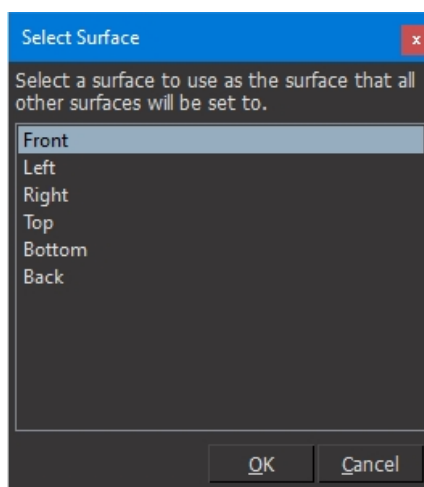
A damage system can vary the amount of damage sustained based on the surface that is struck or can use the same effect data for all surfaces.

➤ **To configure the same damage for all surfaces of an entity:**

1. In the Damage Power Entries list, select the damage power for which you want to configure impact surfaces.
2. Under the Impact Surface options (Figure 99), select All Surfaces Identical. The Select Surface dialog box opens (Figure 100).

Figure 99. Damage power entries and impact configuration

3. In the Select Surface dialog box, select the surface that will represent all surfaces and then click OK.

Figure 100. Select Surface dialog box

Text displays below the Damage Power Entries list to indicate that you are editing effect data for all surfaces.

4. Optionally, specify the angles of incidence to use, as described in "12.3.3 Configuring the Angle of Incidence" on the facing page.
5. Specify effect data for each angle of incidence, as described in "12.3.4 Specifying Effect Data" on the facing page.

If you configure damage differently for each surface, you must specify the angle of incidence and effect data for each surface.

► **To specify damage for each surface:**

1. In the Damage Power Entries list, select the damage power for which you want to configure impact surfaces.
2. In the Impact Surface options list, select Configure Surfaces Independently. Text displays below the Damage Power Entries list to indicate which surface you are editing (Figure 99).

3. For each surface that you want to configure:
 - a. Click the box that represents the surface you want to configure.
 - b. Optionally, specify the angles of incidence to use for this surface, as described in "12.3.3 Configuring the Angle of Incidence" below.
 - c. Specify effect data for each angle of incidence, as described in "12.3.4 Specifying Effect Data" below.

12.3.3 Configuring the Angle of Incidence

You can specify different damage results based on the angle at which a munition strikes an entity. If a damage system treats all surfaces identically, you configure one set of angles of incidence. If it treats different surfaces differently, you can configure angles of incidence for each surface. The default for most damage systems is that there is no difference based on the angle. You can specify multiple angles as ranges. The total range of all angles must be 0 degrees to 90 degrees.

► **To add an angle of incidence:**

1. In the Damage Power Entries list, select the damage power for which you want to configure angles of incidence.
2. If you are configuring each impact surface independently, click the surface for which you want to configure the angle of incidence.
3. Click the Add button (+) next to the Angle of Incidence label (Figure 99 on the previous page). The New Angle dialog box opens.
4. Enter an angle, in degrees.
5. Click OK. The new angle is added to the list.

Editing an Angle of Incidence

► **To edit an angle of incidence:**

1. In the Damage Power Entries list, select the damage power for which you want to configure angles of incidence.
2. If you are configuring each impact surface independently, click the surface for which you want to configure the angle of incidence.
3. Click the angle that you want to edit.
4. Click the Edit button (✎) next to the Angle of Incidence label. The Edit Angle dialog box opens.
5. Enter the new angle.
6. Click OK.

12.3.4 Specifying Effect Data

For each of the damage power types used by a damage system, you can specify effect data. Effect data can be specified using a range list or a coefficient list. You can change the type of list that is used for a damage system. You can specify effect data for catastrophic kill, firepower kill, and mobility kill. (Many of the systems provided with VR-Forces only configure catastrophic kill.)

If you are using a range list, edit the ranges and range values as described in Figure 30 on page 69.

If you are using a coefficient list, edit the probability table as described in "Editing a Coefficient List Probability Table" on page 70.

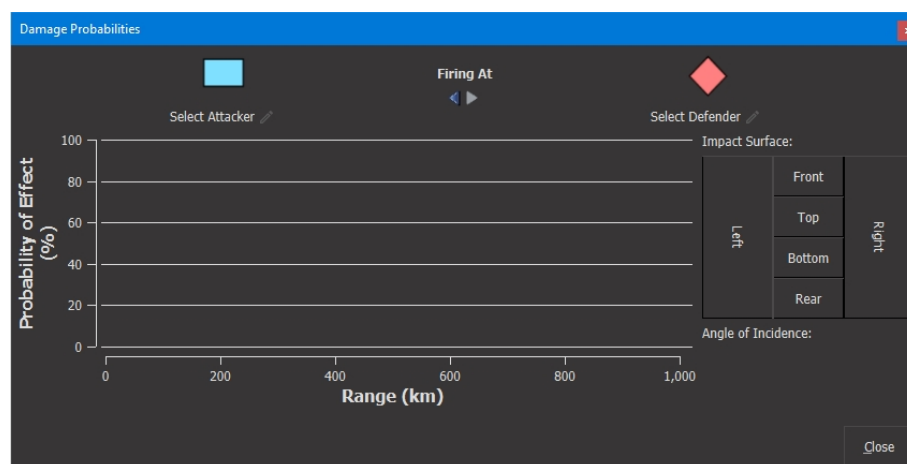
12.4. Viewing Inter-Object Damage Probability Results

Sometimes when VR-Forces users create a new weapon system or damage system, the systems do not work as expected. The system might not fire on a target or might not damage it when it does. This is usually due to a mismatch in systems or a misunderstanding of how damage probabilities are calculated. The Damage Probabilities viewer shows you what you can expect when a specific entity fires on another specific entity.

► **To view damage probability results:**

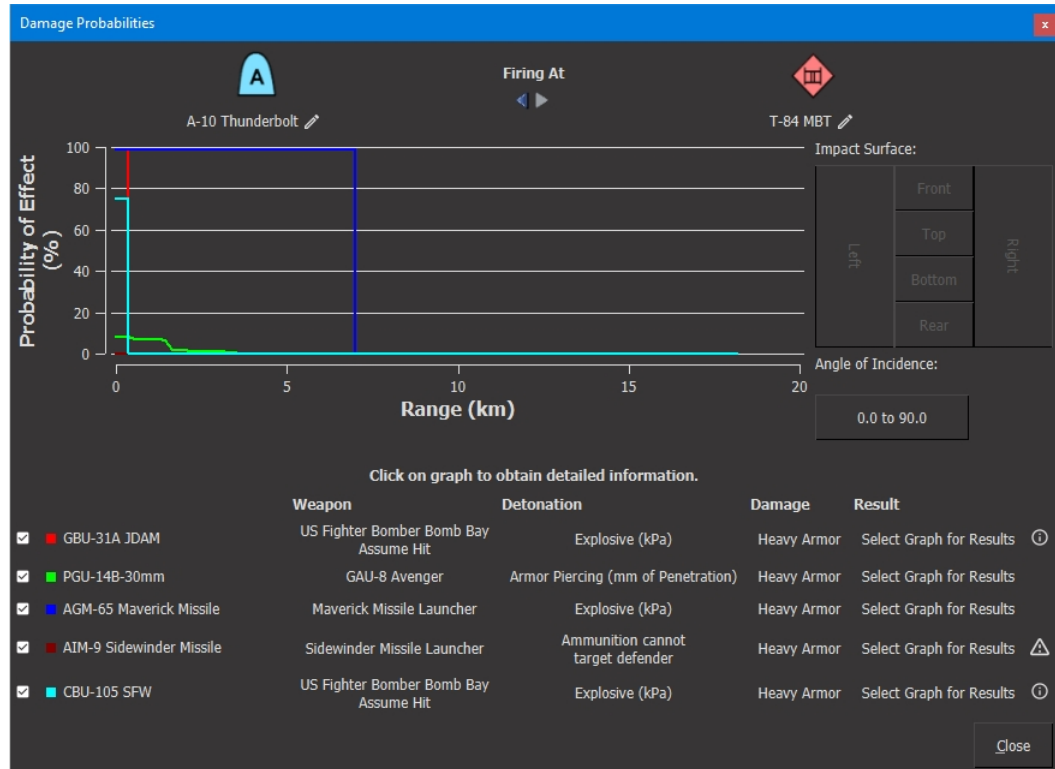
1. Choose **View > Damage Probability Results**. The Damage Probabilities viewer opens (Figure 101).

Figure 101. Damage Probabilities viewer



2. Click the Select Attacker button. The Select Simulation Object dialog box opens. It is an object list similar to the object list in the SOE or the Simulation Objects Panel.
3. Select the entity that you want to test as the attacker.
4. Click OK. The entity name is displayed under the button.
5. Click the Select Defender button. The Select Simulation Object dialog box opens.
6. Select the entity that you want the attacker to fire on.
7. Click OK. The entity name is displayed under the button. The viewer redisplay to show the probability graph for each weapon that the attacker has (Figure 102).

Figure 102. Damage Probabilities viewer with entities selected



8. Click in the graph at the range for which you want to see the probability of effect result. The Information table title (Information at range x.x) shows the range you have selected. The Result column in the Information table shows the probability that each weapon system on the attacker has of damaging the defender.
9. If the defender has multiple impact surfaces configured, select the surface whose probability you want to view.
10. If the defender has multiple angles of incidence configured, select the angle of incidence whose probability you want to view.
11. Optionally, clear the check boxes next to individual weapon systems to filter the number of graphs in the window.
12. Optionally, to reverse the relationship between attacker and defender, click the Firing At arrow buttons.
13. Optionally, to see the configuration page for the attacker or defender, click the Edit button (✎) next to the entity name.
14. Optionally to see the configuration page for the weapon, munition, or damage system, click the label in the appropriate column.
15. Optionally, click an Alert button (⚠) or Information button (ⓘ) to display explanatory information, such as the fact that a weapon system cannot target the selected defender.

12.5. Configuring Damage for Dynamic Terrain

Dynamic terrain damage is configured using the damage-by-power approach. It is configured in `./data/simulationModelSets/base/vrfSim/systems/other/dynamic-terrain.sysdef` in the settings for the dynamic-terrain-munition-damage controller. Terrain can be damaged using standard terrain damage or high fidelity terrain damage.

i Dynamic terrain is supported on OpenFlight and FBX terrain patches, and Heightfields and Point Model Replace in osgEarth. MetaFlight, Terrapage, and CDB terrains, and also props do not support dynamic terrain.

12.5.1 Standard Terrain Damage

When you use standard terrain damage, the controller publishes a series of concentric damage areas of decreasing damage from the impact point. The type of terrain damage (that is, the switch in the terrain to be updated) is defined by **standard-terrain-damage-type** in `dynamic-terrain.sysdef`. The detonation power domain that causes this damage is defined by **standard-power-type**. The `standard-terrain-damage-table` block specifies the concentric areas. When a detonation is received, for each entry in this table a new area is created based on the range that produces the specified power. Then the switch in the terrain model is set to the specified damage setting.

This code is from `./data/simulationModelSets/base/vrfSim/systems/other/dynamic-terrain.sysdef`:

```
(standard-terrain-damage-table
  (damage-by-power-entry
    (power 15) ;; explosive power is in kPa
    (damage "slight")
  )
  (damage-by-power-entry
    (power 40)
    (damage "moderate")
  )
  (damage-by-power-entry
    (power 70)
    (damage "destroyed")
  )
)
```

This table generates three dynamic terrain areas. The smallest area is defined by the area on the terrain that experiences an explosive power of at least 75. It is switched to a value of "destroyed". The next larger area is the area that experiences power between 40 and 75. It is switched to a value of "moderate". Finally the outer area is the area that experiences power between 15 and 40. It is switched to a value of "slight".

12.5.2 High Fidelity Terrain Damage

You can configure high fidelity damage areas along with the standard damage. To determine where high fidelity damage areas should be created, VR-Forces queries the terrain to find any dynamic terrain features that match the given **dynamic-terrain-type**

setting. This can be used to generate additional or more specific types of damage to select locations in the terrain that support a higher fidelity of damage. Just as with standard damage, a **power-type** is specified that corresponds to the detonation power domain that is causing this damage. High fidelity terrain damage is configured in the `high-fidelity-terrain-damage-table` block. It has the same format as the `standard-terrain-damage-table`.

The high fidelity damage lets you specify the damage-algorithm used. VR-Forces only supports the range-based algorithm, but you can develop your own algorithms in a plugin. If you add additional damage algorithms, they might require configuration options that are not supported in the `high-fidelity-damage-table` provided by MAK. If necessary, you can specify additional parameters using the **additional-algorithm-config** parameter, which accepts a list of key/value string pairs.

High fidelity damage can produce a secondary detonation. This can be specified by setting **detonation-on-destroy** to True and defining the **detonation-munition** parameter. The **delay** parameter defines a delay in seconds between when power was first experienced and when the damage should be done. This allows the secondary detonation to create a particle effect that can mask the underlying terrain switch.

This code is from `./data/simulationModelSets/base/vrfSim/systems/other/dynamic-terrain.sysdef`:

```
(high-fidelity-terrain-damage-table
  (entry
    (dynamic-terrain-type "damage_structure")
    (power-type "explosive")
    (damage-algorithm "range-based")
    (delay 2.0)
    (detonate-on-destroy True)
    (detonation-munition 2 0 0 4 2 0 0)
    (terrain-damage-table
      (damage-by-power-entry
        (power 15) ;; explosive power is in kPa
        (damage "slight")
      )
      (damage-by-power-entry
        (power 40)
        (damage "moderate")
      )
      (damage-by-power-entry
        (power 70)
        (damage "destroyed")
      )
    )
    (additional-algorithm-config
      ;; Additional key/value pairs (string values) can be added here
      ;; for custom algorithm configuration
      ;; For example:
      ;; (my-custom-algorithm-config "config value string")
    )
  )
)
```

12.6. Configuring Detonation Results

The SISO Enumerations Document specifies a list of detonation results, such as Entity Impact, Ground Impact, Detonation, and Air Burst. When a munition detonates, a detonation-result message is sent on the network. Applications can use this information to determine how to visualize the detonation or for other purposes.

By default, VR-Forces uses a small subset of the detonation results specified in the SISO Enumerations Document. You can customize the detonation result messages sent by VR-Forces by editing the detonation-results-table in `./data/simulationModelSets/base/detonationParams.mtl`.

For each munition type, you can configure the detonation results for entity impacts, surface impacts, and mid-air detonations. You can specify the detonation results using the strings listed in the comments at the top of the detonation-results-table or you can use any of the integer values specified in the SISO Enumerations Document for the different detonation result types. The *DtDetonationManager* converts the strings or integers into the appropriate detonation result messages to send over the network. (For details about the *DtDetonationManager*, please see the *VR-Forces Developers Guide* class documentation. If editing *detonationParams.mtl* does not meet your needs, you can use the API to override *DtDetonationManager*.)

This code shows the entry in the detonation-results-table for an anti-ship missile:

```
(detonation-result-entry
  (munition
    ;; Anti-Ship Missile (incl. Exocet)
    (munition-type 2 6 -1 1 -1 -1 -1)
    (warhead -1)
    (guidance-mode 0)
  )
  (entity-impact-result-list
    (default-entity
      (entity-type -1 -1 -1 -1 -1 -1 -1)
      (detonation-result "entity-impact")
    )
  )
  (surface-impact-result-list
    (default
      (surface-type "default")
      (detonation-result "ground-impact")
    )
    (water
      (surface-type "water")
      (detonation-result "detonation")
    )
    (undefined
      (surface-type "undefined")
      (detonation-result "detonation")
    )
  )
  (ground
```

```
        (surface-type "ground")
        (detonation-result "ground-impact")
    )
)
(mid-air-detonation-result "detonation")
)
```




13. Configuring Movement Systems

This chapter describes configuration files that affect simulation object movement.

13.1. Configuring Formations	260
13.1.1 Formation Files	260
13.1.2 Creating a User-Defined Formation	261
13.2. Configuring Obstruction Avoidance	262
13.2.1 Configuring Object Types	262
13.2.2 Configuring Vector Feature Types	262
13.3. Configuring Embarkation	263

13.1. Configuring Formations

i Although you can create and edit formation files by hand as described in this section, the recommended approach for create and edit formations is to use the Simulation Object Editor.

The object parameter database lists the formations available to a unit in the `formations` element in its `.entity` file. Each formation is identified by its name and the path to the formation file, for example:

```
<formations paramName="formation-list" autoLayout="0">
  <formation formationName="Line" formationFile=
    "@(formation-dir)/platoonLine.frm"/>
  <formation formationName="Column" formationFile=
    "@(formation-dir)/platoonColumn.frm"/>
  <formation formationName="Wedge" formationFile=
    "@(formation-dir)/platoonWedge.frm"/>
  <formation formationName="Vee" formationFile=
    "@(formation-dir)/platoonVee.frm"/>
</formations>
```

13.1.1 Formation Files

A formation file specifies the name of the formation and has an entry for each simulation object in the default formation definition. Each entry specifies a **promotion-id**, a **leader-promotion-id**, and a **position-offset**, where:

- The **promotion-id** specifies the order in which simulation objects are promoted to unit leader if the leader is incapacitated.
- The **leader-promotion-id** specifies the unit member from whom the entry's position will be offset.
- The **position-offset** specifies how many meters behind, to the right, and above the entry will follow its leader. The **position-offset** is specified in terms of the body coordinate system of its leader.

The following is a portion of the formation file for a column formation:

```
(column-formation
  (entry-0
    (promotion-id 0)
    (leader-promotion-id -1)
    (position-offset 0.000000 0.000000 0.000000)
  )
  (entry-1
    (promotion-id 1)
    (leader-promotion-id 0)
    (position-offset -25.000000 0.000000 0.000000)
  )
  (entry-2
    (promotion-id 2)
```

```

    (leader-promotion-id 1)
    (position-offset -25.000000 0.000000 0.000000)
  )
  (entry-3
    (promotion-id 3)
    (leader-promotion-id 2)
    (position-offset -25.000000 0.000000 0.000000)
  )
)

```

In a column, each simulation object follows the simulation object in front of it by 25 meters. Therefore, the **leader-promotion-id** for each entry is the **promotion-id** of the simulation object in front of it, and each entry has a **position-offset** of -25 meters.

The number of entries in a formation file is finite. However, it is possible to place more simulation objects into a unit than there are entries in a formation file. If this happens, VR-Forces increments the **promotion-id** of the last entry in the formation file and assigns the new simulation object the same **position-offset** as the last entry in the formation file. For a simple formation like a column, this works well. If you are using a more complex formation, this mechanism might create unintended relationships among simulation objects.

If you want to change the layout of the formations provided with VR-Forces, you can edit the formation files.

13.1.2 Creating a User-Defined Formation

You can create formations other than those provided by VR-Forces. It is recommended that you create formations in the Simulation Object Editor. If you use the Simulation Object Editor, you do not have to edit any formation files or *.entity* files by hand.

► To create a formation:

1. Using one of the formation files supplied with VR-Forces as your template, place as many entries as you want in the formation file and specify their offsets.
2. Save the new formation under a new name.
3. Add an entry to the formation list block in the *.entity* file for each unit that you want to be able to use the formation. Be sure to use the same structure and naming conventions as the default entries. For example:

```

<formations paramName="formation-list" autoLayout="0">
  <formation formationName="Line" formationFile="@(\
    formation-dir)/DI-companyLine.frm"/>
  <formation formationName="Column" formationFile="@(\
    formation-dir)/DI-companyColumn.frm.frm"/>
  <formation formationName="wedge" formationFile="@(\
    formation-dir)/DI-companywedge.frm"/>
  <formation formationName="Vee" formationFile="@(\
    formation-dir)/DI-companyVee.frm"/>
</formations>

```

We recommend that you configure your formations so that simulation objects follow the leader, rather than the next closest member of the formation (as is done in column formation.) This ensures that formations have the best chance of maintaining their formation as they maneuver.

13.2. Configuring Obstruction Avoidance

You can configure ground entities to avoid specific entity types, vector features, or both.

i The more vector feature types and entity types specified for avoidance, the more memory and processing power required.

Obstruction avoidance is influenced by the **stopping-distance-factor** and **lateral-clearance** parameters. If entities display erratic behavior, try raising the values, lowering the speed of the entities, or both.

13.2.1 Configuring Object Types

To specify the types of objects that an entity type should avoid, add or remove the appropriate object type from the obstruction sensor entry in the movement system definition file. For example, these object types are configured for ground entities:

```
(object-type
  (object-type 1 (18 -1 -1 2 -1 -1 -1)) ;; vrfobstacles object type
  (object-type 1 (1 -1 -1 -1 -1 -1 -1)) ;; DtPlatform entity kind
  (object-type 1 (3 -1 -1 -1 -1 -1 -1)) ;; DtLifeForm entity kind
)
```

13.2.2 Configuring Vector Feature Types

Configuring avoidance of vector features relies on the attributes on the feature data as well as the named queries defined in `./appData/settings/featureconfig.txt`. To configure specific feature data for obstruction avoidance, modify the feature-types-to-avoid type list in the obstruction sensor in the appropriate system definition file. The types in this file are named feature queries, and you can add as many as you need.

Example of *human.sysdef*:

```
(obstruction-sensor
  ...
  (feature-types-to-avoid
    (type "MAK_OBSTACLE")
  )
)
```

Example of *air-cushion-default.sysdef*. This entity specifically calls out building, infrastructure, and vegetation, instead of obstacle, because obstacle also includes waterway.

```
(obstruction-sensor
  ...
  (feature-types-to-avoid
```

```
        (type "MAK_BUILDING")
        (type "MAK_INFRASTRUCTURE")
        (type "MAK_VEGETATION")
    )
)
```

13.3. Configuring Embarkation

Entities and units that are configured to allow entities to embark on them have an occupancy-director-controller component. Its parameters configure embarkation:

- The locations at which entities can embark on an entity are configured as **ingress-points**.
- The locations from which entities can disembark are configured as **egress-points**.
- The number of entities that can embark on an entity is defined by the **embarkation-slots** parameter.

The **embarkation-slots** parameter has one or more **embarkation-slot-entry** parameters that identify a location on the entity where an embarked entity can be placed. The **embarkation-slot-entry** parameters also specify the orientation of the embarked entity and the types of entities that can occupy the slot.

The **ingress-points** and **egress-points** parameters each have one or more **load-entry** parameters. The **load-entry** parameters specify one or more **load-entry-point** parameters. If there is one **load-entry-point**, an embarked entity enters the parent entity at that point and is immediately placed in an available slot. If a **load-entry** has multiple load entry points, they specify the points in a route that the embarking entity follows to reach its slot. This configuration is for large parent entities, such as aircraft carriers, on which entities land and then taxi to a slot.

You configure embarkation in the Simulation Object Editor. For details, see "Chapter 7. Configuring Embarkation" on page 139.



14. Configuring Emitters and Radios

This chapter describes configuration files for emitters and radios. For information about configuring the display of emitters, see "Configuring Emitter Volumes".

14.1. Configuring Emitters	266
14.2. Configuring Radios	268
14.2.1 Publishing Radio Transmitters	268

14.1. Configuring Emitters

All fixed-wing aircraft have emitters as part of their radar sensor. Emitters have search modes. Search modes have a list of beams. You can configure entities with emitters that have more than one beam. You can also configure multiple emitters. The basic radar sensor with an emitter and two search modes (*./data/simulationModelSets/<model_set>/vrfSim/systems/sensors/active-radar-sensor.sysdef*) is as follows:

```
(active-radar-sensor-system
  (systems )
  (sensors
    (radar-sensor
      (component-descriptor-type "radar-signature-sensor-
        descriptor")
      (component-type "radar-signature-sensor")
      . . .
      (sensor-domain "radar")
      (sensor-offset $sensor-position)
      (sensor-positional-error 0.000000)
      (detection-level-determinator
        (determinator-type "signature-detection-level-
          determinator")
        (detection-level-to-set-hostility 3)
        (combat-identification-level-table-file
          (filename "$(detection-dir)\std-radar-detection-
            table.csv")
        )
      )
    )
    (detection-period 0.000000)
    (publish-emitter-system True)
    (emitter-system
      (system-name-enum 5570)
      (system-function-enum 5)
      (radar-mode-list
        (search-mode
          (index 0)
          (radar-mode "Search")
          (beam-list
            (beam-0
              (index 0)
              (type "track")
              (frequency 4999999488.000000)
              (frequency-range 10000.000000)
              (power 70.000000)
              (pulse-repetition-frequency 0.000000)
              (pulse-width 0.000000)
              (azimuth-center 0.000000)
              (azimuth-sweep 0.523599)
              (elevation-center 0.349066)
              (elevation-sweep 0.523599)
```



```
(radar-mode "Search")
(beam-list
  (beam-0
    ...
  )
  (beam-1
    ...
  )
)
```

14.2. Configuring Radios

All radios are *DtVrfRadio* classes or subclasses. The radio is responsible for managing incoming and outgoing radio messages on a single simulation object. Each radio is associated with a specific communication model, and uses this communication model to send and receive messages.

Radios are configured in platform (OPE) files. The radios block in the OPE file lists all the radios on the simulation object and the parameters for each radio. Parameters include the type of radio descriptor and type of radio class, which are strings used to create the C++ classes from the factories. The **comm-model-name** parameter identifies which communication model a radio is associated with. Every radio must be associated with a communication model.

A typical radio entry is:

```
(main-radio
  (radio-descriptor-type "radio-descriptor")
  (radio-type "default-radio")
  (comm-model-name "default-radio-model")
  (publish-transmitter True)
  (transmitter-params
    (radio-type 7 1 225 2 1 20)
    (initial-frequency 3000000.000000)
    (power 30.000000)
  )
)
```

14.2.1 Publishing Radio Transmitters

The external communication model requires that any simulation objects sending or receiving messages through the communications effects server publish a DIS or HLA radio transmitter object. By default, the internal communication model does not publish radio transmitters.

Publishing radio transmitters is controlled by two parameters:

- The **publish-transmitters** parameter in *./data/simulationModelSets/<model_set>/commModelParams.mtl*. By default, this parameter is False for the simple communication model and True for the external communication model.

- The **publish-transmitter** parameter in an entity's OPE file. This parameter is set to true for all object types.

To publish radio transmitters, both of these parameters must be True. If you do not change the OPE files and set **publish-transmitters** to True, all simulation objects will publish radio transmitters. If you set **publish-transmitters** to True and do not want all simulation objects to publish radio transmitters, you can set **publish-transmitter** to False for the object types that you do not want to publish them.



15. Joystick and Keyboard Controls

This chapter explains how to configure joysticks and the keyboard to control entity movement.

15.1. Using a Joystick to Control Entities	272
15.1.1 Switching Between Systems	273
15.1.2 Disabling Joystick Control	274
15.2. Using the Keyboard to Control an Entity	274
15.3. Configuring Joysticks and Keyboard Control	274
15.3.1 Creating a Joystick Configuration	275
15.3.2 Mapping the Joystick Stick	277
15.3.3 Mapping Joystick Buttons and Keyboard Keys	278
15.3.4 Configuring “Switch Between” Options	280
15.3.5 Using Multiple Joysticks	283
15.3.6 Specifying a Favorite Joystick Configuration	284
15.4. Installing a Gamepad on Linux	284

15.1. Using a Joystick to Control Entities

VR-Forces supports use of joysticks, other types of game controllers, and keyboard keys to control ground, fixed-wing, and rotary-wing entities. You can attach multiple joysticks, which allows you to control multiple entities, movement and weapons fire on one entity, or some combination. You can also use a joystick and keyboard mappings together to control an entity.

Joystick control is managed in the GUI. Joysticks should be attached to the computer from which you are running the GUI. Control messages are sent to the sim. engine, which can be on any computer in the simulation.

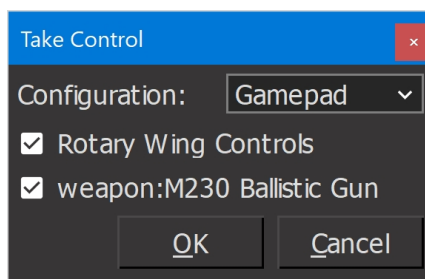


- Red Hat 7 does not support joysticks by default, but it can be enabled in the operating system. For details, go to this website: <https://access.redhat.com/solutions/64114>
- You cannot enable or disable joystick support for all entities at once. You can only enable joysticks for individual entities.
- Embarked entities controlled by joystick cannot disembark. For example, if you use a joystick to control a fixed-wing taking off from a carrier, the aircraft will take off, but it will remain in an embarked state.
- For information about how to configure joysticks, see "15.3. Configuring Joysticks and Keyboard Control" on page 274.

► To enable joystick control:

1. Select the entity for which you want to enable joysticks.
2. Choose **Objects > Joystick > Take Control**. The Take Control dialog box opens.

Figure 103. Take Control dialog box



3. Select a joystick configuration from the Configuration list. (If you have specified a joystick configuration as a favorite, you do not have a choice of the configuration to use.)
4. Select the entity functions you want to control, usually movement and one or more weapons. (You can have more than one joystick attached and can also use keyboard controls, if they are configured.)
5. Click OK.

15.1.1 Switching Between Systems

Systems that are controlled by joysticks can be organized into functional groups so that you can configure them similarly and switch between the systems using a "switch between" key or button. VR-Forces includes a switch group for weapons. If an entity has multiple weapon systems, you can manage them:

- Configure all weapons to use the same keys or joystick controls for the same function. For example, to fire any weapon, you could press **F**. Use the switch between key to switch between weapons for a given entity. You only need to learn one set of controls to manage all your weapons.
- Configure all weapons to use different keys or joystick controls for the same functions. For example, to fire a 120 mm gun, press **F**; to fire a 7.62 mm gun, press **G**. You would not use the switch between option.

If an entity has multiple weapons and you use switch options to change which one you control, there is no visual cue in the GUI to show you which weapon you are currently controlling. If there are more than two weapons on an entity, the switch option cycles through them.

You can create your own switch groups. For details, see "15.3.4 Configuring "Switch Between" Options" on page 280.

Configuring Multiple Identical Systems

If an entity has more than one instance of the same system, you must change the name of the function group if you want to control them separately. For example, suppose you want to mount two chain guns on an Apache helicopter. If they both have the same function group name (`weapon:M230 Ballistic Gun`), then you could not switch between them. You would need to rename one of them. This is done in the Simulation Object Editor when you add the additional system. (For complete details about using the Simulation Object Editor, see "Chapter 2. The Simulation Object Editor and SMSs" on page 3 through "Chapter 7. Configuring Embarkation" on page 139.)

► To rename a system function group:

1. On the Start menu, choose **MAK Technologies > VR-Forces 5.2 Simulation Object Editor**. The Simulation Object Editor opens.
2. Load *EntityLevel.sms* or your own SMS.
3. Select the entity whose system you want to edit.
4. Select the Systems tab.
5. In the systems list—for example, the Weapons list—select the system you want to edit.
6. Click the Edit Properties button () for the system. A Properties dialog box opens.
7. Edit the function group name. For example, change `weapon:M230 Ballistic Gun` to `weapon:M230 Ballistic Gun 1`.
8. Click OK.
9. Save the SMS.

15.1.2 Disabling Joystick Control

► To disable joystick control:

1. Select the entity for which you want to disable joysticks.
2. Choose **Objects > Joystick > Release Control**.

15.2. Using the Keyboard to Control an Entity

The joystick configuration process allows you to configure keyboard control of entity movement and weapons. You can use keyboard control along with joysticks. For example, you could control entity movement from the keyboard and weapons from a joystick, or the opposite.

To enable or disable keyboard control, follow the procedures for enabling and disabling joystick control. Similarly, to configure keyboard control, follow the joystick configuration process. For more information, see "15.1. Using a Joystick to Control Entities" on page 272 and "15.3. Configuring Joysticks and Keyboard Control" below.

15.3. Configuring Joysticks and Keyboard Control

You can control entity movement and weapon fire using a joystick, the keyboard, or both. You can attach multiple joysticks and can use a mix of joystick and keyboard control at the same time. You configure joystick and keyboard mappings on the Application Settings dialog box, Joystick Configuration page.

The controls available for configuration on the Joystick Configuration page are taken from the joystick controllers in the movement, gun, and other system definitions. For example, the *ground-tracked.sysdef* file has the following entry:

```
(joystick
  (component-descriptor-type "joy-controller-descriptor")
  (component-type "ground-move-joystick-controller")
  ...
  (joystick-controls
    (steering
      (function-name "steering")
      (function-group $automotive-driving)
      (description "Controls the steering of this entity")
      (min-value -1.0)
      (max-value 1.0)
    )
    (throttle
      (function-name "throttle")
      (function-group $automotive-driving)
      (description "Controls the throttle of this entity")
      (min-value 0.0)
      (max-value 1.0)
    )
    (brake
      (function-name "brake")
```

```

    (function-group $automotive-driving)
    (description "Controls the braking of this entity")
    (min-value 1.0)
    (max-value 1.0)
  )
  (change-gear
    (function-name "change-gear")
    (function-group $automotive-driving)
    (description "Puts the entity in forward, reverse or
      pivot gear")
    (min-value 0)
    (max-value 2)
    (cycle True) ;; Each click cycles from min to max
  )
)
)

```

Based on this system definition file, the Joystick Configuration page (Figure 104 on the next page) has an Automotive Driving function with controls for brake, change-gear, steering, and throttle.

The **min-value** and **max-value** parameters determine how the edit buttons on the Joystick Configuration page work:

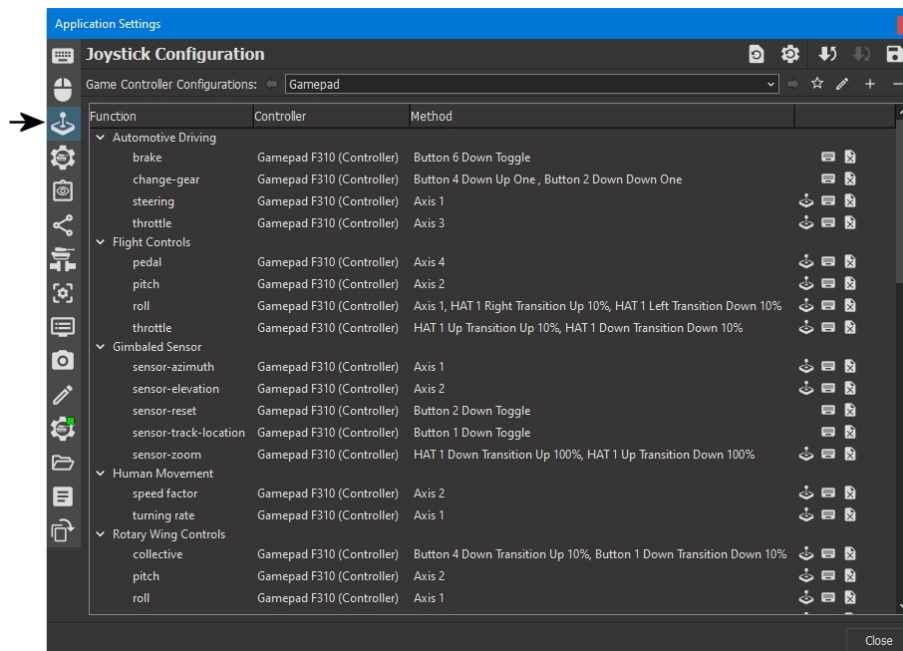
- If **min-value** and **max-value** are both set to 1 (as in the brake control in the example system definition file), the control is a key mapping button (). You can set it to Yes or No.
- If **max-value** is an integer greater than 1 (as in the change gear control in the example system definition file), the control is a key mapping button. You can configure keyboard buttons to increase or decrease the control by 1, or to set it to a specific value.
- If **min-value** does not equal **max-value** and **max-value** is less than or equal to 1 (as in the throttle control in the example system definition file), there is a joystick button () and a key mapping button on the Joystick Configuration page. The keyboard button lets you specify a percentage increase or decrease of the control value. (For example, increase throttle by 10%.)

15.3.1 Creating a Joystick Configuration

A joystick configuration contains the joystick mappings and, optionally, keyboard mappings that you can use to control an entity. VR-Forces includes default configurations for a game controller and a joystick.

► **To create a joystick configuration:**

1. Create or open a scenario.
2. Choose **Settings > Application**. The Application Settings dialog box opens.
3. Select the Joystick Configuration page (Figure 104).

Figure 104. Joystick Configuration page

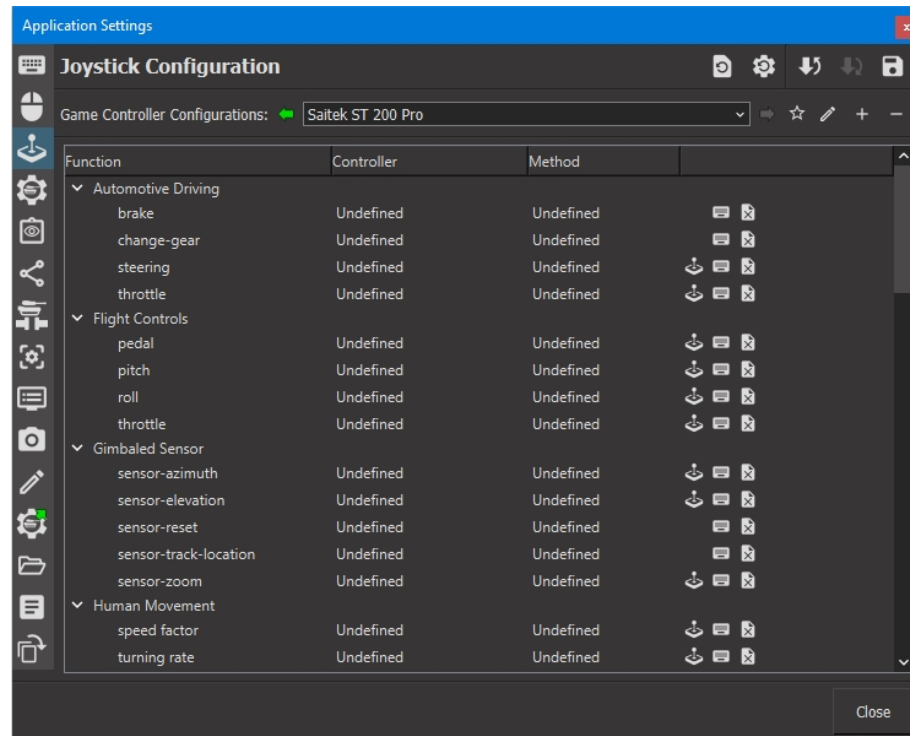
4. Click the Add button (+). The New Configuration dialog box opens.
5. Enter a name for the configuration.



The name of the joystick configuration must match the name that the joystick uses to identify itself to the computer.

6. Click OK. The window redisplay to show the functions that this device can control (Figure 105).

Figure 105. Joystick configuration



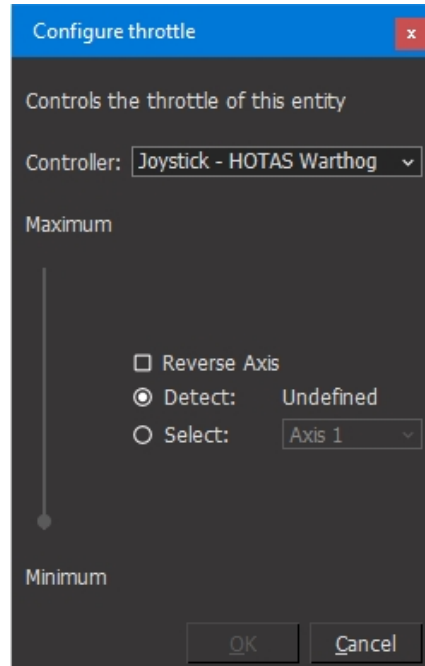
7. Edit the joystick and keyboard mappings as described in "15.3.2 Mapping the Joystick Stick" below and "15.3.3 Mapping Joystick Buttons and Keyboard Keys" on the next page.

15.3.2 Mapping the Joystick Stick

To control an entity with a joystick, you map the joystick controls to specific entity movement components and weapons.

► **To map joystick controls:**

1. Create a joystick configuration, as described in "15.3.1 Creating a Joystick Configuration" on page 275 .
2. On the Joystick Configuration page (Figure 104 on the previous page), click the Joystick button (🕹️) for the function you want to map. A joystick configuration dialog box opens (Figure 106).

Figure 106. Joystick configuration dialog box

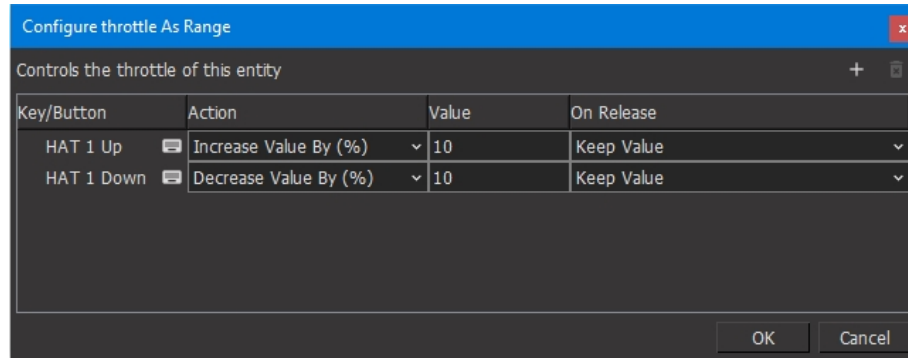
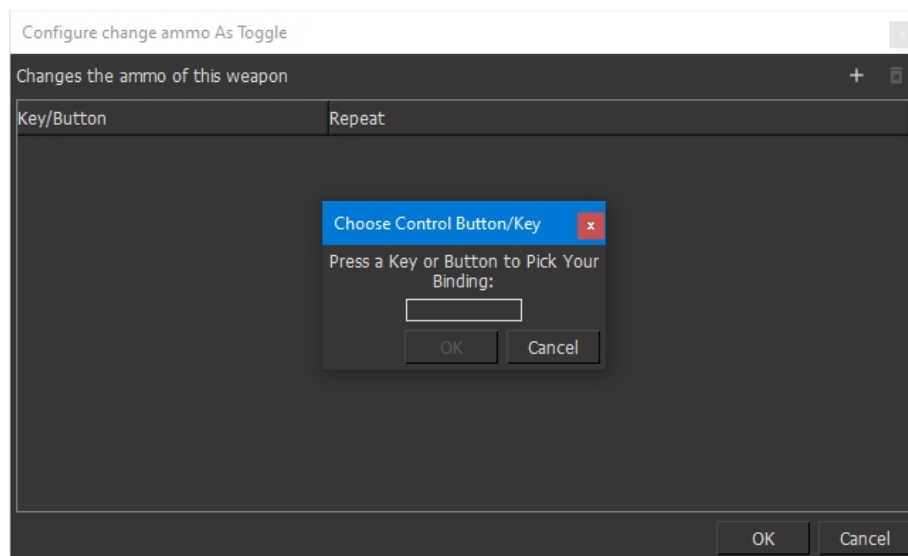
3. In the Controller list, select the controller you want to configure.
4. Configure each axis:
 - a. To let VR-Forces select the axis, select the Detect option and move the joystick in the axis you want to use for this control.
 - b. To choose the axis, select the Select option and select an Axis from the list.
 - c. Move the joystick in the axis you are configuring. If the Maximum/Minimum slider moves in the direction that you expect, you are finished. If the slider moves opposite to what you expect, select the Reverse Axis check box. It should now move as expected.
5. Click OK.
6. Repeat this process for each function you want to control with the joystick.

15.3.3 Mapping Joystick Buttons and Keyboard Keys

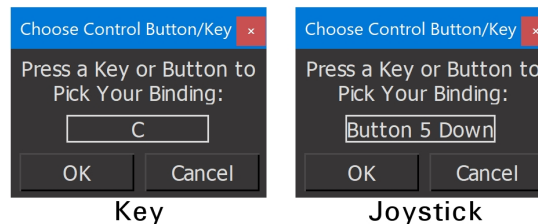
You can map joystick buttons and keyboard keys to entity movement control and weapon control. When entity control is enabled, the entity control key mappings override any other key mappings, such as observer control mappings.

► **To map joystick buttons and keyboard keys to entity control:**

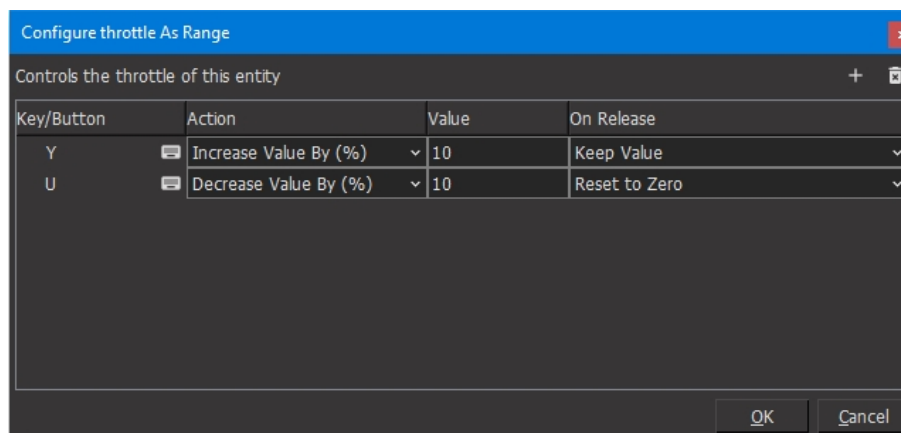
1. Create a joystick configuration, as described in "15.3.1 Creating a Joystick Configuration" on page 275.
2. On the Joystick Configuration page (Figure 104 on page 276), click the Key Mapping button (🗑️) for the control you want to map. A configuration dialog box opens. It may call for a range of values (Figure 107), or just be a toggle (Figure 108).

Figure 107. Key mapping configuration dialog box**Figure 108. Key mapping configuration as toggle dialog box**

3. If the mapping requires a range, for each row in the dialog box:
 - a. Click the Key Mapping button (🖱️). The Choose Control Button/Key dialog box opens (Figure 109).
 - b. Press the joystick button or keyboard key that you want to use to activate this control. It displays in the dialog box (Figure 109). (If a key binding is not available, the OK button is not enabled.)

Figure 109. Choose Control Button/Key dialog box with selection

- c. Click OK.
- d. In the Action column, select an option from the list.
- e. In the Value column, enter an appropriate value for the Action option.
- f. In the On Release column, select an option from the list. (This is what happens when you release the key.) A complete dialog box looks similar to Figure 110.

Figure 110. Key mapping configuration dialog box

4. If the keyboard mapping is a toggle, the Choose Control Button/Key dialog box opens automatically. Do the following:
 - a. Press the key or joystick button that you want to use.
 - b. Click OK.
 - c. In the Repeat column, select Yes or No. Yes means that the action repeats if you hold down the key. No means it happens one time for each key press.
5. Click OK.
6. Repeat this process for each function you want to configure.

15.3.4 Configuring “Switch Between” Options

Some entities may have multiple systems of a certain type, such as weapons. You might want to be able to control all similar systems with the same set of keys and joystick controls. VR-Forces lets you do this with switch groups. You can specify a key or button to press to switch between the different systems in a switch group. For example, you might press a button to switch between a tank's main battle gun and a machine gun.

The controls for which you can specify a switch between option are listed in the Switch Between section of the Joystick Configuration page. Specifying the key or button to use is described in "15.3.3 Mapping Joystick Buttons and Keyboard Keys" on page 278. This section explains how to set up switch options.

To set up a switch option, when you define a function group in a system definition file's metadata, you use the syntax `switch_option:system`. The `switch_option` is a keyword. If an entity has multiple systems that use that keyword, you can switch between

them using the switch between key or button. VR-Forces systems use the **weapon** keyword for switching between weapon systems. The following code, from *mm-gun.template_sysdef* shows how this works.

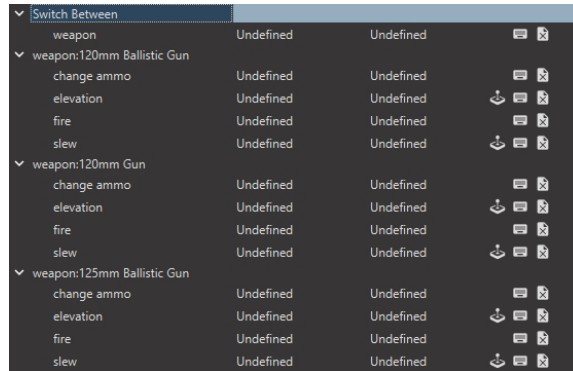
The controllers specify two function groups for joystick control: `$slew-group` and `$ballistic-gun-group`. These groups specify four functions that the joystick can control: slew, elevation, fire, and change ammo:

```
(weapon-120mm-gun
  (controllers
    (turret-joystick
      ...
      (joystick-controls
        (slew
          (function-name "slew")
          (function-group $slew-group)
          (description "Controls the slew of this weapon")
          (min-value -1.0)
          (max-value 1.0)
        )
      )
    )
    ...
    (main-gun-control-joy
      (joystick-controls
        (elevation
          (function-name "elevation")
          (function-group $ballistic-gun-group)
          (description "Controls the elevation of this weapon")
          (min-value -1.0)
          (max-value 1.0)
        )
        (fire
          (function-name "fire")
          (function-group $ballistic-gun-group)
          (description "Fires this weapon")
          (min-value 1.0)
          (max-value 1.0)
        )
        (change-ammo
          (function-name "change ammo")
          (function-group $ballistic-gun-group)
          (description "Changes the ammo of this weapon")
          (min-value 1.0)
          (max-value 1.0)
        )
      )
    )
  )
)
```

The meta-data section of the sysdef file specifies a value for these groups. It is specified as the keyword `weapon` followed by the name of the system – `weapon:120mm Ballistic Gun`:

```
(meta-data
  (system-name "Ballistic Gun (Elevating/Rotating)")
  (system-description "Template for a system that will allow for the
simulation Object to be equipped with a rotating, elevating mounted
ballistic weapon.")
  (allowed-state-repository-types "ground-vehicle-param"
    "surface-entity-param" "rotary-wing-entity-param")
  (system-categories "weapon")
  (parameter-data-list
    ...)
  (string-parameter-data
    (parameter-name "slew-group")
    (variable-type "DtRwString")
    (display-label "Slew Joystick Group Name")
    (display-units "")
    (source-units "")
    (list-items "")
    (list-display 0)
    (allow-edit True)
    (display-value True)
    (roll-up-rule "none")
    (default-value $default-slew-group
  )
  (string-parameter-data
    (parameter-name "ballistic-gun-group")
    (variable-type "DtRwString")
    (display-label "Ballistic Gun Joystick Group Name")
    (display-units "")
    (source-units "")
    (list-items "")
    (list-display 0)
    (allow-edit True)
    (display-value True)
    (roll-up-rule "none")
    (default-value $default-ballistic-gun-group)
  )
  ...
)
```

Other weapon systems are configured similarly. On the Joystick Configuration page (Figure 111), the Switch Between group lists weapon as an option. The weapons configured for switching all show the `switch_option:sysdef` syntax.

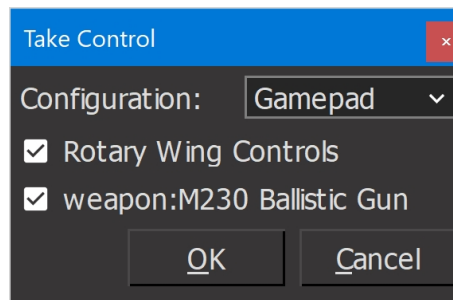
Figure 111. Switch between option

For details about using the switch between option, see "15.1.1 Switching Between Systems" on page 273.

15.3.5 Using Multiple Joysticks

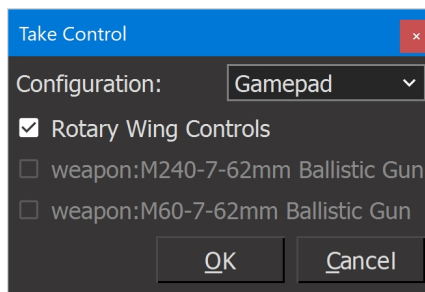
You can attach multiple joysticks or game controllers to your computer. You must create a configuration for each joystick that you want to use. When you take control of an entity, you select the configuration you want to use and the functions that you want to control with the selected configuration.

In Figure 112, the selected entity (a helicopter) has two functions that can be controlled by joystick or game controller.

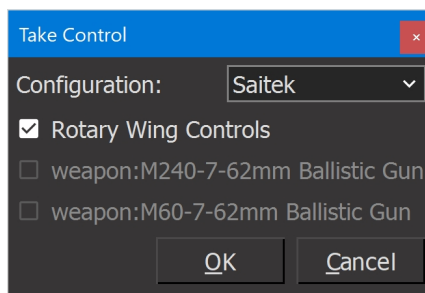
Figure 112. Take Control dialog box

You could select one configuration to control both, or you could control the two functions with different controllers:

1. Select the entity.
2. Choose **Objects > Joystick > Take Control**. The Take Control dialog box opens.
3. Clear the Rotary Wing Controls check box. This leaves the Gamepad configuration controlling the M240 ballistic gun.
4. Click OK.
5. Select the entity again and choose **Objects > Joystick > Take Control**. The Take Control dialog box opens (Figure 113). Only the Rotary Wing Controls function is available, because the ballistic gun is already under the control of the Gamepad configuration.

Figure 113. Take Control dialog box, one function available

6. In the Configuration list, select a different configuration (Figure 114).

Figure 114. Take Control dialog box, second configuration selected

7. Click OK. The rotary wing controls are now controlled by the Saitek configuration and the ballistic gun by the Gamepad configuration.

15.3.6 Specifying a Favorite Joystick Configuration

Ordinarily, when you ask to take control of an entity with a joystick, you must specify the joystick configuration to use. You can specify a configuration as the favorite. In that case the configuration is automatically applied. However, you cannot choose a different configuration.

► **To specify a favorite joystick configuration:**

1. Choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the Joystick Configuration page (Figure 104 on page 276).
3. In the Game Controller Configurations list, select the configuration that you want to specify as the favorite.
4. Click the Star button (☆). A star is added to the configuration name in the list.

15.4. Installing a Gamepad on Linux

You can use Logitech Gamepads on Linux. You configure them like you would a joystick.

► **To install a Gamepad driver on Linux:**

1. Download the xboxdrv driver for Linux.
2. Install the driver:

```
yum install xboxdrv
```

3. Test to see if your controller is recognized:

```
xboxdrv -L
```

Your device should be listed with an ID next to it.

4. To test that the device is working:

```
xboxdrv -i id
```

5. To have the device work with VR-Forces, issue the command:

```
xboxdrv --detach-kernel-driver --silent
```




16. Attaching Components to Remote Entities

This chapter explains how to configure components on remote entities.

16.1. Attaching Components to Remote Entities	288
---	-----

16.1. Attaching Components to Remote Entities

You can attach VR-Forces sensors, actuators, and controllers to remote (non-VR-Forces) entities. This feature allows you to add to the capabilities of the remote entity and better integrate it with the VR-Forces simulation. For example, you could add a radio to an entity or you could enable spot reporting. When you attach a component to a remote entity, you can write a plan for that entity that uses the attached components. For details, see "Writing Plans for Remote Entities".

VR-Forces includes an example scenario, `remoteAttachment`, that uses remote attachment. Its use requires that you run an external application, the `f18` example that comes with VR-Link. For details about how to run the scenario, please see the scenario's description. This and other example scenarios are found in `./userData/scenarios/Sample`.

When you add a component to a remote entity, the application that is simulating the entity is responsible for updating its position and health status. VR-Forces simulates the attached components. You can specify that VR-Forces simulate the components on one sim. engine or across all sim. engines.

Simulation of components on remote entities has limitations:

- Only individual platforms and lifeforms can attach components. Units and environmental process objects are not supported.
- VR-Forces components cannot control remote entities. That is, they cannot change their position or state. Therefore, you should not attach components such as movement systems to remote entities, because they will have no effect on the entities. Similarly, components that expect to be attached to a locally simulated entity, such as an articulated part, may not work properly when attached to remote entities.
- By default, remote entities are not added to the echelon hierarchy. The result is that they do not send or receive spot reports. If you want spot reporting for remote entities, set `connection-mode` to `force` in the `commModelParams.mtl` for the appropriate SMS.

Components are attached to remote entities on the basis of entries in a component attachment table. The component attachment table provided by VR-Forces is `./data/simulationModelSets/<model_set>/extra/componentAttachmentTable.mtl`.

The component attachment table associates entities with attachment key-names. The following is a sample component attachment table:

```
(remote-configurations
  (attachment-entry
    ;; CIS MIG 29
    (attach-by object-type (obj-type 1 (1 2 222 1 2 1 -1)))
    (key-name "mig29-config")
  )
  (attachment-entry
    ;; US F-18
    (attach-by object-type (obj-type 1 (1 2 225 1 9 -1 -1)))
    (key-name "f18-config")
  )
)
```

)

The entries in this table associate the key-names mig29-config and f18-config with appropriate entity type enumerations. You can also associate a key-name with marking text:

```
(attachment-entry
  (attach-by marking-text "HMMWV 1")
  (key-name "hmmwv-attachments")
)
```

The key-names in the component attachment table have corresponding entries in *vrfSim.opd* in the remote-configurations block:

```
(remote-configurations
  (remote-configuration
    (key-name "f18-config")
    (configuration-path "${remote-config-dir}/f18.mtl")
  )
  (remote-configuration
    (key-name "mig29-config")
    (configuration-path "${remote-config-dir}/mig29.mtl")
  )
)
```

The key-names are associated with specific configuration files, which by default are located in *./data/simulationModelSets/<model_set>/vrfSim/remoteConfigurations*. The configuration files contain the components that are to be attached to the remote entity. For examples, please see the files in the *./data/simulationModelSets/EntityLevel/vrfSim/remoteConfigurations* directory.